NEURO-INSPIRED COMPUTING FOR NEXT-GEN AI: COMPUTING MODEL, ARCHITECTURES AND LEARNING ALGORITHMS

EDITED BY: Angeliki Pantazi, Emre O. Neftci, Bipin Rajendran and Osvaldo Simeone
PUBLISHED IN: Frontiers in Neuroscience

frontiers Research Topics

Frontiers eBook Copyright Statement

The copyright in the text of individual articles in this eBook is the property of their respective authors or their respective institutions or funders. The copyright in graphics and images within each article may be subject to copyright of other parties. In both cases this is subject to a license granted to Frontiers.

The compilation of articles constituting this eBook is the property of Frontiers.

Each article within this eBook, and the eBook itself, are published under the most recent version of the Creative Commons CC-BY licence. The version current at the date of publication of this eBook is CC-BY 4.0. If the CC-BY licence is updated, the licence granted by Frontiers is automatically updated to the new version.

When exercising any right under the CC-BY licence, Frontiers must be attributed as the original publisher of the article or eBook, as applicable.

Authors have the responsibility of ensuring that any graphics or other materials which are the property of others may be included in the CC-BY licence, but this should be checked before relying on the CC-BY licence to reproduce those materials. Any copyright notices relating to those materials must be complied with.

Copyright and source acknowledgement notices may not be removed and must be displayed in any copy, derivative work or partial copy which includes the elements in question.

All copyright, and all rights therein, are protected by national and international copyright laws. The above represents a summary only. For further information please read Frontiers' Conditions for Website Use and Copyright Statement, and the applicable CC-BY licence.

ISSN 1664-8714
ISBN 978-2-88976-856-1
DOI 10.3389/978-2-88976-856-1

About Frontiers

Frontiers is more than just an open-access publisher of scholarly articles: it is a pioneering approach to the world of academia, radically improving the way scholarly research is managed. The grand vision of Frontiers is a world where all people have an equal opportunity to seek, share and generate knowledge. Frontiers provides immediate and permanent online open access to all its publications, but this alone is not enough to realize our grand goals.

Frontiers Journal Series

The Frontiers Journal Series is a multi-tier and interdisciplinary set of open-access, online journals, promising a paradigm shift from the current review, selection and dissemination processes in academic publishing. All Frontiers journals are driven by researchers for researchers; therefore, they constitute a service to the scholarly community. At the same time, the Frontiers Journal Series operates on a revolutionary invention, the tiered publishing system, initially addressing specific communities of scholars, and gradually climbing up to broader public understanding, thus serving the interests of the lay society, too.

Dedication to Quality

Each Frontiers article is a landmark of the highest quality, thanks to genuinely collaborative interactions between authors and review editors, who include some of the world's best academicians. Research must be certified by peers before entering a stream of knowledge that may eventually reach the public - and shape society; therefore, Frontiers only applies the most rigorous and unbiased reviews.
Frontiers revolutionizes research publishing by freely delivering the most outstanding research, evaluated with no bias from both the academic and social point of view. By applying the most advanced information technologies, Frontiers is catapulting scholarly publishing into a new generation.

What are Frontiers Research Topics?

Frontiers Research Topics are very popular trademarks of the Frontiers Journals Series: they are collections of at least ten articles, all centered on a particular subject. With their unique mix of varied contributions from Original Research to Review Articles, Frontiers Research Topics unify the most influential researchers, the latest key findings and historical advances in a hot research area! Find out more on how to host your own Frontiers Research Topic or contribute to one as an author by contacting the Frontiers Editorial Office: frontiersin.org/about/contact

NEURO-INSPIRED COMPUTING FOR NEXT-GEN AI: COMPUTING MODEL, ARCHITECTURES AND LEARNING ALGORITHMS

Topic Editors:
Angeliki Pantazi, IBM Research - Zurich, Switzerland
Emre O. Neftci, University of California, Irvine, United States
Bipin Rajendran, King's College London, United Kingdom
Osvaldo Simeone, King's College London, United Kingdom

Citation: Pantazi, A., Neftci, E. O., Rajendran, B., Simeone, O., eds. (2022). Neuro-inspired Computing for Next-gen AI: Computing Model, Architectures and Learning Algorithms. Lausanne: Frontiers Media SA. doi: 10.3389/978-2-88976-856-1

Table of Contents

TYPE Editorial
PUBLISHED 25 July 2022
DOI 10.3389/fnins.2022.974627

Check for updates

OPEN ACCESS

EDITED AND REVIEWED BY
André van Schaik,
Western Sydney University, Australia

*CORRESPONDENCE
Angeliki Pantazi
agp@zurich.ibm.com

SPECIALTY SECTION
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

RECEIVED 21 June 2022
ACCEPTED 28 June 2022
PUBLISHED 25 July 2022

CITATION
Pantazi A, Rajendran B, Simeone O and Neftci E (2022) Editorial: Neuro-inspired computing for next-gen AI: Computing model, architectures and learning algorithms.
Front. Neurosci. 16:974627.
doi: 10.3389/fnins.2022.974627

COPYRIGHT
© 2022 Pantazi, Rajendran, Simeone and Neftci. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Editorial: Neuro-inspired computing for next-gen AI: Computing model, architectures and learning algorithms

Angeliki Pantazi[1]*, Bipin Rajendran[2], Osvaldo Simeone[2] and Emre Neftci[3]

[1]IBM Research - Zurich, Switzerland, [2]Department of Engineering, King's College London, London, United Kingdom, [3]Department of Cognitive Sciences, University of California, Irvine, Irvine, CA, United States

KEYWORDS

neuromorphic computing, Spiking Neural Networks, neuronal model, learning algorithms, neuromorphic hardware

Editorial on the Research Topic
Neuro-inspired Computing for Next-gen AI: Computing Model, Architectures and Learning Algorithms

Introduction

Today's advances in Artificial Intelligence (AI) have been primarily driven by deep learning and have led to astounding progress in several tasks such as image classification, multiple object detection, language translation, speech recognition and even in the ability to play strategic games. However, the AI systems of today have several limitations. Specifically, the hardware infrastructure is limited to high-power and large-scale processing systems that are based on the von Neumann computing paradigm. Moreover, there is a growing demand for applications with cognitive functionality that will be able to operate in real time and in an autonomous manner in the field. The limitations of contemporary AI systems are in stark contrast to the capabilities of the brain which can learn and adapt very quickly consuming just about 20 W of power.

Neuromorphic computing, inspired by neuroscience, is a promising path toward the next-generation AI systems. The research focuses on different levels of the design stack, i.e., the computing model, the architecture and the learning algorithms. The computing model is based on Spiking Neural Networks (SNNs), which possess more biologically realistic neuronal dynamics as compared to those of Artificial Neural Networks (ANNs). At the architectural level, SNNs implement in-memory computing, which is well suited for efficient SNN hardware realizations. At the algorithmic level, neuro-inspired learning paradigms are based on the insight that the brain continuously processes incoming information and is able to adapt to changing conditions. Thus, online learning, learning-to-learn, and unsupervised learning provide the main conceptual platforms for the design of low-power, accurate and reliable neuromorphic computing systems.

This Research Topic provides an overview of the recent advances on computing models, architecture, and learning algorithms for neuromorphic computing. In the rest of this Editorial, we provide a brief description of the accepted papers contributing to each of these areas.

Computing model

State-of-the-art deep learning is based on ANNs that only take inspiration from biology to a very limited extent—primarily in terms of the ANNs' networked structure. This has several drawbacks, especially in terms of power consumption and energy efficiency. More biologically realistic neural models have been considered as promising contenders for the next generation of neural networks. In this Research Topic, Dellaferrera et al. present a biologically-inspired computational model for blind source decomposition based on a two-compartment somatodendritic neuron and synaptic connections trained by Hebbian-like learning. Their results demonstrate blind source separation on a sequence of mixtures of acoustic stimuli, suggesting that the proposed neuronal model can capture characteristics of the brain's segregation capability. Delacour and Todri-Sanial present an emerging neuromorphic architecture in which neurons are represented with oscillators, and the information is encoded in the oscillator's phase relations. They present an oscillatory neural network (ONN) using relaxation oscillators based on VO_2 material. They demonstrate that an ONN consisting of 60 fully-connected oscillator neurons can implement a Hopfield Neural Network that performs pattern recognition.

Architectures

The network structure in biological systems provides energy efficiency and low latency, while combining memory and computation. In recent years, ANN-to-SNN conversion techniques have enabled the design of SNNs, starting from well-known ANN architectures, that offer lower computation cost compared to their non-spiking counterparts. Moreover, the concept of in-memory computing, which aims at co-locating the memory and processing units, has recently demonstrated that substantial acceleration may be achieved for ANNs. In this Research Topic, Wu et al. propose a framework for developing an energy-efficient SNN using a novel explicit current control (ECC) method that converts a CNN to an SNN. The key contribution of this framework is that, during the conversion, multiple objectives are considered including accuracy, latency, and energy efficiency. Zou et al. present a hardware-friendly algorithm that converts a quantized ANN to an SNN by minimizing the spike approximation errors that are typically emerging in ANN-to-SNN conversion. Furthermore, they develop strategies for mapping the designed CNN to crossbar-based neuromorphic hardware.

Yan et al. propose a sparsity-driven SNN learning algorithm (BPSR) that incorporates spiking regularization to minimize the neuronal spiking rate. To further mitigate the redundancy of the network structure, they suggest a rewiring mechanism with synaptic regularization. The proposed BPSR scheme improves the spiking and synaptic sparsity while achieving comparable accuracy with related works. Finally, Datta et al. propose a deep SNN for 3D image recognition using algorithmic and hardware co-design approaches, namely quantization-aware backpropagation and processing-in-memory (PIM) architecture. Their results yield low latency (5 time steps) and low bit width (6-bit weights). The adoption of the PIM architecture in the first layer further improves the average energy, delay, and energy-delay-product.

Learning algorithms

The brain is equipped with impressive learning capabilities, enabling animals to dynamically adapt to the surrounding world. Hebbian learning and Spike-Timing Dependent Plasticity (STDP) are commonly employed learning rules in neuro-inspired models. The convergence properties and computational characteristics remain largely unknown. In this Research Topic, Chakraborty and Mukhopadhyay study the generalizability properties of SNNs equipped with STDP. They achieve this goal by analyzing the dimensionality of the space spanned by the learning process, and propose a method to optimize hyperparameters to improve the network generalization properties.

Neuro-inspired computing has recently taken inspiration from machine learning to implement online learning rules based on gradient descent. These generally transferred the basic modules of deep learning, but lag behind in other components essential to deep learning. One of these components is batch normalization, which is now ubiquitous in deep learning to improve convergence speed and accuracy. In this Research Topic, Kim and Panda showed how batch normalization can be adapted to SNNs, thereby enabling significant acceleration in SNN training.

Compared to gradient descent, STDP has the advantage that it does not require external supervision, and can therefore operate locally in neuromorphic hardware. However, gradient descent generally performs better if such signals exists. In their proposed model, Krithivasan et al. selectively adjust the learning rules employed by the layer during training to exploit the best of STDP and SGD. In an associative learning framework, Mo et al. use external supervision to improve the performance of STDP, and demonstrate successful STDP learning in common labeled machine learning datasets.

Author contributions

All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

Conflict of interest

AP was employed by IBM Research.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

BR's research is partly funded by Intel, Cisco, European Space Agency, and Semiconductor Research Corporation outside of this work. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

ORIGINAL RESEARCH
published: 29 October 2021
doi: 10.3389/fnins.2021.695357

Characterization of Generalizability of Spike Timing Dependent Plasticity Trained Spiking Neural Networks

Biswadeep Chakraborty and Saibal Mukhopadhyay*

Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, United States

A Spiking Neural Network (SNN) is trained with Spike Timing Dependent Plasticity (STDP), which is a neuro-inspired unsupervised learning method for various machine learning applications. This paper studies the generalizability properties of the STDP learning processes using the Hausdorff dimension of the trajectories of the learning algorithm. The paper analyzes the effects of STDP learning models and associated hyper-parameters on the generalizability properties of an SNN. The analysis is used to develop a Bayesian optimization approach to optimize the hyper-parameters for an STDP model for improving the generalizability properties of an SNN.

Keywords: spiking neural networks, leaky integrate and fire, generalization, Hausdorff dimension, logSTDP, addSTDP, multSTDP, Bayesian optimization

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
David Kappel,
Technische Universität Dresden,
Germany
Gopalakrishnan Srinivasan,
Apple, United States

****Correspondence:***
Biswadeep Chakraborty
biswadeep@gatech.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *14 April 2021*
Accepted: *29 September 2021*
Published: *29 October 2021*

Citation:
Chakraborty B and Mukhopadhyay S (2021) Characterization of Generalizability of Spike Timing Dependent Plasticity Trained Spiking Neural Networks. Front. Neurosci. 15:695357. doi: 10.3389/fnins.2021.695357

1. INTRODUCTION

A Spiking Neural Network (SNN) (Maass, 1997; Gerstner and Kistler, 2002b; Pfeiffer and Pfeil, 2018) is a neuro-inspired machine learning (ML) model that mimics the spike-based operation of the human brain (Bi and Poo, 1998). The Spike Timing Dependent Plasticity (STDP) is a policy for unsupervised learning in SNNs (Bell et al., 1997; Magee and Johnston, 1997; Gerstner and Kistler, 2002a). The STDP relates the expected change in synaptic weights to the timing difference between post-synaptic spikes and pre-synaptic spikes (Feldman, 2012). Recent works using STDP trained SNNs have demonstrated promising results as an unsupervised learning paradigm for various tasks such as object classification and recognition (Masquelier et al., 2009; Diehl and Cook, 2015; Kheradpisheh et al., 2018; Lee et al., 2018; Mozafari et al., 2019; She et al., 2021).

Generalizability is a measure of how well an ML model performs on test data that lies outside of the distribution of the training samples (Kawaguchi et al., 2017; Neyshabur et al., 2017). The generalization properties of Stochastic Gradient Descent (SGD) based training for deep neural network (DNN) have received significant attention in recent years (Allen-Zhu et al., 2018; Allen-Zhu and Li, 2019; Poggio et al., 2019). The dynamics of SGD have been studied via models of stochastic gradient Langevin dynamics with an assumption that gradient noise is Gaussian (Simsekli et al., 2020b). Here SGD is considered to be driven by a Brownian motion. Chen et al. showed that SGD dynamics commonly exhibits rich, complex dynamics when navigating through the loss landscape (Chen et al., 2020). Recently Gurbuzbalaban et al. (2020) and Hodgkinson and Mahoney (2021) have simultaneously shown that the law of the SGD iterates can converge to a heavy-tailed stationary distribution with infinite variance when the step-size η is large and/or the batch-size B is small. These results form a theoretical basis for the origins of the observed heavy-tailed behavior of SGD in practice. The authors proved generalization bounds for SGD under the assumption that its trajectories can be well-approximated by the Feller Process (Capocelli and Ricciardi, 1976), a Markov-based stochastic process. Modeling the trajectories of

SGD using a stochastic differential equation (SDE) under heavy-tailed gradient noise has shed light on several interesting characteristics of SGD.

In contrast, the generalizability analysis of STDP trained SNNs, although important, has received much less attention. Few studies have shown that, in general, the biological learning process in the human brain has significantly good generalization properties (Sinz et al., 2019; Zador, 2019). However, none of them have characterized the generalization of an STDP-trained SNN using a mathematical model. There is little understanding of how hyperparameters of the STDP process impact the generalizability of the trained SNN model. Moreover, the generalization of STDP cannot be characterized by directly adopting similar studies for SGD. For example, SGD has been modeled as a Feller process for studying generalizability.

Rossum et al. showed that random variations arise due to the variability in the amount of potentiation (depression) between the pre-and post-synaptic events at fixed relative timing (Van Rossum et al., 2000). At the neuron level, fluctuations in relative timing between pre-and post-synaptic events also contribute to random variations (Roberts and Bell, 2000). For many STDP learning rules reported in the literature, the dynamics instantiate a Markov process (Bell et al., 1997; Markram et al., 1997; Bi and Poo, 1998; Han et al., 2000; Van Rossum et al., 2000); changes in the synaptic weight depend on the learning rule only on the current weight and a set of random variables that determine the transition probabilities. However, recent literature has shown that weight update using STDP is better modeled as an Ornstein-Uhlenbeck process (Câteau and Fukai, 2003; Legenstein et al., 2008; Aceituno et al., 2020).

As described by Camuto et al. (2021), fractals are complex patterns, and the level of this complexity is typically measured by the Hausdorff dimension (HD) of the fractal, which is a notion of dimension. Recently, assuming that SGD trajectories can be well-approximated by a Feller Process, it is shown that the generalization error,which is the difference between the training and testing accuracy, can be controlled by the Hausdorff dimension of the trajectories of the SDE Simsekli et al. (2020a). That is, the ambient dimension that appears in classical learning theory bounds is replaced with the Hausdorff dimension. The fractal geometric approach presented by Simsekli et al. can capture the low dimensional structure of fractal sets and provides an alternative perspective to the compression-based approaches that aim to understand why over parametrized networks do not overfit.

This paper presents a model to characterize the generalizability of the STDP process and develops a methodology to optimize hyperparameters to improve the generalizability of an STDP-trained SNN. We use the fact that the sample paths of a Markov process exhibit a fractal-like structure (Xiao, 2003). The generalization error over the sample paths is related to the roughness of the random fractal generated by the driving Markov process which is measured by the Hausdorff dimension (Simsekli et al., 2020a) which is in turn dependent on the tail behavior of the driving process. The objective of the paper is to get a model which is more generalizable in the sense that the performance of the network on unknown datasets should not differ much from its performance in the training dataset. It is to be noted that in this paper we are using the generalization error as the metric of generalizability of the network. Generalization error is not a measure of absolute accuracy, but rather the difference between training and test accuracy.

Normally, the validation loss of a model on a testing set is used to characterize the accuracy of that model. However, the validation loss is dependent on the choice of the test set, and does not necessarily give a good measure of the generalization of the learning process. Therefore, generalization error in a model is generally measured normally measured by comparing the difference between training and testing accuracy - a more generalizable model has less difference between training and testing accuracy (Goodfellow et al., 2016). However, such a measure of generalization error requires computing the validation loss (i.e., testing accuracy) for a given test dataset. To optimize the generalizability of the model, we need an objective function that measures the generalizability of the learning process. If the validation loss is used as a measure, then for each iteration of the optimization, we need to compute this loss by running the model over the entire dataset, which will significantly increase the computation time. We use Hausdorff Dimension as a measure of generalizability of the STDP process to address the preceding challenges. First, the Hausdorff measure characterizes the fractal nature of the sample paths of the learning process itself and does not depend on the testing dataset. Hence, HD provides a better (i.e., test set independent) measure of the generalization of the learning process. Second, HD can be computed during the training process itself and does not require running inference on the test data set. This reduces computation time per iteration of the optimization process.

We model the STDP learning as an Ornstein-Uhlenbeck process which is a Markov process and show that the generalization error is dependent on the Hausdorff dimension of the trajectories of the STDP process. We use the SDE representation of synaptic plasticity and model STDP learning as a stochastic process that solves the SDE.

Using the Hausdorff dimension we study the generalization properties of an STDP trained SNN for image classification. We compare three different STDP processes, namely, log-STDP, add-STDP, and mult-STDP, and show that the log-STDP improves generalizability. We show that modulating hyperparameters of the STDP learning rule and learning rate changes the generalizability of the trained model. Moreover, using log-STDP as an example, we show the hyperparameter choices that reduce generalization error increases the convergence time, and training loss, showing a trade-off between generalizability and the learning ability of a model. Motivated by the preceding observations, we develop a Bayesian optimization technique for determining the optimal set of hyperparameters which gives an STDP model with the least generalization error. We consider an SNN model with 6,400 learning neurons trained using the log-STDP process. Optimizing the hyperparameters of the learning process using Bayesian Optimization shows a testing accuracy of 90.65% and a generalization error of 3.17 on the MNIST dataset. This shows a mean increase of almost 40% in generalization performance for a mean drop of about 1% in testing accuracy in comparison to randomly initialized training hyperparameters. In order to further evaluate the learning methodologies, we also evaluated them on the more complex Fashion MNIST dataset and observed a similar trend.

2. MATERIALS AND METHODS

2.1. Background

2.1.1. Spiking Neural Networks

We chose the leaky integrate-and-fire model of a neuron where the membrane voltage X is described by

$$\tau \frac{dX}{dt} = (E_{\text{rest}} - X) + g_e (E_{\text{exc}} - X) + g_i (E_{\text{inh}} - X)$$

where E_{rest} is the resting membrane potential; E_{exc} and E_{inh} are the equilibrium potentials of excitatory and inhibitory synapses, respectively; and g_e and g_i are the conductances of excitatory and inhibitory synapses, respectively. The time constant τ, is longer for excitatory neurons than for inhibitory neurons. When the neuron's membrane potential crosses its membrane threshold, the neuron fires, and its membrane potential is reset. Hence, the neuron enters its refractory period and cannot spike again for the duration of the refractory period.

Synapses are modeled by conductance changes, i.e., synapses increase their conductance instantaneously by the synaptic weight w when a pre-synaptic spike arrives at the synapse, otherwise, the conductance decays exponentially. Thus, the dynamics of the conductance g can be written as:

$$\tau_g \frac{dg}{dt} = -g \tag{1}$$

If the pre-synaptic neuron is excitatory, the dynamics of the conductance is $g = g_e$ with the time constant of the excitatory post-synaptic potential being $\tau_g = \tau_{g_e}$. On the other hand, if the pre-synaptic neuron is inhibitory, it's synaptic conductance is given as $g = g_i$ and the time constant of the inhibitory post-synaptic potential as $\tau_g = \tau_{g_i}$.

2.1.2. STDP Based Learning Methods

Spike-timing-dependent plasticity is a biologically plausible learning model representing the time evolution of the synaptic weights as a function of the past spiking activity of adjacent neurons.

In a STDP model, the change in synaptic weight induced by the pre-and post-synaptic spikes at times t_{pre}, t_{post} are defined by:

$$\Delta W = \eta(1 + \zeta) H\left(W; t_{\text{pre}} - t_{\text{post}}\right) \tag{2}$$

where the learning rate η determines the speed of learning. The Gaussian white noise ζ with zero mean and variance σ^2 describes the variability observed in physiology. The function $H(W; u)$ describes the long term potentiation (LTP) and depression (LTD) based on the relative timing of the spike pair within a learning window $u = t_{\text{pre}} - t_{\text{post}}$, and is defined by:

$$H(W; u) = \begin{cases} a_+(W) \exp\left(-\frac{|u|}{\tau_+}\right) & \text{for u} < 0 \\ -a_-(W) \exp\left(-\frac{|u|}{\tau_-}\right) & \text{for u} > 0 \end{cases} \tag{3}$$

The shape of the weight distribution produced by STDP can be adjusted via the scaling functions $a_{\pm}(W)$ in (3) that determine the weight dependence. We study three different types of STDP processes, namely, add-STDP, mult-STDP, and log-STDP. All STDP models follow the Equations (2) and (3), however, they have different scaling functions ($a_{\pm}$) in (3) as discussed below. The weight distributions of these three different STDP processes at the end of the last training iteration are shown in **Figure 1**. At the beginning of the training iterations, the distribution is uniform for all three reflecting on the weight initialization conditions. Additive STDP gives rise to strong competition among synapses, but due to the absence of weight dependence, it requires hard boundaries to secure the stability of weight dynamics. To reach a stability point for the add-STDP, we followed the analysis done by Gilson and Fukai (2011) and Burkitt et al. (2004) and chose the fixed point $W_0 = 0.006$. **Figure 1** approximates the probability density function based on the weight distributions of the different STDP models. We are using a Gaussian KDE to get this pdf from the empirical weight distribution obtained. Given N independent realizations $\mathcal{X}_N \equiv \{X_1, \ldots, X_N\}$ from an unknown continuous probability density function (p.d.f.) f on $\mathcal{X}$, the Gaussian kernel density estimator is defined as

$$\hat{f}(x; t) = \frac{1}{N} \sum_{i=1}^{N} \phi\left(x, X_i; t\right), \quad x \in \mathbb{R} \tag{4}$$

where

$$\phi\left(x, X_i; t\right) = \frac{1}{\sqrt{2\pi t}} e^{-(x-X_i)^2/(2t)} \tag{5}$$

is a Gaussian p.d.f. (kernel) with location X_i and scale $\sqrt{t}$. The scale is usually referred to as the bandwidth. Much research has been focused on the optimal choice of t, because the performance of $\hat{f}$ as an estimator of f depends crucially on its value (Sheather and Jones, 1991; Jones et al., 1992).

Logarithmic STDP (log-STDP) (Gilson and Fukai, 2011). The scaling functions of log-STDP is defined by:

$$a_+(W) = c_+ \exp\left(-W/W_0\gamma\right) \tag{6}$$

$$a_-(W) = \begin{cases} c_- W/W_0 & \text{for W} \le W_0 \\ c_- \left[1 + \frac{\ln\left[1+\mathcal{S}\left(\frac{W}{W_0}-1\right)\right]}{\mathcal{S}}\right] & \text{for W} > W_0 \end{cases} \tag{7}$$

In this study, W_0 is chosen such that LTP and LTD in log-STDP balance each other for uncorrelated inputs, namely $A\left(W_0\right) = \tau_+ a_+\left(W_0\right) - \tau_- a_-\left(W_0\right) \simeq 0$. Therefore, W_0 will also be referred to as the "fixed point" of the dynamics. Since depression increases sublinearly (blue solid curve for a_- in **Figure 1**), noise in log-STDP is weaker than that for mult-STDP for which depression increases linearly (orange dashed curve for a_- in **Figure 1**).

The weight dependence for LTD in logSTDP is similar to mult-STDP for $W \le W_0$, i.e., it increases linearly with W. However, the LTD curve a_- becomes sublinear for $W \ge W_0$, and $\mathcal{S}$ determines the degree of the log-like saturation. For larger

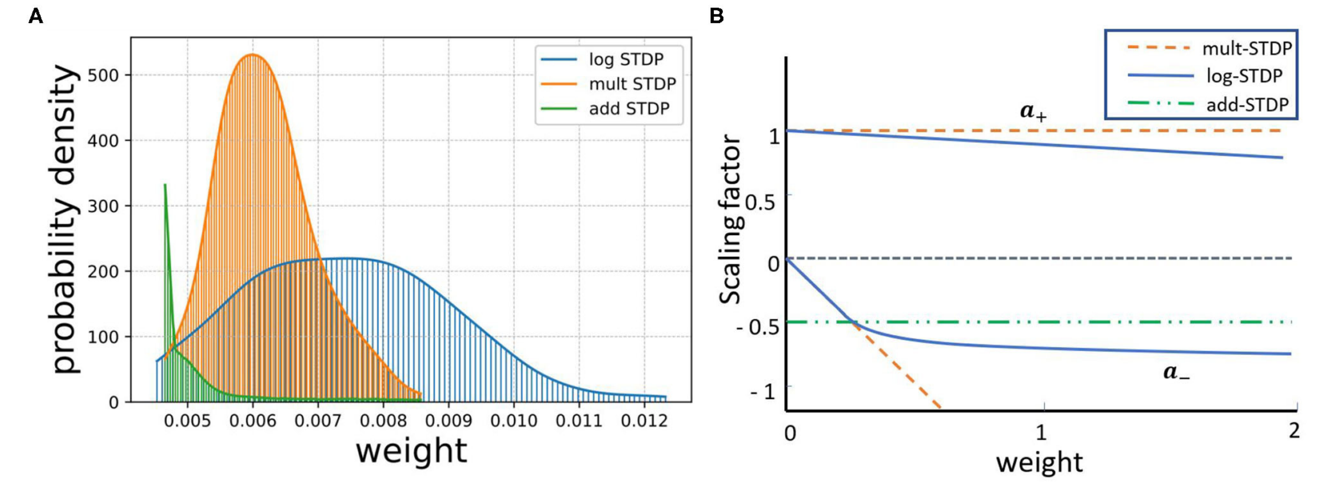


FIGURE 1 | (A) Resulting weight distribution for log-STDP (Gilson and Fukai, 2011);multSTDP (Van Rossum et al., 2000) and add-STDP (Song et al., 2000). **(B)** Plot for Functions a_+ for LTP and $-f_-$ for LTD in log-STDP (blue solid curve); mult-STDP (orange dashed line); and add-STDP model green dashed-dotted curve for depression and orange dashed curve for potentiation.

$\mathcal{S}$, LTD has a more pronounced saturating log-like profile and the tail of the synaptic weight distribution extends further.

We choose the function a_+ for LTP to be roughly constant around W_0, such that the exponential decay controlled by γ only shows for $W >> W_0$. Thus, the scaling functions $a_+, a_-, \mathcal{S}$, and γ are the hyperparameters that can be tuned to model an efficient log-STDP learning model. tAs discussed by Gilson and Fukai (2011), we choose the activation function a_+ to be roughly constant around the threshold fixed-point weight W_0. For $W \geq W_0$, the exponential decay factor γ of log-STDP coincides with mlt-STDP when $\mathcal{S} \to 0$ and $\gamma \to \infty$. Log-STDP tends toward add-STDP when $\mathcal{S} \to \infty$ and $\gamma \to \infty$. High values of $\mathcal{S}$ leads to stronger saturation of LTD and large values of γ leads to a stronger potentiation and keeps LTP almost constant which leads to favoring a winner-takes-all behavior.

Additive STDP (add-STDP) (Song et al., 2000). It is weight independent and is defined by the following scaling functions:

$$\begin{aligned} a_+(W) &= c_+ \\ a_-(W) &= c_- \end{aligned} \tag{8}$$

with $c_+\tau_+ < c_-\tau_-$ such that LTD overpowers LTP. The drift due to random spiking activity thus causes the weights to be depressed toward zero, which provides some stability for the output firing rate (Gilson and Fukai, 2011). For the simulations, we are using a fast learning rate that is synonymous to a high level of noise, and more stability. This requires a stronger depression. Thus, we use $c_+ = 1$ and $c_- = 0.6$.

Multiplicative STDP (mult-STDP) (Van Rossum et al., 2000). The multiplicative STDP has a linear weight dependence for LTD and constant LTP:

$$a_+(W) = c_+ \tag{9}$$

$$a_-(W) = c_- W \tag{10}$$

The equilibrium mean weight is then given by $W^*_{\text{av}} = c_+\tau_+ / (c_-\tau_-)$. For the simulations we use $c_+ = 1$ and $c_- = 0.5/W_0 = 2$. This calibration corresponds to a similar neuronal output firing rate to that for log-STDP in the case of uncorrelated inputs.

2.1.3. Generalization - Hausdorff Dimension and Tail Index Analysis

Recent works have discussed the generalizability of SGD based on the trajectories of the learning algorithm. Simsekli et al. (2020a) identified the complexity of the fractals generated by a Feller process that approximates SGD. The intrinsic complexity of a fractal is typically characterized by a notion called the Hausdorff dimension (Le Guével, 2019; Lőrinczi and Yang, 2019), which extends the usual notion of dimension to fractional orders. Due to their recursive nature, Markov processes often generate random fractals (Xiao, 2003). Significant research has been performed in modern probability theory to study the structure of such fractals (Khoshnevisan, 2009; Bishop and Peres, 2017; Khoshnevisan and Xiao, 2017; Yang, 2018). Thus, the STDP learning method follows an Ornstein-Uhlenbeck (O-U) process which is a special type of Lévy process. Again, the Hausdorff Dimension measures the roughness of the fractal patterns of the sample paths generated by the stochastic process which is measured using the tail properties of the Lévy measure of the O-U process. Lévy measure is a Borel measure on $\mathbb{R}^d \backslash \{0\}$ satisfying $\int_{\mathbb{R}^d} \|x\|^2 / \left(1 + \|x\|^2\right) \nu(dx) < \infty$. The Ornstein-Uhlenbeck process which is a Lévy process can thus be characterized by the triplet (b, Σ, ν) where $b \in \mathbb{R}^d$ denotes a constant drift, $\Sigma \in \mathbb{R}^{d\times d}$ is a positive semi-definite matrix and ν is the Lévy measure as defined above. Thus, taking Lévy processes as stochastic objects, their sample path behavior can be characterized by the Hausdorff dimension which is in turn measured using the BG-indices. Thus, the generalization properties of the STDP process can be modeled using the Hausdorff dimension of the sample paths of the O-U process. We formally define the Hausdorff dimension for the Ornstein-Uhlenbeck process modeling the STDP learning process in section 3.2.

2.2. STDP as a Stochastic Process

In this paper, we evaluate the generalization properties of an STDP model using the concept of the Hausdorff dimension. In this section, we discuss the learning methodology of STDP and how the plasticity change can be modeled using a stochastic differential equation. The state of a neuron is usually represented by its membrane potential X which is a key parameter to describe the cell activity. Due to external input signals, the membrane potential of a neuron may rise until it reaches some threshold after which a spike is emitted and transferred to the synapses of neighboring cells. To take into account the important fluctuations within cells, due to the spiking activity and thermal noise, in particular, a random component in the cell dynamics has to be included in mathematical models describing the membrane potential evolution of both the pre-and post-synaptic neurons similar to the analysis shown by Robert and Vignoud (2020). Several models take into account this random component using an independent additive diffusion component, like Brownian motion, of the membrane potential X. In our model of synaptic plasticity, the stochasticity arises at the level of the generation of spikes. When the value of the membrane potential of the output neuron is at $X = x$, a spike occurs at rate $\beta(x)$ where β is the activation function (Chichilnisky, 2001). In particular, we consider the instants when the output neuron spikes are represented by an inhomogeneous Poisson process as used by Robert and Vignoud (2020). Thus, in summary, (1) The pre-synaptic spikes are modeled using a Poisson process and hence, there is a random variable added to membrane potential. (2) The post-synaptic spikes are generated using a stochastic process based on the activation function. Hence, STDP, which depends on the pre-and post-synaptic spike times can be modeled using a *stochastic differential equation (SDE)*. Hence, we formulate the STDP as a SDE. The SDE of a learning algorithm shares similar convergence behavior of the algorithm and can be analyzed more easily than directly analyzing the algorithm.

2.2.1. Mathematical Setup

We consider the STDP as an iterative learning algorithm $\mathcal{A}$ which is dependent on the dataset $\mathcal{D}$ and the algorithmic stochasticity $\mathcal{U}$. The learning process $\mathcal{A}(\mathcal{D},\mathcal{U})$ returns the entire evolution of the parameters of the network in the time frame $[0, T]$ where $[\mathcal{A}(\mathcal{D},\mathcal{U})]_t = W_t$ being the parameter value returned by the STDP learning algorithm at time t. So, for a given training set $\mathcal{D}$, the learning algorithm $\mathcal{A}$ will output a random process $w_{t\in[0,T]}$ indexed by time which is a trajectory of iterates.

In the remainder of the paper, we will consider the case where the STDP learning process $\mathcal{A}$ is chosen to be the trajectories produced by the Ornstein-Uhlenbeck (O-U) process $\mathrm{W}^{(\mathcal{D})}$, whose symbol depends on the training set $\mathcal{D}$. More precisely, given $\mathcal{D} \in \mathcal{Z}^n$, the output of the training algorithm $\mathcal{A}(\mathcal{D},\cdot)$ will be the random mapping $t \mapsto \mathrm{W}_t^{(\mathcal{D})}$, where the symbol of $\mathrm{W}^{(\mathcal{D})}$ is determined by the drift $b_{\mathcal{D}}(w)$, diffusion matrix $\Sigma_{\mathcal{D}}(w)$, and the Lévy measure $\nu_{\mathcal{D}}(w,\cdot)$, which all depend on $\mathcal{U}$. In this context, the random variable $\mathcal{U}$ represents the randomness that is incurred by the O-U process. Finally, we also define the collection of the parameters given in a trajectory, as the image of $\mathcal{A}(\mathcal{D})$, i.e., $\mathcal{W}_{\mathcal{D}} : = \left\{w \in \mathbb{R}^d : \exists t \in [0,1], w = [\mathcal{A}(\mathcal{D})]_t\right\}$ and the collection of all possible parameters as the union $\mathcal{W} : = \bigcup_{n\geq 1} \bigcup_{\mathcal{D}\in\mathcal{Z}^n} \mathcal{W}_S$. Note that $\mathcal{W}$ is still random due to its dependence on $\mathcal{U}$.

We consider the dynamics of synaptic plasticity as a function of the membrane potential $X(t)$ and the synaptic weight $W(t)$.

2.2.2. Time Evolution of Synaptic Weights and Plasticity Kernels

As described by Robert and Vignoud (2020), the time evolution of the weight distribution $W(t)$ depends on the past activity of the input and output neurons. It may be represented using the following differential equation:

$$\frac{dW(t)}{dt} = M(\Omega_p(t), \Omega_d(t), W(t)) \tag{11}$$

where $\Omega_p(t), \Omega_d(t)$ are two non-negative processes where the first one is associated with potentiation i.e., increase in W and the latter is related to the depression i.e., decrease in W. The function M needs to be chosen such that the synaptic weight W stays at all-time in its definition interval K_W. The function M can thus be modified depending on the type of implementation of STDP that is needed. Further details regarding the choice of M for different types of STDP is discussed by Robert and Vignoud (2020).

When the synaptic weight of a connection between a pre-synaptic neuron and a post-synaptic neuron is fixed and equal to W, the time evolution of the post-synaptic membrane potential $X(t)$ is represented by the following stochastic differential equation (SDE) (Robert and Vignoud, 2020):

$$\mathrm{d}X(t) = -\frac{1}{\tau}X(t)\mathrm{d}t + W\mathcal{N}_\lambda(\mathrm{d}t) - g(X(t-))\mathcal{N}_{\beta,X}(\,\mathrm{d}t) \tag{12}$$

where $X(t-)$ is the left limit of X at $t > 0$, and τ is the exponential decay time constant of the membrane potential associated with the leaking mechanism. The sequence of firing instants of the pre-synaptic neuron is assumed to be a Poisson point process $\mathcal{N}_\lambda$ on $\mathbb{R}_+$ with the rate λ. At each pre-synaptic spike, the membrane potential X is increased by the amount W. If $W > 0$ the synapse is said to be excitatory, whereas for $W < 0$ the synapse is inhibitory. The sequence of firing instants of the post-synaptic neuron is an inhomogeneous Poisson point process $\mathcal{N}_{\beta,X}$ on $\mathbb{R}_+$ whose intensity function is $t \mapsto \beta(X(t-))$. The drop of potential due to a post-synaptic spike is represented by the function g. When the post-synaptic neuron fires in-state $X(t-) = x$, its state $X(t)$ just after the spike is $x - g(x)$.

2.2.3. Uniform Hausdorff Dimension

The Hausdorff dimension for the training algorithm $\mathcal{A}$ is a notion of complexity based on the trajectories generated by $\mathcal{A}$. Recent literature has shown that the synaptic weight update using an STDP rule can be approximated using a type of stochastic process called the Ornstein-Uhlenbeck process which is a type of Markov process (Câteau and Fukai, 2003; Legenstein et al., 2008; Aceituno et al., 2020). Hence, we can infer that the STDP process will also have a uniform Hausdorff dimension for the trajectories.

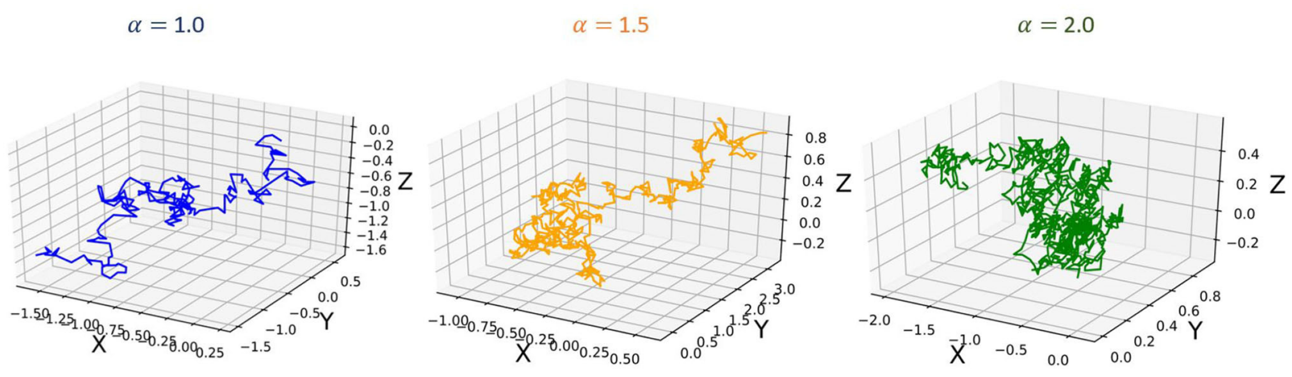


FIGURE 2 | Plot showing the trajectories of the α−stable Lévy process L_t^α for varying values of α.

We use the Hausdorff Dimension of the sample paths of the STDP based learning algorithm which has not been investigated in the literature.

Let Φ be the class of functions $\varphi : (0, \delta) \rightarrow (0, \infty)$ which are right continuous, monotone increasing with $\varphi(0+) = 0$ and such that there exists a finite constant $K > 0$ such that

$$\frac{\varphi(2s)}{\varphi(s)} \leq K, \quad \text{for } 0 < s < \frac{\delta}{2} \tag{13}$$

A function φ in Φ is often called a measure function. For $\varphi \in \Phi$, the φ-Hausdorff measure of $E \subseteq \mathbb{R}^d$ is defined by

$$\varphi_m(E) = \lim_{\varepsilon \to 0} \inf \left\{ \sum_i \varphi\left(2r_i\right) : E \subseteq \bigcup_{i=1}^{\infty} B\left(x_i, r_i\right), r_i < \varepsilon \right\} \tag{14}$$

where $B(x, r)$ denotes the open ball of radius r centered at x. The sequence of balls satisfying Equation (14) is called an ε -covering of E. We know that φ_m is a metric outer measure and every Borel set in $\mathbb{R}^d$ is φ_m measurable. Thus, the function $\varphi \in \Phi$ is called the Hausdorff measure function for E if $0 < \varphi_m(E) < \infty$.

It is to be noted here that in Equation (14), we only use coverings of E by balls, hence φ_m is usually called a spherical Hausdorff measure in the literature. Under Equation (13), φ_m is equivalent to the Hausdorff measure defined by using coverings by arbitrary sets. The Hausdorff dimension of E is defined by

$$\dim_{\mathrm{H}} E = \inf\left\{\alpha > 0 : s^\alpha - m(E) = 0\right\}. \tag{15}$$

The STDP learning process is modeled using the SDEs for the temporal evolution of the synaptic weights and the membrane potential given in Equations (11), (12). Considering the empirical observation that STDP exhibits a diffusive behavior around a local minimum (Baity-Jesi et al., 2018), we take $w_{\mathcal{D}}$ to be the local minimum found by STDP and assume that the conditions of Proposition hold around that point. This perspective indicates that the generalization error can be controlled by the BG index $\beta_{\mathcal{D}}$ of the Lévy process defined by $\psi_S(\xi)$; the sub-symbol of the process (11) around $w_{\mathcal{D}}$. The choice of the SDE (11) imposes some structure on $\psi_{\mathcal{D}}$, which lets us express $\beta_{\mathcal{D}}$ in a simpler form. This helps us in estimating the BG index for a general Lévy process. As shown by Simsekli et al., there is a layer-wise variation of the tail-index of the gradient noise in a DNN-based multi-layer neural network (Simsekli et al., 2019). Thus, for our STDP model we assume that around the local minimum $w_{\mathcal{D}}$, the dynamics of STDP will be similar to the Lévy motion with frozen coefficients: $\Sigma_2\left(w_S\right) \mathrm{L}^{\alpha(w_{\mathcal{D}})}$. Also assuming the coordinates corresponding to the same layer l have the same tail-index α_l around $w_{\mathcal{D}}$, the BG index can be analytically computed as $\beta_{\mathcal{D}} = \max_l \alpha_l \in (0, 2]$ (Meerschaert and Xiao, 2005). We note here that $\dim_H \mathcal{W}_{\mathcal{D}}$ and thus, $\beta_{\mathcal{D}}$ determines the order of the generalization error. Using this simplification, we can easily compute $\beta_{\mathcal{D}}$, by first estimating each α_l by using the estimator proposed by Mohammadi et al. (2015), that can efficiently estimate α_l by using multiple iterations of the STDP learning process.

As explained by Simsekli et al. (2020a), due to the decomposability property of each dataset $\mathcal{D}$, the stochastic process for the synaptic weights given by $W^{(\mathcal{D})}(t)$ behaves like a Lévy motion around a local point w_0. Because of this locally regular behavior, the Hausdorff dimension can be bounded by the Blumenthal-Getoor (BG) index (Blumenthal and Getoor, 1960), which in turn depends on the tail behavior of the Lévy process. Thus, in summary, we can use the BG-index as a bound for the Hausdorff dimension of the trajectories from the STDP learning process. Now, as the Hausdorff dimension is a measure of the generalization error and is also controlled by the tail behavior of the process, heavier-tails imply less generalization error (Simsekli et al., 2020a,b).

To further demonstrate the heavy-tailed behavior of the Ornstein-Uhlenbeck process (a type of α−stable Lévy process L_t^α) that characterizes the STDP learning mechanism, we plot its trajectories and their corresponding pdf distributions. We plot these for varying values of the stability factor of the Lévy process L_t^α, α. We hence also plotted the probability density function of the Lévy processes to show the heavy-tailed nature of the Lévy trajectories as the tail index α decreases. **Figure 2** shows a continuous-time random walk performed using the O-U random process in 3D space. In the figure, X, Y, Z are random variables for the α-stable distributions generated using the O-U process. **Figure 3**, shows the corresponding probability density function

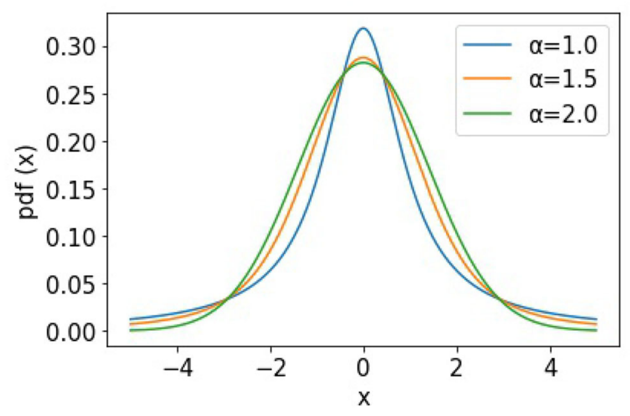


FIGURE 3 | Figure showing the probability density functions of the $\alpha-$stable Lévy process L_t^{α} for varying values of α.

of the O-U process for varying values of α corresponding to the different trajectories shown in **Figure 2**. From **Figures 2**, **3**, that as the O-U process becomes heavier-tailed (i.e., α decreases), and the Hausdorff dimension $\mathtt{dim}_H$ gets smaller.

2.3. Optimal Hyperparameter Selection

Using the Hausdorff Dimension as a metric for the generalizability of the learning process, we formulate an optimization process that selects the hyperparameters of the STDP process to improve the generalizability of the models. The Hausdorff dimension is a function of the hyperparameters of the STDP learning process. Thus, we formulate an optimization problem to select the optimal hyperparameters of the STDP using the Hausdorff dimension of the STDP learning process as the optimization function. Now, since the BG-index is the upper bound of the Hausdorff dimension, as discussed earlier, we in turn aim to optimize the BG-index of the STDP stochastic process. The optimization problem aims to get the optimal set of hyperparameters of the STDP process that can give a more generalizable model without looking at the test set data. Now, given an STDP process, we aim to tune its hyperparameters to search for a more generalizable model. Let us define $\boldsymbol{\lambda} := \{\lambda_1, \ldots, \lambda_N\}$ where $\boldsymbol{\lambda}$ is the set of N hyperparameters of the STDP process, $\lambda_1, \ldots, \lambda_N$. Let Λ_i denote the domain of the hyperparameter λ_i. The hyperparameter space of the algorithm is thus defined as $\boldsymbol{\Lambda} = \Lambda_1 \times \ldots \times \Lambda_N$. Now, we aim to design the optimization problem to minimize the Hausdorff Dimension of the learning trajectory for the STDP process. This is calculated over the last training iteration of the model, assuming that it reaches near the local minima. When trained with $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ on the training data $\mathcal{D}_{\textbf{train}}$, we denote the algorithm's Hausdorff dimension as $\dim_H G(\boldsymbol{\lambda}; \mathcal{D}_{\text{train}})$. Thus, using K-fold cross validation, the hyperparameter optimization problem for a given dataset $\mathcal{D}$ is to given as follows:

$$\boldsymbol{\lambda_s} = \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \frac{1}{K} \sum_{i=1}^{K} \dim_H G(\boldsymbol{\lambda}; \mathcal{D}_{\text{train}}) \tag{16}$$

We choose the Sequential Model-based Bayesian Optimization (SMBO) technique for this problem (Feurer et al., 2015). SMBO constructs a probabilistic model $\mathcal{M}$ of $f = \dim_H G$ based on point evaluations of f and any available prior information. It then uses that model to select subsequent configurations $\boldsymbol{\lambda}$ to evaluate. Given a set of hyperparameters $\boldsymbol{\lambda}$ for an STDP learning process G, we define the point functional evaluation as the calculation of the BG index of G with the hyperparameters $\boldsymbol{\lambda}$. The BG index gives an upper bound on the Hausdorff dimension of the learning process. In order to select its next hyperparameter configuration $\boldsymbol{\lambda}$ using model $\mathcal{M}$, SMBO uses an acquisition function $a_{\mathcal{M}} : \boldsymbol{\lambda} \rightarrow \mathbb{R}$, which uses the predictive distribution of model $\mathcal{M}$ at arbitrary hyperparameter configurations $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. This function is then maximized over $\boldsymbol{\Lambda}$ to select the most useful configuration $\boldsymbol{\lambda}$ to evaluate next. There exists a wide range of acquisition functions (Mockus et al., 1978), all of whom aim to trade-off between exploitation and exploration. The acquisition function tries to balance between locally optimizing hyperparameters in regions known to perform well and trying hyperparameters in a relatively unexplored region of the space.

In this paper, for the acquisition function, we use the expected improvement (Mockus et al., 1978) over the best previously-observed function value $f_{\min}$ attainable at a hyperparameter configuration $\boldsymbol{\lambda}$ where expectations are taken over predictions with the current model $\mathcal{M}$:

$$a(\boldsymbol{\lambda}, \mathcal{M}) = \int_{-\infty}^{f_{\min}} \max\left\{f_{\min} - f, 0\right\} \cdot p_{\mathcal{M}}(f \mid \boldsymbol{\lambda}) df \tag{17}$$

3. RESULTS

3.1. Experimental Setup

We empirically study the generalization properties of the STDP process by designing an SNN with 6,400 learning neurons for hand-written digit classification using the MNIST dataset. The MNIST dataset contains $60,000$ training examples and $10,000$ test examples of 28×28 pixel images of the digits $0 - 9$. It must be noted here that the images from the ten classes in the MNIST dataset are randomized so that there is a reinforcement of the features learned by the network.

3.1.1. Architecture

We use a two-layer SNN architecture as done by the implementation of Diehl and Cook (2015) as shown in **Figure 4**. The first layer is the input layer, containing 28×28 neurons with one neuron per image pixel. The second layer is the processing layer, with an equal number of excitatory and inhibitory neurons. The excitatory neurons of the second layer are connected in a one-to-one fashion to inhibitory neurons such that each spike in an excitatory neuron will trigger a spike in its corresponding inhibitory neuron. Again, each of the inhibitory neurons is connected to all excitatory ones, except for the one from which it receives a connection. This connectivity provides lateral inhibition and leads to competition among excitatory neurons. There is a balance between the ratio of inhibitory and excitatory synaptic conductance to ensure the correct strength of lateral inhibition. For a weak lateral inhibition, the conductance will not

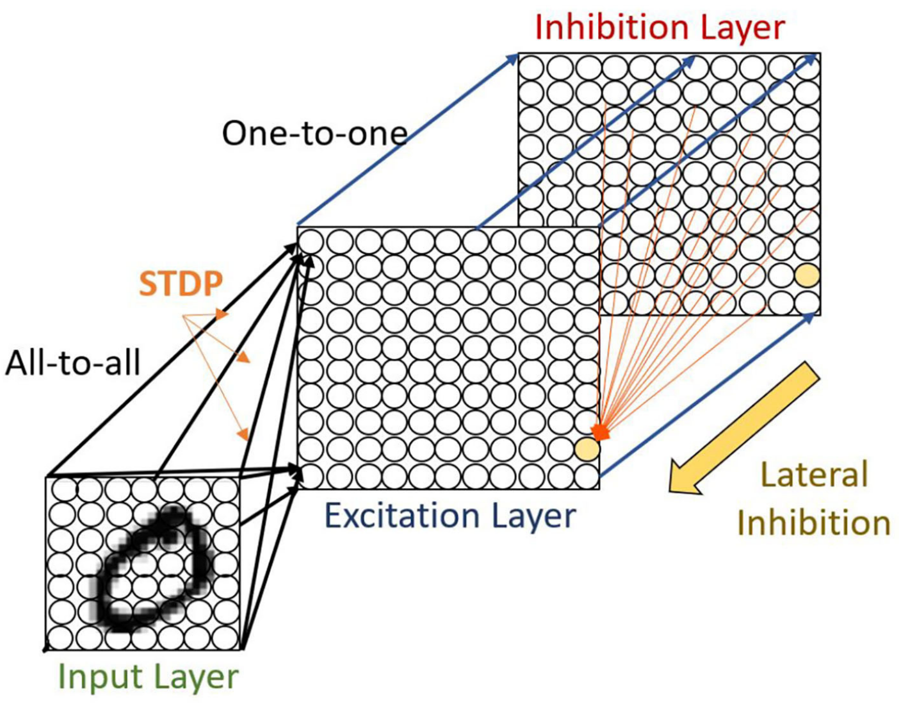


FIGURE 4 | The intensity values of the MNIST image are converted to Poisson-spike trains. The firing rates of the Poisson point process are proportional to the intensity of the corresponding pixel. These spike trains are fed as input in an all-to-all fashion to excitatory neurons. In the figure, the black shaded area from the input to the excitatory layer shows the input connections to one specific excitatory example neuron. The red shaded area denotes all connections from one inhibitory neuron to the excitatory neurons. While the excitatory neurons are connected to inhibitory neurons via one-to-one connection, each of the inhibitory neurons is connected to all excitatory ones, except for the one it receives a connection from.

have any influence while an extremely strong signal would ensue that one dominant neuron suppresses the other ones.

The task for the network is to learn a representation of the dataset on the synapses connecting the input layer neurons to the excitatory layer neurons. The excitatory-inhibitory connectivity pattern creates competition between the excitatory neurons. This allows individual neurons to learn unique filters where the single most spiked neuron in each iteration updates its synapse weights to match the current input digit using the chosen STDP rule. Increasing the number of neurons allows the network to memorize more examples from the training data and recognize similar patterns during the test phase.

3.1.2. Homeostasis

The inhomogeneity of the input leads to different firing rates of the excitatory neurons, and lateral inhibition further increases this difference. However, all neurons should have approximately equal firing rates to prevent single neurons from dominating the response pattern and to ensure that the receptive fields of the neurons differentiate. To achieve this, we employ an adaptive membrane threshold resembling intrinsic plasticity (Zhang and Linden, 2003). Specifically, each excitatory neuron's membrane threshold is not only determined by v_{thresh} but by the sum $v_{\text{thresh}} + \theta$, where θ is increased every time the neuron fires and is exponentially decaying (Querlioz et al., 2013). Therefore, the more a neuron fires, the higher will be its membrane threshold and in turn, the neuron requires more input to a spike soon. Using this mechanism, the firing rate of the neurons is limited because the conductance-based synapse model limits the maximum membrane potential to the excitatory reversal potential E_{exc}, i.e., once the neuron membrane threshold is close to E_{exc} (or higher) it will fire less often (or even stop firing completely) until θ decreases sufficiently.

3.1.3. Input Encoding

The input image is converted to a Poisson spike train with firing rates proportional to the intensity of the corresponding pixel. This spike train is then presented to the network in an all-to-all fashion for 350 ms as shown in **Figure 4**. The maximum pixel intensity of 255 is divided by 4, resulting in input firing rates between 0 and 63.75 Hz. Additionally, if the excitatory neurons in the second layer fire less than five spikes within 350 ms, the maximum input firing rate is increased by 32 Hz and the example is presented again for 350 ms. This process is repeated until at least five spikes have been fired during the entire time the particular example was presented.

3.1.4. Training and STDP Dynamics Analysis

To train the network, we present digits from the MNIST training set to the network. It is to be noted that, before presenting a new image, no input to any variable of any neuron is given

TABLE 1 | Table showing the set of hyperparameters for various STDP processes.

Hyperparameter	logSTDP	addSTDP	multSTDP
Synaptic Delay	0.75 ms	0.75 ms	0.75 ms
Synaptic epsp τ_A	1 ms	1 ms	1 ms
Synaptic epsp τ_B	5 ms	5 ms	5 ms
Number of correlated pools	4	4	4
Number of neurons per pool	50	50	50
Spiking rate of inputs	10 Hz	10 Hz	10 Hz
Learning rate (η)	0.0002	0.0002	0.0002
STDP Apre (LTP) time constant	17 ms	17 ms	17 ms
STDP Apre (LTD) time constant	34 ms	34 ms	34 ms
Increase in A_{pre} (LTP), on pre-spikes A_{pre0}	1.0	1.0	1.0
Increase in A_{post} (LTD), on post-spikes A_{post0}	0.5	0.55	100
LTD curvature factor t($\mathcal{S}$)	5	N/A	N/A
Exponential LTP decay factor t(γ)	50	N/A	N/A
Threshold fixed-point weight (W_0)	0.006	N/A	N/A

for a time interval of 150 ms. This is done to decay to their resting values. All the synaptic weights from input neurons to excitatory neurons are learned using the STDP learning process as described in section 2.1.2. To improve simulation speed, the weight dynamics are computed using synaptic traces such that every time a pre-synaptic spike x_{pre} arrives at the synapse, the trace is increased by 1 (Morrison et al., 2007). Otherwise, x_{pre} decays exponentially. When a post-synaptic spike arrives at the synapse the weight change Δw is calculated based on the pre-synaptic trace as described in section 2.1.2. To evaluate the model, we divide the training set into 100 divisions of 600 images each and check the model performance after each such batch is trained using the STDP learning model. In the remainder of the paper, we call this evaluation of the model after 600 images as one iteration.

3.1.5. Inference

After the training process is done, the learning rate is set to zero and each neuron's spiking threshold is fixed. A class is assigned to each neuron based on its highest response to the ten classes of digits over one presentation of the training set. This is the first time labels are used in the learning algorithm, which makes it an unsupervised learning method. The response of the class-assigned neurons is used to predict the digit. It is determined by taking the mean of each neuron response for every class and selecting the class with the maximum average firing rate. These predictions are then used to measure the classification accuracy of the network on the MNIST test set.

3.1.6. Computation of Generalization Error and Hausdorff Dimension

We empirically study the generalization behavior of STDP trained SNNs. We vary the hyperparameters of the STDP learning process which controls the LTP/LTD dynamics of the STDP learning algorithm. **Table 1** shows the hyperparameters for various STDP processes. We trained all the models for 100 training iterations. In this paper, we consider the synaptic weight update to follow a Lévy process, i.e., continuous with discrete jumps similar to the formulation of Stein (1965) and Richardson and Swarbrick (2010). As discussed in section 2.2, the generalizability can be measured using the Hausdorff dimension which is bounded by BG-index.

Therefore, the BG-index is computed in the last iteration when all the neurons have learned the input representations. We also compute the generalization error as the difference between the training and test accuracy. we study the relations between BG-index, generalization error, testing accuracy, and convergence behavior of the networks.

3.2. Analysis of Generalizability of STDP Processes

3.2.1. Impact of Scaling Functions

Kubota et al. showed that the scaling functions play a vital role in controlling the LTP/LTD dynamic of the STDP learning method (Kubota et al., 2009). In this subsection, we evaluate the impact of scaling functions (i.e., $a_\pm$ in the Equation 3) on the generalizability properties of the STDP methods. We define the ratio of the post-synaptic scaling function to the pre-synaptic one (i.e., c_+/c_- in add-, mult-, and log- STDP equations), hereafter referred to as the scaling function ratio (SFR), as our variable. Kubota et al. has shown that the learning behavior is best when this SFR lies between the range of 0.9 to 1.5. Hence, we also modulate the SFR within this set interval. **Table 2** shows the impact of scaling function on Hausdorff dimension (measured using BG-index), generalization error, and testing accuracy. We observe that a smaller SFR leads to a lower Hausdorff dimension and a lower generalization error, while a higher ratio infers a less generalizable model. However, a higher SFR marginally increases the testing accuracy. The analysis shows confirms that a higher Hausdorff dimension (i.e., a higher BG-index) corresponds to a higher generalization error, as discussed in section 2.2, justifying the use of BG-index as a measure of the generalization error. We also plot the learned digit representations for different SFRs for better visualization of the distinction in generalization behavior. The plots are shown in **Figure 5**.

3.2.2. Impact of the Learning Rate

One of the major parameters that control the weight dynamics of the STDP processes is the learning rate i.e., the variable η in Equation (2). In this subsection, we evaluate the effect of the learning rate on the generalizability of the STDP process. We have summarized the results in **Table 3**. We observe that a larger learning rate converges to a lesser generalizable solution. This can be attributed to the fact that a higher learning rate inhibits convergence to sharper minimas; rather facilitates convergence to a flatter one resulting in a more generalizable solution. We also observe the monotonic relation between the BG-index and the generalization error.

3.2.3. Impact of STDP models on Generalizability

In this section, we compare the three different STDP models, namely, add-STDP, mult-STDP, and log-STDP to its generalization abilities with changing SFR (scaling function ratio) and learning rate. The results are summarized in **Tables 2**, **3**. In all the above cases we see that the log-STDP process has a

TABLE 2 | Impact of the post-synaptic to pre-synaptic scaling functions ratio on generalization.

$\frac{c_+}{c_-}$	log-STDP			add-STDP			mult-STDP		
	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy
2.1	1.352	6.8	89.92	1.969	9.7	88.17	1.824	8.1	89.26
1.7	1.294	6.2	89.98	1.911	9.3	88.12	1.797	7.6	89.15
1.2	1.209	5.9	89.79	1.875	8.9	88.09	1.702	7.0	88.99
0.9	1.174	5.7	89.26	1.799	8.6	88.10	1.633	6.5	88.87

The values noted as generalization error in the table is computed as: (|Training Accuracy–Test Accuracy|).

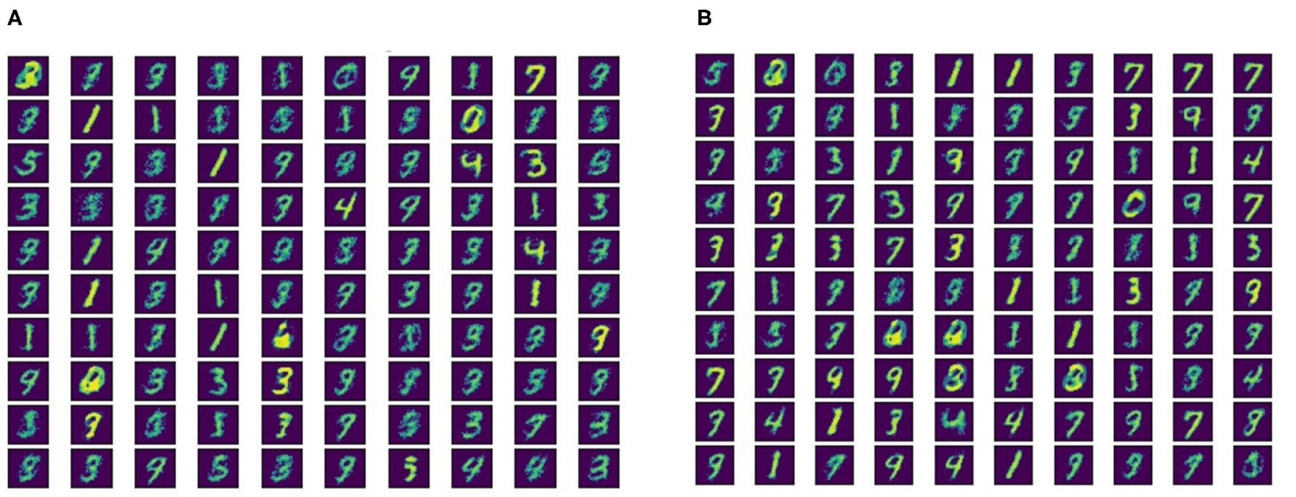


FIGURE 5 | Neuron connection weights for learned digit representations for **(A)** SFR = 0.9 and **(B)** SFR = 2.1.

TABLE 3 | The impact of learning rate on the generalization error.

η	log-STDP			add-STDP			mult-STDP		
	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy
0.2	1.312	6.6	89.12	1.844	9.1	87.69	1.769	7.9	88.38
0.15	1.255	6.1	89.03	1.783	8.9	87.53	1.648	7.4	88.19
0.1	1.112	5.3	88.35	1.698	8.5	87.11	1.596	6.8	87.95
0.05	1.068	5.0	88.01	1.632	8.2	87.02	1.512	6.3	87.82

The values noted as Generalization Error in the table is computed as: (|Training Accuracy–Test Accuracy|).

TABLE 4 | Table showing comparison of the STDP models on the fashion-MNIST dataset.

	log-STDP			add-STDP			mult-STDP		
	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy	BG index	Generalization error (| Train Acc. - Test Acc.|)	Testing accuracy
$c_+/c_- = 0.9$	1.322	10.02	82.43	1.788	13.87	79.16	1.573	11.38	81.92
$\eta = 0.05$	1.204	9.93	81.75	1.692	11.73	79.76	1.489	10.11	80.35

The values noted as generalization error in the table is computed as: (|Training Accuracy–Test Accuracy|).

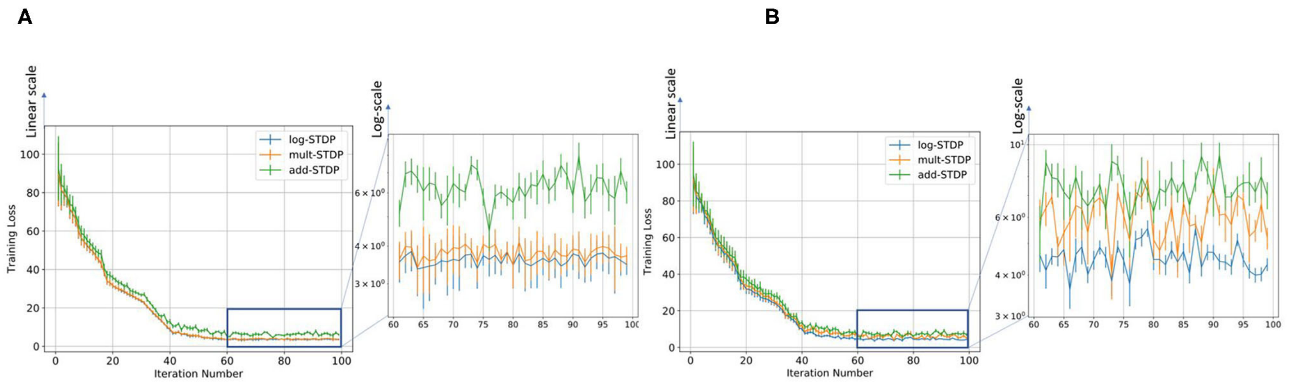


FIGURE 6 | Figure showing the variation in the training loss with increasing iterations for different types of STDP models keeping **(A)** SFR= 0.9 and **(B)** $\eta = 0.05$.

significantly lower generalization error compared to the other two STDP methods. The difference between the generalizability of various STDP models comes from the nature of the stochastic distribution of weights generated by different models.

Gilson and Fukai (2011) has discussed that add-STDP (Gütig et al., 2003) can rapidly and efficiently select synaptic pathways by splitting synaptic weights into a bimodal distribution of weak and strong synapses. However, the stability of the weight distribution requires hard bounds due to the resulting unstable weight dynamics. In contrast in mult-STDP (Rubin et al., 2001), weight-dependent update rules can generate stable unimodal distributions. However, mult-STDP weakens the competition among synapses leading to only weakly skewed weight distributions. The probability distributions of the three different STDP models are shown in **Figure 1**. On the other hand, log-STDP proposed by Gilson and Fukai (2011) bypass these problems by using a weight-dependent update rule while does not make the other synapses weak as in mult-STDP. The log-STDP results in a log-normal solution of the synaptic weight distribution as discussed by Gilson and Fukai (2011). A log-normal solution has a heavier tail and thus a smaller Hausdorff dimension leading to a lower generalization error. A detailed comparison of the weight distributions of the three types of STDP processes can be found in the paper by Gilson and Fukai (2011). We further evaluated the training loss for iterations for the different STDP models. The results are plotted in **Figure 6**. From the figures, we see that the log-STDP outperforms the add-STDP and the mult-STDP in terms of training loss for either case.

3.2.4. Impact on Different Datasets

To demonstrate the generalizability of the STDP models, we also tested its performance on Fashion-MNIST (Xiao et al., 2017) which is an MNIST-like fashion product dataset with 10 classes. Fashion-MNIST shares the same image size and structure of the training and testing splits as MNIST but is considered more realistic as its images are generated from front look thumbnail images of fashion products on Zalando's website via a series of conversions. Therefore, Fashion-MNIST poses a more challenging classification task than MNIST. We preprocessed the data by normalizing the sum of a single sample gray value because of the high variance among examples. The results for SFR $c_+/c_- = 0.9$ and learning rate $\eta = 0.05$ is shown in **Table 4**. We see that as seen in MNIST datasets, for the Fashion-MNIST also, the log-STDP method has a lower generalization error corresponding to a lower BG Index.

3.3. Generalizability vs. Trainability Tradeoff

In this section, we study the relations between the generalizability and trainability of a learning model. For the sake of brevity, we only focus on the log-STDP process as it has shown better generalizability compared to add-STDP and mult-STDP.

We plot the training loss as a function of the time evolution of the synaptic weights trained with the STDP learning method. Since STDP is an online unsupervised learning algorithm, there is no formal concept of training loss. So, to evaluate the performance of the model, we define the training loss as follows: We divide the MNIST dataset into 100 divisions, with each division consisting of 600 images. We evaluate the model after training the model on each subset of the full training dataset and this is considered as one training iteration. We train the SNN model using STDP with this limited training dataset. After the training is done, we set the learning rate to zero and fix each neuron's spiking threshold. Then, the image of each class is given as input to the network, and the total spike count of all the neurons that have learned that class is calculated. Hence, the spike counts are normalized by dividing the number of spikes by the maximum number of spike counts and the cross-entropy loss was calculated across all the classes. This is defined as the training loss. To show the confidence interval of each training iteration, we evaluated each of the partially trained models on the test dataset 5 times, randomizing the order of images for each of the test runs. We see from **Figures 7**, **8**, that initially, as the model was trained with fewer images, the variance in training loss was high demonstrating low confidence. However, as the model is trained on a larger training set, the variance decreases as expected.

Table 2 and **Figure 4** show the training loss vs. the number of iterations for the log-STDP process for various SFR.

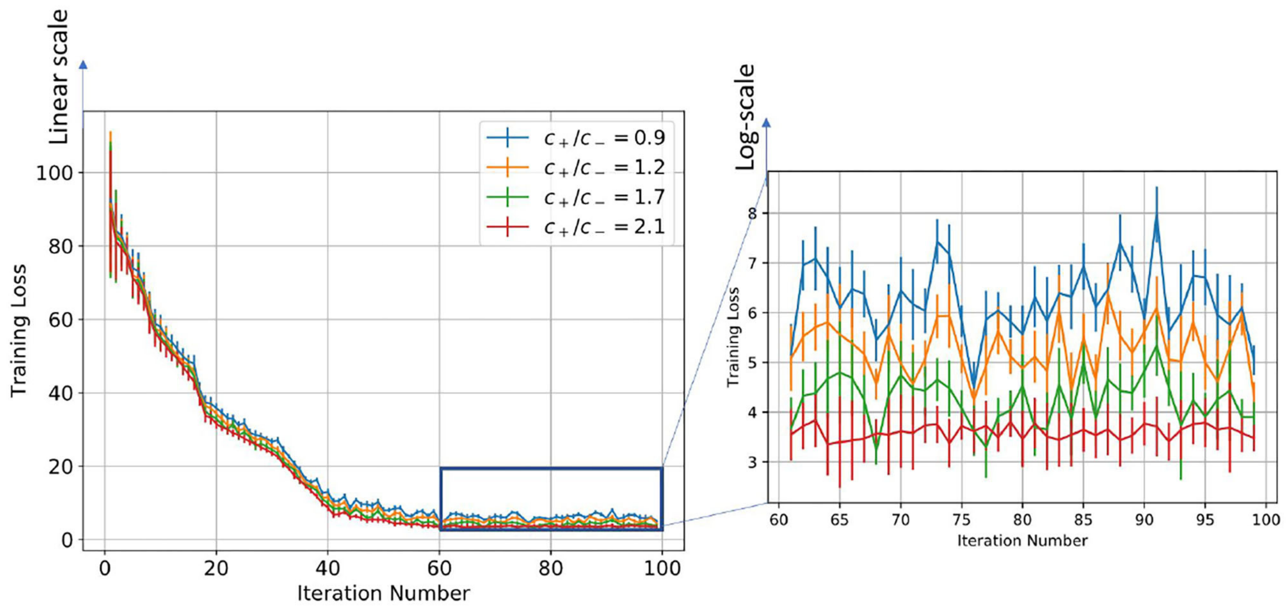


FIGURE 7 | Figure showing the change in training loss with iterations for varying scaling function ratios for the log-STDP learning process.

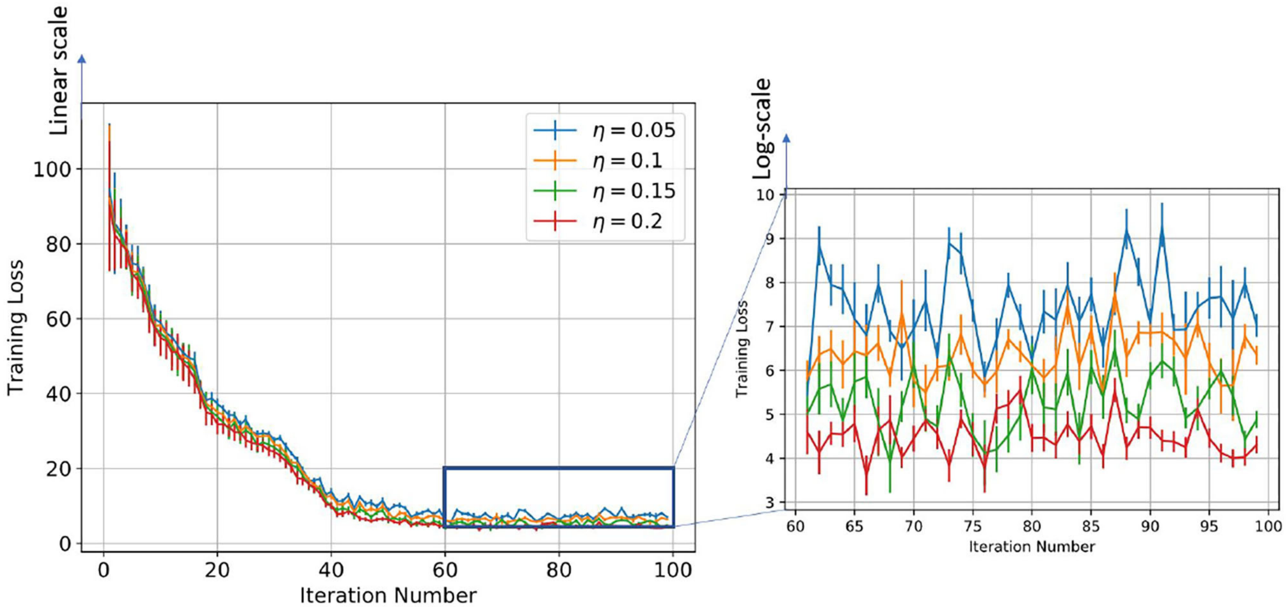


FIGURE 8 | Figure showing the change in training loss with iterations for varying learning rates for the log-STDP learning process.

We see that the $SFR = c_+/c_- = 0.9$ shows a lower generalization error and almost similar testing accuracy, compared to the other SFRs. The results show that increasing the SFR increases the generalization error. If the pre-synaptic scaling function is stronger than the post-synaptic scaling function (i.e., c_+/c_- is lower), it implies that the synaptic weights of the neurons gradually decay. Since we have the images in the MNIST dataset randomized over the ten classes, the more important features which help in the generalization ability of the network over unknown data are reinforced so that the network does not forget these filters as shown by Panda et al. (2017). Thus, the network only forgets the less important features and preserves the more important ones, hence making it more generalizable. Since the neuron forgets some features which would help to fit better into the current dataset, it affects its training/testing

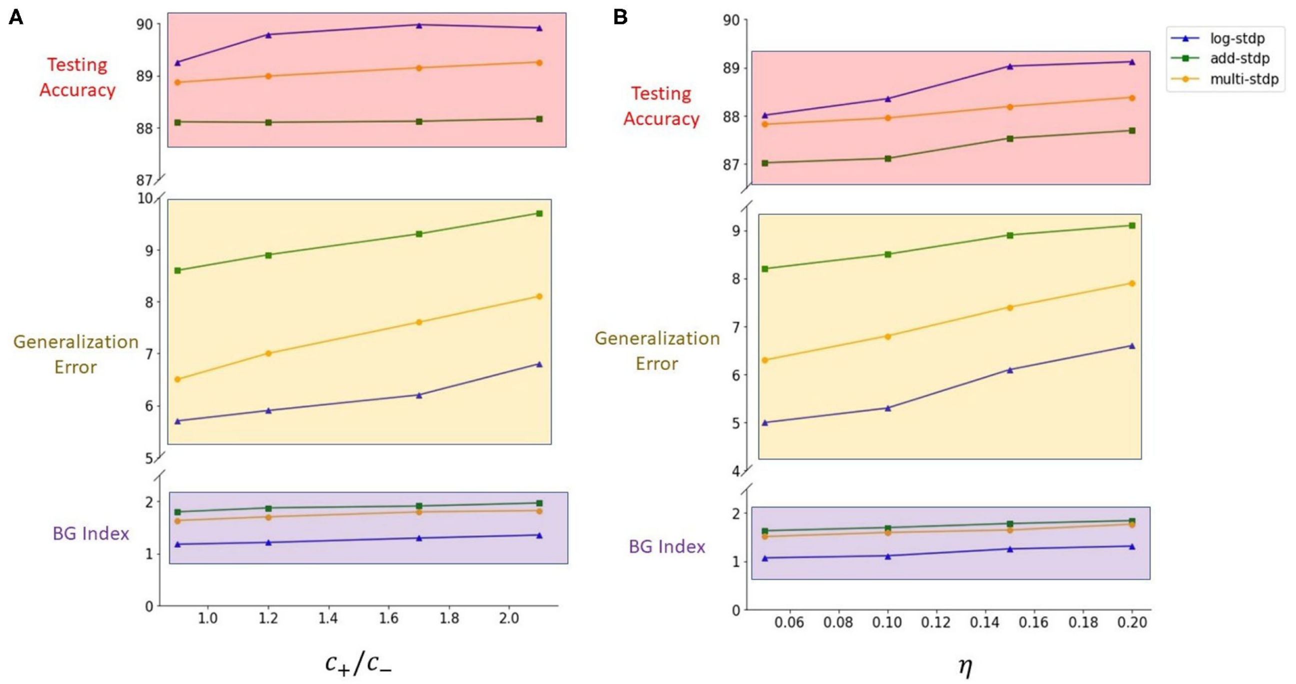


FIGURE 9 | (A) Plots for the impact of the scaling function ratios on generalization (results shown in **Table 2**). **(B)** Plots for the impact of the learning rates on generalization (results shown in **Table 3**).

TABLE 5 | Table showing the set of hyperparameters used for the Bayesian optimization problem for the log-STDP process.

Hyperparameter	**Domain**	**logSTDP**		**add-STDP**	
		Before BO	**After BO**	**Before BO**	**After BO**
Learning rate (η)	[0.05, 0.2]	0.1	0.063	0.1	0.017
Variance of Noise ζ (σ)	[0.1, 1]	0.5	0.581	0.5	0.632
Degree of log-like saturation ($\mathcal{S}$)	$\mathbb{Z} \in [1, 10]$	3	5	N/A	N/A
Exponential Decay factor (γ)	$\mathbb{Z} \in [10, 100]$	45	57	N/A	N/A
Threshold Fixed-point weight (W_0)	[0, 1]	0.5	0.244	N/A	N/A
Scaling functions (c_+, c_-)	(0,1]×(0,1]	0.5, 0.45 ($\frac{c_+}{c_-} = 1.11$)	0.752, 0.788 ($\frac{c_+}{c_-} = 0.954$)	0.5, 0.45 ($\frac{c_+}{c_-} = 1.11$)	0.894, 0.652 ($\frac{c_+}{c_-} = 1.371$)
Time Constants (ms) (τ_+, τ_-)	[10,20]×[20, 40]	15,30	17, 36	15,30	18,22
Testing accuracy		91.41	90.65	88.68	89.77
Generalization error		5.79	3.17	8.29	4.61

accuracy as can be seen in **Tables 2**, **3**. Thus, the model learns the more important features and is essentially more generalizable. The results for testing accuracy, generalization error and BG index for varying SFR and learning rates are shown in **Figure 9**.

Note here that the training loss for the STDP processes all reach their convergence around iteration 60 - i.e., images after that added little information for the training of the model. The models here are not optimized and hence optimizing the hyperparameters can also help in reducing the number of images required for extracting enough information from the training dataset. Thus, if SFR is too high, training gets messed up since a neuron starts spiking for multiple patterns, in which case there is no learning. As the SFR value increases from 1, the SNN tends

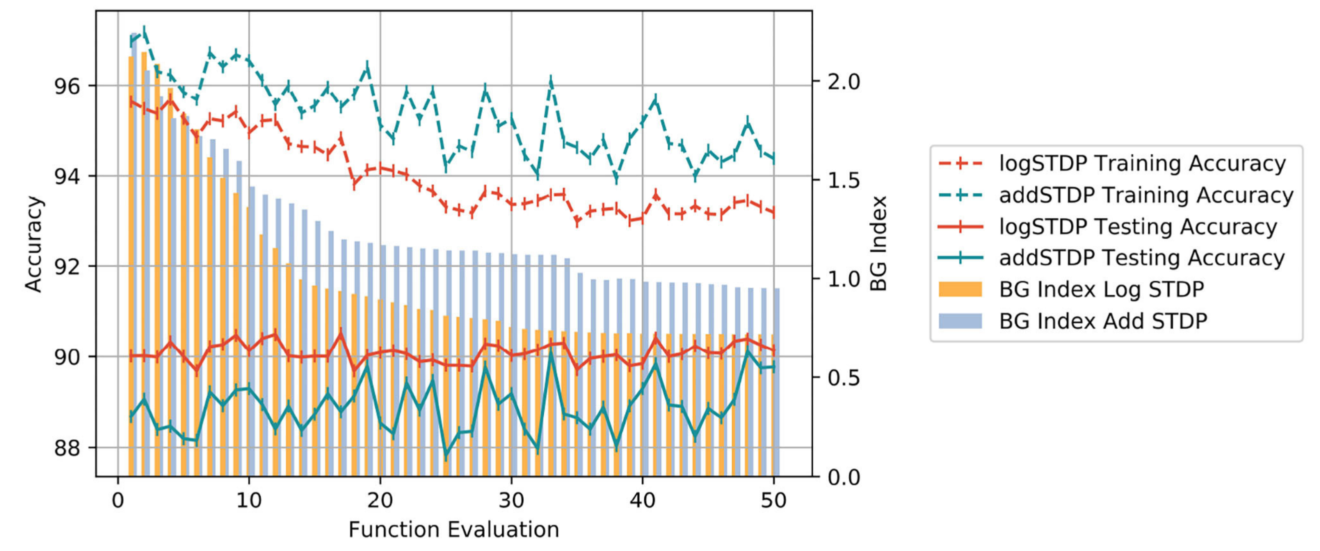


FIGURE 10 | Plot showing the change of BG Index, Training and Testing Accuracy between the add-STDP and log-STDP over functional evaluations during Bayesian Optimization.

to memorize the training pattern and hence the generalization performance is poor. On the other hand, if when SFR is less than 1 but is close to 1, it is hard to memorize the training patterns as the STDP process tends to forget the patterns which are non-repeating, leading to better generalization.

On the other hand, if the post-synaptic scaling function is stronger than the pre-synaptic one (i.e., c_+/c_- is higher), then the neurons tend to learn more than one pattern and respond to all of them. Similar results can be verified from **Figure 7** where the learning rate was varied instead of the SFR. In this study as well we observe that a higher learning rate, although leads to faster convergence and lower training loss, leads to a less generalizable model. Hence, we empirically observe that hyperparameters of STDP models that lead to better generalizability can also make an SNN harder to train.

3.4. Results of Hyperparameter Optimization

In section 3.2, we empirically showed that the Hausdorff dimension is a good measure of the generalizability of the model and it can be very efficiently controlled using the hyperparameters of the STDP learning process. In this section, we show the application of our Bayesian optimization strategy to search for the optimal hyperparameters to increase the generalizability of an STDP-trained SNN model. For the sake of brevity, we demonstrate the application of Bayesian optimization on the log-STDP process. **Table 5** shows the set of hyperparameters that are optimized and their optimal values obtained by our approach. It should be clarified here that the hyper-parameters are not necessarily the absolute global optimum but a likely local optimum found in the optimization algorithm. The optimized log-STDP model results in a training accuracy of 93.75%, testing accuracy of 90.49%, and a BG Index of 0.718 for the MNIST dataset.

We study the behavior of Bayesian optimization. Each iteration in the Bayesian optimization process corresponds to a different set of hyperparameters for the log-STDP model. Each such iteration is called a functional evaluation. For each functional evaluation, the Bayesian Optimization trains the SNN with the corresponding set of hyperparameters of the log-STDP model and measures the BG-index of the weight dynamics of the trained-SNN. **Figure 10** shows the change in the BG-Index as a function of a number of the function evaluations of the search process. It is to be noted here that at each functional evaluation, we train the network with the STDP learning rule with the chosen hyperparameters and estimate the Hausdorff dimension from the trained network. We see that for the optimal set of hyperparameters, the BG Index converges to 0.7. **Figure 10** also shows the corresponding training accuracy of the model with the change in iteration number. We see that the log-STDP configurations during Bayesian optimization that have a higher BG Index (i.e., a higher generalization error) also have has a higher training accuracy. These results further validate our observations on the generalizability vs. trainability tradeoff for a log-STDP trained SNN.

3.4.1. Comparison With Add-STDP

In order to compare the performance of the log-STDP, we performed a similar analysis using the add-STDP model. The results of the Bayesian Optimization for the add-STDP and the log-STDP are plotted in **Figure 10**. We see that the log-STDP process outperforms the add-STDP model in terms of both training/testing accuracy and the generalization error thus showing the robustness of the log-STDP process. We see that though the add-STDP has a higher training accuracy, and a comparable test accuracy, its generalization error is higher compared to the log-STDP method.

4. DISCUSSION

In this paper, we presented the generalization properties of the spike timing-dependent plasticity (STDP) models. A learning process is said to be more generalizable if it can extract features that can be transferred easily to unknown testing sets thus decreasing the performance gap between the training and testing sets. We provide a theoretical background for the motivation of the work treating the STDP learning process as a stochastic process (an Ornstein-Uhlenbeck process) and modeling it using a stochastic differential equation. We control the hyperparameters of the learning method and empirically study their generalizability properties using the Hausdorff dimension as a measure. From **Tables 2**, **3** and corresponding **Figure 9**, we observed that the Hausdorff Dimension is a good measure for the estimation of the generalization error of an STDP-trained SNN. We compared the generalization error and testing error for the log-STDP, add-STDP, and mult-STDP models, as shown in **Tables 2**, **3**. We observed that the lognormal weight distribution obtained from the log-STDP learning process leads to a more generalizable STDP-trained SNN with a minimal decrease in testing accuracy. In this paper, when we refer to a model as more generalizable, we mean there is a smaller difference between the training and testing performance, i.e., the generalization error. The objective of the paper was to get a model which is more generalizable in the sense that the performance of the network on unknown datasets should not differ much from its performance in the training dataset. It is to be noted that in this paper we are using the generalization error as the metric of generalizability of the network. Generalization error is not a measure of absolute accuracy, but rather the difference between training and testing datasets. As such, we see that models which have lower generalization error extract lesser and more important features compared to less generalizable models. However, we see that with this reduced set of features, the model has almost no drop in the testing accuracy, showing the generalizability of the model at comparable accuracy. Thus, we get a model which is more generalizable in the sense that the performance of the network on unknown datasets does not differ much from its performance in the training dataset. As such, these "more generalizable" models, extract lesser and more important features compared to less generalizable models. However, we see that with this reduced set of features, the model has almost no drop in the testing accuracy, showing the generalizability of the model as we can see from **Tables 2**, **3**. This phenomenon can be explained using the observations of Panda et al. (2017) on how the ability to forgets boosts the performance of spiking neuron models. The authors showed that the final weight value toward the end of the recovery phase is greater for the frequent input. The prominent weights will essentially encode the features that are common across different classes of old and new inputs as the pre-neurons across those common feature regions in the input image will have frequent firing activity. This eventually helps the network to learn features that are more common with generic representations across different input patterns. This extraction of more generalizable features can be interpreted as a sort of regularization wherein the network tries to generalize over the input rather than overfitting such that the overall accuracy of the network improves. However, due to this regularization, we see that the training performance of the network decreases. However, since the model is more generalizable, the testing performance remains almost constant as seen in **Figure 10**. We further observe that the log-STDP models which have a lower Hausdorff dimension and hence have lower generalization error, have a worse trainability i.e., takes a long time to converge during training and also converges to a higher training loss. The observations show that an STDP model can have a trade-off between generalizability and trainability. Finally, we present a Bayesian optimization problem that minimizes the Hausdorff dimension by controlling the hyperparameter of a log-STDP model leading to a more generalizable STDP-trained SNN.

Future work on this topic will consider other models of STDP. In particular, the stochastic STDP rule where the probability of synaptic weight update is proportional to the time difference of the arrival of the pre and post-synaptic spikes has shown improved accuracy over deterministic STDP studied in this paper. The trajectories of such a stochastic STDP model will lead to a Feller process as shown by Kuhn (Helson, 2017). Hence, in the future, we will perform a similar Hausdorff dimension-based analysis for generalization for the stochastic STDP model. Moreover, in this work, we have only considered the hyperparameters of the STDP model to improve the generalizability of the SNN. An important extension is to consider the properties of the neuron dynamics, which also controls the generation of the spikes and hence, weight distribution. The choice of the network architecture will also play an important role in the weight distribution of the SNN. Therefore, a more comprehensive optimization process that couples hyperparameters of the STDP dynamics, neuron dynamics, and network architecture like convolutional SNN (Kheradpisheh et al., 2018) and heterogeneous SNN (She et al., 2021) will be interesting future work.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: https://deepai.org/dataset/mnist.

AUTHOR CONTRIBUTIONS

BC developed the main concepts, performed simulation, and wrote the paper under the guidance of SM. Both authors assisted in developing the concept and writing the paper.

FUNDING

This material was based on work sponsored by the Army Research Office and was accomplished under Grant Number W911NF-19-1-0447. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government.

REFERENCES

Aceituno, P. V., Ehsani, M., and Jost, J. (2020). Spiking time-dependent plasticity leads to efficient coding of predictions. *Biol. Cybernet*. 114, 43–61. doi: 10.1007/s00422-019-00813-w

Allen-Zhu, Z., and Li, Y. (2019). Can SGD learn recurrent neural networks with provable generalization? *arXiv preprint arXiv:1902.01028*.

Allen-Zhu, Z., Li, Y., and Liang, Y. (2018). Learning and generalization in overparameterized neural networks, going beyond two layers. *arXiv preprint arXiv:1811.04918*.

Baity-Jesi, M., Sagun, L., Geiger, M., Spigler, S., Arous, G. B., Cammarota, C., et al. (2018). "Comparing dynamics: deep neural networks versus glassy systems," in *International Conference on Machine Learning* (PMLR) (Stockholm), 314–323. doi: 10.1088/1742-5468/ab3281

Bell, C. C., Han, V. Z., Sugawara, Y., and Grant, K. (1997). Synaptic plasticity in a cerebellum-like structure depends on temporal order. *Nature* 387, 278–281. doi: 10.1038/387278a0

Bi, G.-Q., and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci*. 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Bishop, C. J., and Peres, Y. (2017). *Fractals in Probability and Analysis*, Vol. 162. Cambridge University Press. doi: 10.1017/9781316460238

Blumenthal, R. M., and Getoor, R. K. (1960). Some theorems on stable processes. *Trans. Am. Math. Soc*. 95, 263–273. doi: 10.1090/S0002-9947-1960-0119247-6

Burkitt, A. N., Meffin, H., and Grayden, D. B. (2004). Spike-timing-dependent plasticity: the relationship to rate-based learning for models with weight dynamics determined by a stable fixed point. *Neural Comput*. 16, 885–940. doi: 10.1162/089976604773135041

Camuto, A., Deligiannidis, G., Erdogdu, M. A., Gürbüzbalaban, M., Şimşekli, U., and Zhu, L. (2021). Fractal structure and generalization properties of stochastic optimization algorithms. *arXiv preprint arXiv:2106.04881*.

Capocelli, R., and Ricciardi, L. (1976). On the transformation of diffusion processes into the feller process. *Math. Biosci*. 29, 219–234. doi: 10.1016/0025-5564(76)90104-8

Câteau, H., and Fukai, T. (2003). A stochastic method to predict the consequence of arbitrary forms of spike-timing-dependent plasticity. *Neural Comput*. 15, 597–620. doi: 10.1162/089976603321192095

Chen, G., Qu, C. K., and Gong, P. (2020). Anomalous diffusion dynamics of learning in deep neural networks. *arXiv preprint arXiv:2009.10588*.

Chichilnisky, E. (2001). A simple white noise analysis of neuronal light responses. *Netw. Comput. Neural Syst*. 12, 199–213. doi: 10.1080/713663221

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci*. 9:99. doi: 10.3389/fncom.2015.00099

Feldman, D. E. (2012). The spike-timing dependence of plasticity. *Neuron* 75, 556–571. doi: 10.1016/j.neuron.2012.08.001

Feurer, M., Springenberg, J., and Hutter, F. (2015). "Initializing bayesian hyperparameter optimization via meta-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence* (Austin, TX).

Gerstner, W., and Kistler, W. M. (2002a). Mathematical formulations of hebbian learning. *Biol. Cybernet*. 87, 404–415. doi: 10.1007/s00422-002-0353-y

Gerstner, W., and Kistler, W. M. (2002b). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press. doi: 10.1017/CBO9780511815706

Gilson, M., and Fukai, T. (2011). Stability versus neuronal specialization for STDP: long-tail weight distributions solve the dilemma. *PLoS ONE* 6:e25339. doi: 10.1371/journal.pone.0025339

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gurbuzbalaban, M., Simsekli, U., and Zhu, L. (2020). The heavy-tail phenomenon in SGD. *arXiv preprint arXiv:2006.04740*.

Gütig, R., Aharonov, R., Rotter, S., and Sompolinsky, H. (2003). Learning input correlations through nonlinear temporally asymmetric Hebbian plasticity. *J. Neurosci*. 23, 3697–3714. doi: 10.1523/JNEUROSCI.23-09-03697.2003

Han, V. Z., Grant, K., and Bell, C. C. (2000). Reversible associative depression and nonassociative potentiation at a parallel fiber synapse. *Neuron* 27, 611–622. doi: 10.1016/S0896-6273(00)00070-2

Helson, P. (2017). A new stochastic stdp rule in a neural network model. *arXiv preprint arXiv:1706.00364*.

Hodgkinson, L., and Mahoney, M. (2021). "Multiplicative noise and heavy tails in stochastic optimization," in *International Conference on Machine Learning* (PMLR), 4262–4274.

Jones, M. C., Marron, J. S., and Sheather, S. J. (1992). *Progress in Data-Based Bandwidth Selection for Kernel Density Estimation*. Technical report. North Carolina State University. Dept. of Statistics, Raleigh, NC, United States.

Kawaguchi, K., Kaelbling, L. P., and Bengio, Y. (2017). Generalization in deep learning. *arXiv preprint arXiv:1710.05468*.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw*. 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Khoshnevisan, D. (2009). "From fractals and probability to Lévy processes and stochastic PDES," in *Fractal Geometry and Stochastics IV*, eds C. Bandt, M. Zähle, and P. Mörters (Basel: Springer), 111–141. doi: 10.1007/978-3-0346-0030-9_4

Khoshnevisan, D., and Xiao, Y. (2017). "On the macroscopic fractal geometry of some random sets," in *Stochastic Analysis and Related Topics,* eds F. Baudoin and J. Peterson (Cham: Springer), 179–206. doi: 10.1007/978-3-319-59671-6_9

Kubota, S., Rubin, J., and Kitajima, T. (2009). Modulation of LTP/LTD balance in STDP by an activity-dependent feedback mechanism. *Neural Netw*. 22, 527–535. doi: 10.1016/j.neunet.2009.06.012

Lőrinczi, J., and Yang, X. (2019). Multifractal properties of sample paths of ground state-transformed jump processes. *Chaos Solitons Fractals* 120, 83–94. doi: 10.1016/j.chaos.2019.01.008

Le Guével, R. (2019). The hausdorff dimension of the range of the Lévy multistable processes. *J. Theoret. Probabil*. 32, 765–780. doi: 10.1007/s10959-018-0847-8

Lee, C., Srinivasan, G., Panda, P., and Roy, K. (2018). Deep spiking convolutional neural network trained with unsupervised spike-timing-dependent plasticity. *IEEE Trans. Cogn. Dev. Syst*. 11, 384–394. doi: 10.1109/TCDS.2018.2833071

Legenstein, R., Pecevski, D., and Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Comput. Biol*. 4:e1000180. doi: 10.1371/journal.pcbi.1000180

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw*. 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Magee, J. C., and Johnston, D. (1997). A synaptically controlled, associative signal for Hebbian plasticity in hippocampal neurons. *Science* 275, 209–213. doi: 10.1126/science.275.5297.209

Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APS and EPSPS. *Science* 275, 213–215. doi: 10.1126/science.275.5297.213

Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2009). Competitive STDP-based spike pattern learning. *Neural Comput*. 21, 1259–1276. doi: 10.1162/neco.2008.06-08-804

Meerschaert, M. M., and Xiao, Y. (2005). Dimension results for sample paths of operator stable Lévy processes. *Stochast. Process. Appl*. 115, 55–75. doi: 10.1016/j.spa.2004.08.004

Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of bayesian methods for seeking the extremum. *Towards global optimization* 2:2.

Mohammadi, M., Mohammadpour, A., and Ogata, H. (2015). On estimating the tail index and the spectral measure of multivariate α-stable distributions. *Metrika* 78, 549–561. doi: 10.1007/s00184-014-0515-7

Morrison, A., Aertsen, A., and Diesmann, M. (2007). Spike-timing-dependent plasticity in balanced random networks. *Neural Comput*. 19, 1437–1467. doi: 10.1162/neco.2007.19.6.1437

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., and Masquelier, T. (2019). Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern Recogn*. 94, 87–95. doi: 10.1016/j.patcog.2019.05.015

Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. (2017). Exploring generalization in deep learning. *arXiv preprint arXiv:1706.08947*.

Panda, P., Allred, J. M., Ramanathan, S., and Roy, K. (2017). ASP: learning to forget with adaptive synaptic plasticity in spiking neural networks. *IEEE J. Emerg. Select. Top. Circ. Syst*. 8, 51–64. doi: 10.1109/JETCAS.2017.2769684

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci*. 12:774. doi: 10.3389/fnins.2018.00774

Poggio, T., Banburski, A., and Liao, Q. (2019). Theoretical issues in deep networks: approximation, optimization and generalization. *arXiv preprint arXiv:1908.09375*.

Querlioz, D., Bichler, O., Dollfus, P., and Gamrat, C. (2013). Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Trans. Nanotechnol*. 12, 288–295. doi: 10.1109/TNANO.2013.2250995

Richardson, M. J., and Swarbrick, R. (2010). Firing-rate response of a neuron receiving excitatory and inhibitory synaptic shot noise. *Phys. Rev. Lett*. 105:178102. doi: 10.1103/PhysRevLett.105.178102

Robert, P., and Vignoud, G. (2020). Stochastic models of neural synaptic plasticity. *arXiv preprint arXiv:2010.08195*.

Roberts, P. D., and Bell, C. C. (2000). Computational consequences of temporally asymmetric learning rules: II. Sensory image cancellation. *J. Comput. Neurosci*. 9, 67–83. doi: 10.1023/A:1008938428112

Rubin, J., Lee, D. D., and Sompolinsky, H. (2001). Equilibrium properties of temporally asymmetric Hebbian plasticity. *Phys. Rev. Lett*. 86:364. doi: 10.1103/PhysRevLett.86.364

She, X., Dash, S., Kim, D., and Mukhopadhyay, S. (2021). A heterogeneous spiking neural network for unsupervised learning of spatiotemporal patterns. *Front. Neurosci*. 14:1406. doi: 10.3389/fnins.2020.615756

Sheather, S. J., and Jones, M. C. (1991). A reliable data-based bandwidth selection method for kernel density estimation. *J. R. Stat. Soc. Ser. B* 53, 683–690. doi: 10.1111/j.2517-6161.1991.tb01857.x

Simsekli, U., Sagun, L., and Gurbuzbalaban, M. (2019). A tail-index analysis of stochastic gradient noise in deep neural networks. *arXiv preprint arXiv:1901.06053*.

Simsekli, U., Sener, O., Deligiannidis, G., and Erdogdu, M. A. (2020a). "Hausdorff dimension, heavy tails, and generalization in neural networks," in *Advances in Neural Information Processing Systems 33*.

Simsekli, U., Zhu, L., Teh, Y. W., and Gurbuzbalaban, M. (2020b). "Fractional underdamped langevin dynamics: retargeting SGD with momentum under heavy-tailed gradient noise," in *International Conference on Machine Learning* (PMLR) (Vienna), 8970–8980.

Sinz, F. H., Pitkow, X., Reimer, J., Bethge, M., and Tolias, A. S. (2019). Engineering a less artificial intelligence. *Neuron* 103, 967–979. doi: 10.1016/j.neuron.2019.08.034

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci*. 3, 919–926. doi: 10.1038/78829

Stein, R. B. (1965). A theoretical analysis of neuronal variability. *Biophys. J.* 5, 173–194. doi: 10.1016/S0006-3495(65)86709-1

Van Rossum, M. C., Bi, G. Q., and Turrigiano, G. G. (2000). Stable Hebbian learning from spike timing-dependent plasticity. *J. Neurosci*. 20, 8812–8821. doi: 10.1523/JNEUROSCI.20-23-08812.2000

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xiao, Y. (2003). Random fractals and markov processes. *Math. Preprint Arch*. 2003, 830–907.

Yang, X. (2018). Multifractality of jump diffusion processes. *Ann. Inst. H. Probab. Stat*. 54, 2042–2074. doi: 10.1214/17-AIHP864

Zador, A. M. (2019). A critique of pure learning and what artificial neural networks can learn from animal brains. *Nat. Commun*. 10, 1–7. doi: 10.1038/s41467-019-11786-6

Zhang, W., and Linden, D. J. (2003). The other side of the engram: experience-driven changes in neuronal intrinsic excitability. *Nat. Rev. Neurosci*. 4, 885–900. doi: 10.1038/nrn1248

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2021 Chakraborty and Mukhopadhyay. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

ORIGINAL RESEARCH
published: 08 November 2021
doi: 10.3389/fnins.2021.694549

Mapping Hebbian Learning Rules to Coupling Resistances for Oscillatory Neural Networks

*Corentin Delacour and Aida Todri-Sanial**

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Département de Microélectronique, Université de Montpellier, CNRS, Montpellier, France

Oscillatory Neural Network (ONN) is an emerging neuromorphic architecture with oscillators representing neurons and information encoded in oscillator's phase relations. In an ONN, oscillators are coupled with electrical elements to define the network's weights and achieve massive parallel computation. As the weights preserve the network functionality, mapping weights to coupling elements plays a crucial role in ONN performance. In this work, we investigate relaxation oscillators based on VO_2 material, and we propose a methodology to map Hebbian coefficients to ONN coupling resistances, allowing a large-scale ONN design. We develop an analytical framework to map weight coefficients into coupling resistor values to analyze ONN architecture performance. We report on an ONN with 60 fully-connected oscillators that perform pattern recognition as a Hopfield Neural Network.

Keywords: oscillatory neural network, VO_2 device, coupled relaxation oscillators dynamics, Hopfield Neural Network, Hebbian learning rule, pattern recognition

OPEN ACCESS

Edited by:
Bipin Rajendran,
King's College London,
United Kingdom

Reviewed by:
Zhijun Yang,
Middlesex University, United Kingdom
Nikhil Shukla,
University of Virginia, United States

****Correspondence:***
Aida Todri-Sanial
aida.todri@lirmm.fr

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *13 April 2021*
Accepted: *14 October 2021*
Published: *08 November 2021*

Citation:
Delacour C and Todri-Sanial A (2021) Mapping Hebbian Learning Rules to Coupling Resistances for Oscillatory Neural Networks. Front. Neurosci. 15:694549. doi: 10.3389/fnins.2021.694549

1. INTRODUCTION

Coupled oscillators have been studied for decades by scientists to describe natural phenomena (Winfree, 1967) such as the synchronization of pacemaker cells responsible for the heart beating, the synchronous behavior of insect populations, or to model neuronal activity. For instance, oscillator interactions have been shown to describe memory mechanisms and other cognitive processes in the brain (Fell and Axmacher, 2011). To characterize this variety of natural oscillations, several mathematical models (Acebrón et al., 2005; Izhikevich and Kuramoto, 2006) have been developed to explain the synchronization and phase relations in groups of coupled oscillators. Meanwhile, their massive parallel computing capability has been proved by Hoppensteadt and Izhikevich (2000), Vassilieva et al. (2011), and Parihar et al. (2017) and has raised interest in designing ONN as hardware accelerators for Artificial Neural Networks (ANN) by encoding neurons' activation in the phase between oscillators. Different types of ONN have since been developed using PLLs (Hoppensteadt and Izhikevich, 2000) or oscillators in CMOS technology (Maffezzoni et al., 2015a, 2016; Jackson et al., 2018), demonstrating pattern recognition or resolution of some optimization problems like the Traveling-Salesman-Problem (Endo and Takeyama, 1992). However, to design a competitive ONN at a large scale, a design framework is needed to establish a formalism on how to perform computations with ONN and also compare its energy efficiency with ANNs running on digital processors.

Researchers have developed oscillators by using non-linear devices such as spin-torque-oscillators (Raychowdhury et al., 2019) or with materials presenting a hysteresis resistive state to induce electrical oscillations when properly biased (Sharma et al., 2015; Wang et al., 2017). A compact device that transitions from metallic to insulating state (MIT) can be manufactured with vanadium dioxide (VO_2) (Corti et al., 2018), and has recently become an interesting candidate to design energy-efficient relaxation oscillators. Moreover, coupled-VO_2-based oscillators have been experimentally validated for various applications such as image saliency detection (Shukla et al., 2014), graph coloring (Parihar et al., 2017), filters in Convolutional Neural Networks (Corti et al., 2021) and implementing Hopfield Neural Networks (HNN) for pattern recognition (Corti et al., 2020).

In an HNN defined by Hopfield (1982), every neuron is connected to all the others, and synaptic weights are computed with the Hebbian rule (Hoppensteadt and Izhikevich, 2000). However, setting the right coupling element between oscillators in ONNs remains a challenge (Todri-Sanial et al., 2021). Given N fully-coupled VO_2-oscillators, it is yet unknown how to transform the coefficients obtained analytically via the Hebbian learning rule to coupling resistor values among oscillators. Further, how can one interpret the coefficient signs W_{ij} such as positive or negative values?

For weakly coupled oscillators with sinusoidal waveform, one can use the models that exist in literature (Izhikevich and Kuramoto, 2006; Maffezzoni et al., 2016) for synaptic design. However, it is more difficult for non-linear relaxation oscillators as there is no direct mapping between models and hardware. Two VO_2-oscillators coupled by a capacitance or a resistance have been studied (Maffezzoni et al., 2015b; Parihar et al., 2015), but to the best of our knowledge, there is not yet a formalism to map weights to coupling elements in a larger network. This formalism is a crucial step to allow large-scale ONN design exploration. A greedy approach would be to tune the coupling elements corresponding to the most negative and most positive weights and linearly interpolate all the other weight values. However, this would be impractical. It would require repeated simulations and re-tuning coupling resistances when changing any oscillator parameter; hence, it is not suitable for large-scale ONN design. Parihar et al. (2015) proposed to use capacitors or resistors to implement a negative or a positive weight, respectively. However, it would imply using twice as many components to emulate a complete signed synaptic range.

In this work, we propose a mathematical framework to map both negative and positive Hebbian coefficient values to ONN coupling resistances, as illustrated on **Figure 1**. We first present a single VO_2-oscillator followed by the dynamics of two coupled oscillators. Then, we show that adding switches between oscillators and coupling elements enhances the ONN dynamics control. Based on this simple architecture, we present the ONN computation style and how coupling resistances set the ONN memory expressed in different phase states. Next, by merging oscillators' dynamics with HNN formalism, we introduce the ONN *mapping function* that maps Hebbian coefficient values to coupling resistance values. Finally, we report on the architecture and mapping results by simulating 60 coupled VO_2-oscillators for pattern recognition.

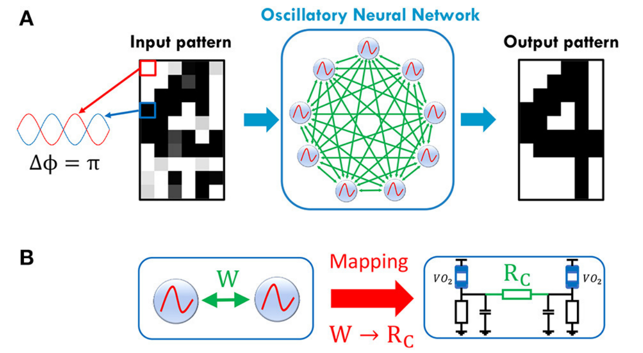


FIGURE 1 | (A) Illustration of ONN as a Hopfield Neural Network (HNN). HNN binary activations are translated in the phase relation between oscillators and a reference (here the first oscillator). ONN stores patterns via weights between the oscillators. Patterns can be retrieved if the input pattern is sufficiently close to the stored pattern. **(B)** The proposed mapping function to map any weight to a coupling resistance, facilitating large scale ONN design.

2. MATERIALS AND METHODS

2.1. Description of the ONN Building Blocks

2.1.1. General Properties

Unlike most ANNs where signals of interest are amplitudes, ONN consists of N VO_2-oscillators coupled by resistances where the final result is given by N-1 phase relations to a reference oscillator (Corti et al., 2018). Despite this difference, there are several aspects common to any ANN that motivate our ONN study:

- The network is composed of neurons (oscillators) having input and output nodes.
- The activation function between the oscillator input and output is non-linear. For more than two oscillators, the activation function is unknown but bounded as the output phase difference of neuron i $\Delta\phi_i^{out}$ is in the range [0; 180°].
- ONN has a memory (coupling resistances) and can therefore be trained to achieve specific functionality.

We use these properties to implement an HNN (Hopfield, 1982) by encoding its binary outputs in the ONN phase as shown in **Figure 1A**. Conveniently, we can represent each oscillator's output as a black pixel if $\Delta\phi_i^{out} = 180°$ or as a white pixel if $\Delta\phi_i^{out} = 0°$.

2.1.2. VO_2 Device

Vanadium dioxide is a material which presents temperature-driven phase change transitions. At room temperature, it remains semiconductor (insulating state) with a monoclinic crystal structure and transitions to a rutile metallic state when the temperature reaches a threshold (Corti et al., 2021). VO_2 presents an hysteresis behavior as it needs to reach a lower temperature threshold to transition back to the insulating state (**Figure 2A**).

When a VO_2 device is in series with a biasing load (**Figure 2B**), the voltage drop V across the device induces Joule heating which can trigger the insulating to metallic transition (IMT). To obtain the VO_2 IV characteristic, we sweep the supply voltage which triggers IMT and MIT when $V \geq V_H$ and $V \leq$

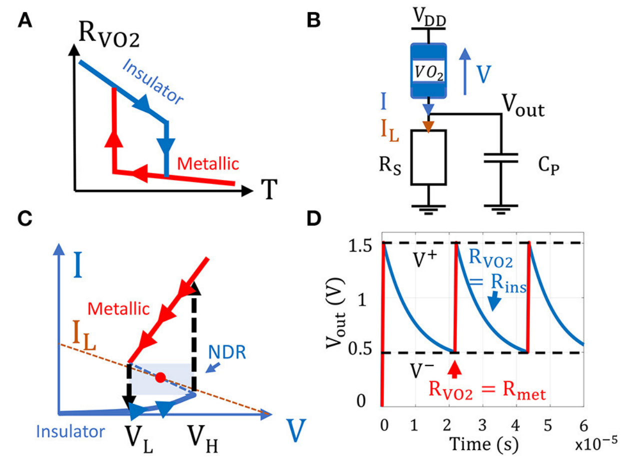


FIGURE 2 | (A) VO_2 resistance vs. temperature. R_{VO_2} axis is in logarithmic scale. **(B)** VO_2 oscillator circuit **(C)** VO_2 I-V curve showing the device hysteresis behavior. When its voltage reaches a threshold V_H, it transitions from insulating to the metallic state. To transition from a metallic to insulating state, V must be lower than the threshold V_L. **(D)** If the device is biased in its Negative Differential Resistance region (NDR), its state alternates between the metallic and insulating state, producing electrical oscillations.

V_L, respectively. The VO_2 hysteresis behavior appears in the IV characteristic with a typical Negative Differential Region (NDR) suitable to bias the device and produce oscillations (**Figure 2C**).

In this work, we use the VO_2 compact model from Maffezzoni et al. (2015b) which reproduces the VO_2 hysteresis along with continuous and abrupt transitions between the two states. The VO_2 hysteresis behavior is conceptually emulated by an amplifier with positive feedback that charges or discharges a RC circuit when the VO_2 is in metallic or insulating state, respectively. The voltage V_c across the capacitor commands the VO_2 conductance G_{VO_2} as:

$$G_{VO_2}(t) = \frac{1 - V_c(t)}{R_{ins}} + \frac{V_c(t)}{R_{met}} \tag{1}$$

Where R_{ins} and R_{met} are the VO_2 resistances in insulating and metallic state, respectively. The dynamics of the VO_2 conductance is given by the RC circuit where τ_0 is its time constant modeling the transition time of the VO_2:

$$\tau_0 \frac{dV_c(t)}{dt} + V_c(t) = 1 - V_0(t) \tag{2}$$

$V_0(t)$ is the output of the positive feedback amplifier (gain α) and is expressed as:

$$V_0(t) = \frac{1}{2}\left[1 + \tanh\left(2\alpha\left((V_H - V_L)\, V_0(t) + V_L - V(t)\right)\right)\right] \tag{3}$$

2.1.3. VO_2 Oscillator Circuit and Dynamics

We bias the VO_2 device with a series resistor R_S to obtain a compact relaxation oscillator. To produce oscillations, the load line I_L set by V_{DD} and R_S must intercept the VO_2 I-V curve in its NDR to obtain an unstable fixed point (**Figure 2C**). The VO_2 device state hence alternates between the metallic and insulating state. When the VO_2 device is in the insulating state, the parallel capacitance C_P at the output node discharges through R_S until the VO_2 voltage reaches V_H and transitions to the metallic state. Then, C_P charges through the VO_2 device until its voltage reaches V_L and a new cycle begins (**Figure 2D**).

TABLE 1 | List of parameters used for simulations in this work.

Parameter	Value
V_{DD}	2.5 V
R_S	20 kΩ
C_P	500 pF
R_{ins}	100.2 kΩ
R_{met}	0.99 kΩ
V_L	1 V
V_H	1.99 V
α	200
τ_0	10 ns
$V^+ = V_{DD} - V_L$	1.5 V
$V^- = V_{DD} - V_H$	0.501 V
T_{osc}	21.6 μs

We use circuit parameters depicted in **Table 1**. In this case, the load resistance R_S is 20 times larger than the VO_2 metallic resistance R_{met}; thus, the capacitance charge time is much faster than its discharge (**Figure 2D**).

Oscillator's dynamics can be described by Kirchoff's law as:

$$C_P \frac{dV_{out}(t)}{dt} = \left(V_{DD}(t) - V_{out}(t)\right) G_{VO_2}(t) - \frac{V_{out}(t)}{R_S} \tag{4}$$

Note that despite the first order differential equation, oscillations can occur as Equations (1–3) describe the hysteresis behavior of $G_{VO_2}(t)$. To solve the oscillator dynamics, we start from an initial insulating VO_2 state and solve numerically on Matlab the system of Equations (1–4) by using Euler's method and Newton-Raphson's algorithm for non-linear Equation (3). **Figure 2D** shows an example where $V_{out}(t = 0) = 0\ V$, $G_{VO_2}(t = 0) = 1/R_{ins}$, and $V_{DD}(t = 0) = 2.5\ V$.

2.1.4. Initialization of Two Coupled Oscillators

Two coupled oscillators represent the smallest ONN and serve as the building block for large-scale ONN. To provide input to the ONN, we delay the second oscillator V_{DD} starting time with respect to the first oscillator (reference oscillator) to set an initial phase relation between them. Assuming oscillators have the same period T_{osc}, we can translate the input delay Δt_{init} as an initial phase relation as:

$$\Delta\phi_{init} = \frac{\Delta t_{init}}{T_{osc}} 2\pi \tag{5}$$

However, if the two oscillators are always connected, they might have different oscillation periods during initialization. Therefore, their initial phase relation cannot be represented as a proportion of T_{osc} (5). For example, as shown in **Figure 3A**, the second oscillator starts $\Delta t_{init} = 0.5\ T_{osc}$ after the first one to set an

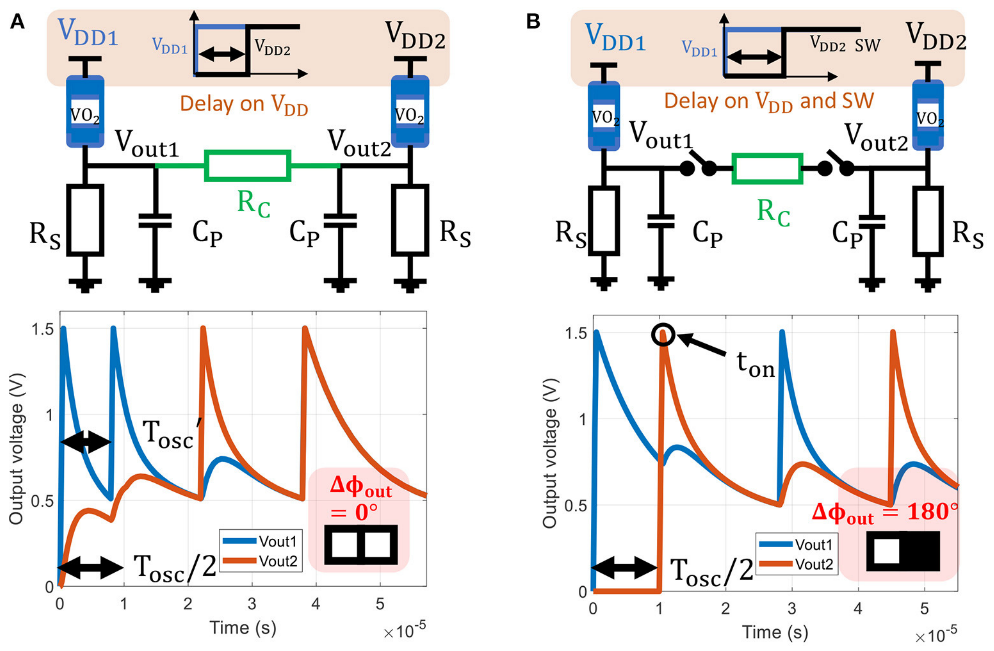


FIGURE 3 | (A) Two oscillators are coupled by a resistance R_C=10 kΩ without coupling switches in between. V_{DD2} is turned-on 0.5T_{osc} after V_{DD1} to set an initial phase of 180°. However, for t <0.5T_{osc} the first oscillator period is decreased due to the shunt R_C at its output node, and we cannot control the initial phase difference. The two oscillators are in-phase after convergence. **(B)** Two oscillators coupled with resistance R_C=10 kΩ and coupling switches between isolate the oscillators during initialization. This time, the 0.5T_{osc} input delay sets desired 180° initial phase state. Here, the switches are closed at $t_{on} = 0.5T_{osc} + t_c$ such that $V_{out2}(t_{on}) = V^+$. The two oscillators converge to an 180° phase state relation.

initial phase relation $\Delta\phi_{init} = \pi$. For $t < \Delta t_{init}$, the second oscillator is off, and its output node is floating. Therefore, during this time, the equivalent load resistance of the first oscillator is $R_S//R_C$, which induces a shorter period of oscillation $T'_{osc} < T_{osc}$ and hence no control on the initial phase.

We introduce switches between each oscillator and coupling elements as in **Figure 3B** to tackle this lack of control. We let each oscillator switch freely with a known oscillation period T_{osc} before coupling them at t_{on}. Their dynamics can be expressed by Equation (4) and the initial conditions will be known at t_{on}. ONN initialization is improved at the cost of one additional switch per oscillator, which can be achieved with a transfer gate.

2.1.5. Dynamics of Two Coupled Oscillators

To predict the output phases and demonstrate ONN ability to store information, we express the dynamics of the two coupled oscillators using Kirchhoff's laws:

$$\begin{cases} C_P \frac{dV_{out1}(t)}{dt} = \left(V_{DD1}(t) - V_{out1}(t)\right) G^1_{VO_2}(t) - \frac{V_{out1}(t)}{R_S} + I_{c1} \\ C_P \frac{dV_{out2}(t)}{dt} = \left(V_{DD2}(t) - V_{out2}(t)\right) G^2_{VO_2}(t) - \frac{V_{out2}(t)}{R_S} + I_{c2} \end{cases} \quad (6)$$

Currents are $I_{c1} = -I_{c2}$ representing the coupling element's current flow. As for the single oscillator case, we numerically solve (Equation 6) along with VO$_2$ (Equations 1–3). **Figure 4** shows a simulation where V_{DD2} is turned on 0.1T_{osc} after V_{DD1} which initializes a light-gray pixel for oscillator 2 input image. For a small coupling resistance, R_C=10 kΩ, ONN converges to a stable state with both oscillators in-phase (0°,0°). Whereas, for R_C=100 kΩ, ONN converges to out-of-phase (0°,180°). In the next subsection, we study the role of R_C on ONN memory and investigate how to retrieve a stored pattern by applying an input delay Δt_{init}. This formulation is the core of our proposed *mapping function* to translate Hebbian coefficients to ONN coupling resistances.

2.1.6. Memory of Two Coupled Oscillators

We solve numerically (6) and extract the output phase relation between oscillators. **Figure 5A** shows the simulation results. As already observed by Corti et al. (2018), a large coupling resistance $R_C > 40$ kΩ induces oscillators in out-of-phase relation (0°, 180°) for any input delay, whereas a small coupling resistance $R_C <$ 10kΩ aligns oscillators in-phase (0°, 0°) for any input delay.

In contrast, we examine the region between these two ranges, highlighted in the center of **Figure 5A**. We observe that for 10kΩ ≤ R_C ≤ 40kΩ both states co-exist and oscillators store two patterns (0°, 0°) and (0°, 180°) that can be retrieved by adjusting the input delay. The line *transition function* between in-phase and out-of-phase regions represents our analytical function for ONN memory with respect to coupling resistance and initialization. It is defined as:

$$\zeta : R_C \longrightarrow \Delta t_{transit} \quad (7)$$

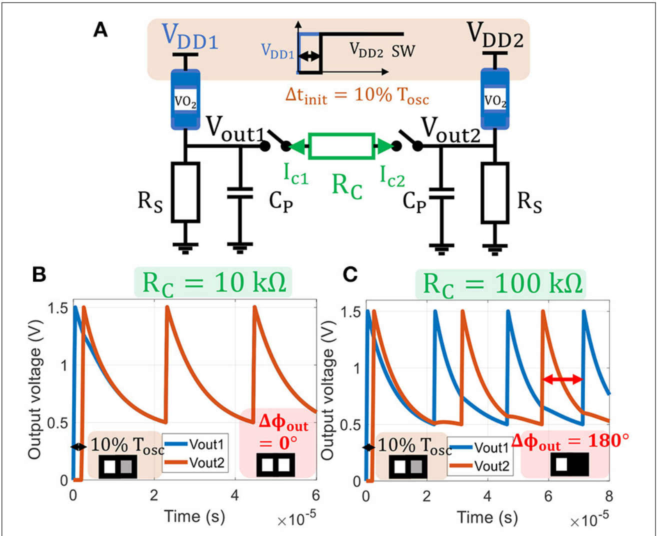


FIGURE 4 | (A) Two identical VO$_2$ oscillators are coupled with a resistance R_C and switches. V_{DD2} starting time and coupling time t_{on} are delayed by $0.1T_{osc}$ with respect to the first oscillator, representing a light-gray second pixel as ONN input. **(B)** Output voltages for $R_C = 10k\Omega$: the oscillators converge to an in phase state (0°, 0°) and the corresponding output pattern corresponds to two white pixels. **(C)** Output voltages for $R_C = 100k\Omega$: the oscillators are out-of-phase (0°, 180°) and the output pattern corresponds to a white and a black pixels.

With $\Delta t_{transit}$ the initial delay such that:

$$\Delta t_{transit} = \zeta(R_C) \mid \begin{cases} \Delta t_{init} < \Delta t_{transit} \Rightarrow \Delta\phi_{out} = 0° \\ \Delta t_{init} \geq \Delta t_{transit} \Rightarrow \Delta\phi_{out} = 180° \end{cases} \tag{8}$$

To confirm the existence of $\zeta(R_C)$, we emulate VO$_2$ oscillators with off-the-shelf components on a Printed Circuit Board (PCB) and we reproduce the experiment of two coupled oscillators. The relaxation oscillator circuit consists of an inverting Schmitt trigger (Schmitt, 1938) Operational Amplifier (OPA) that implements the VO$_2$ hysteresis behavior (**Figure 5B**). The OPA saturates to $+V_{sat}$ and $-V_{sat}$ while the 1.8 nF output capacitor charges and discharges, respectively. **Figure 5C** shows the voltage across the output capacitor for a decoupled oscillator. Similarly to a VO$_2$ oscillator, the OPA transitions to another state when the voltage across the output capacitor reaches V^+ or V^-. The 5.6 kΩ resistor implements the metallic VO$_2$ resistance, whereas the 100 kΩ resistor corresponds to the load R_S. For a fixed oscillating period of T_{osc}=200 μs, we vary R_C and we measure $\Delta t_{transit}$ values that define the experimental transition function $\zeta(R_C)$ (**Figure 5D**). There is a good match between experimental $\zeta(R_C)$ data points and the analytical transition function derived in next subsection. Such formulation $\zeta(R_C)$ is of interest as it represents a closed-form representation of ONN memory instead of repeating numerical simulations for different oscillator parameters.

2.1.7. Phase Transition Function for Two Coupled Oscillators

The phase transition function has already been observed (Nez et al., 2021) but to the best of our knowledge, no closed-form expression has ever been reported. To obtain the transition function, we solve node voltage equations analytically for two coupled oscillators during initialization (see **Supplementary Material**). We derive oscillator outputs as:

$$\Delta V = V_{out2} - V_{out1} = \left(V^0_{out2} - V^0_{out1}\right)\exp\left(-\frac{t}{\tau'}\right) \tag{9}$$

V^0_{out1} and V^0_{out2} are the initial voltages when oscillators are coupled and τ' is defined in **Supplementary Material** Equation (S18). Equation (9) describes both oscillator output voltages attracted via the coupling resistance R_C. If the coupling is strong enough (small R_C), both oscillators are rapidly pulled together with a speed determined by τ'. If $\Delta V < \epsilon$ (ϵ defined in **Supplementary Material** Equation S22) before reaching the VO$_2$ threshold V^-, then both oscillators will transition to low resistive states, and the exponential term in Equation (9) will keep the two voltages locked. This concept is illustrated in **Figure 6A** when both oscillators are in-phase. However, if V_{out1} reaches V^- before V_{out2} such that $\Delta V > \epsilon$ as in **Figure 6B**, the first oscillator transitions to a low resistance state (metallic state) while the other oscillator is still in high resistance state (insulating state). The two oscillators are then in opposite states, and this leads to out-of-phase relation.

Thus, to obtain the transition function ζ (Equation 7), we study the case when both $\Delta V = \epsilon$ and $V_{out1} = V^-$ conditions are fulfilled (see **Supplementary Material** for details). By combining (Equations S17, S24, and S25 in **Supplementary Material**) we derive coupling resistance as:

$$R_C = 2\frac{R_S R_{ins}}{R_S + R_{ins}} \frac{\log\left(\frac{V^- - V^{ins}_{std}}{V^0_{out1}/2 + V^0_{out2}/2 - V^{ins}_{std}}\right)}{\log\left(\frac{\epsilon(R_C)}{V^0_{out2} - V^0_{out1}}\right) - \log\left(\frac{V^- - V^{ins}_{std}}{V^0_{out1}/2 + V^0_{out2}/2 - V^{ins}_{std}}\right)} \tag{10}$$

where V^{ins}_{std} is defined in **Supplementary Material** Equation (S5). Finally, we introduce (Equation S9 in **Supplementary Material**) into Equation (10), and obtain a relation between R_C and $\Delta t_{transit}$. Note that ϵ is a function of R_C, thus we cannot solve (10) analytically. Instead, we numerically solve (Equation 10) using Newton-Raphson's algorithm for $\Delta t_{transit}$ values. We finally obtain R_C values that describe the inverse of the transition function as:

$$\zeta^{-1} : \Delta t_{transit} \longrightarrow R_C \tag{11}$$

Transition function ζ is plotted as the curve line (red line) in **Figure 5A**, and there is an excellent fit between our analytical model, simulations (transition region between dark and light green in $(R_C, \Delta t_{init})$ plan) and the transition curve obtained experimentally with off-the-shelf relaxation oscillators (**Figure 5D**). In addition, we can now extract the coupling resistor R_0 that corresponds to a neutral synaptic connection $W = 0$. As by definition, both output phase states can equally occur for $W = 0$, we extract R_0 as:

$$R_0 = \zeta^{-1}(\Delta t_{init} = T_{osc}/4) \tag{12}$$

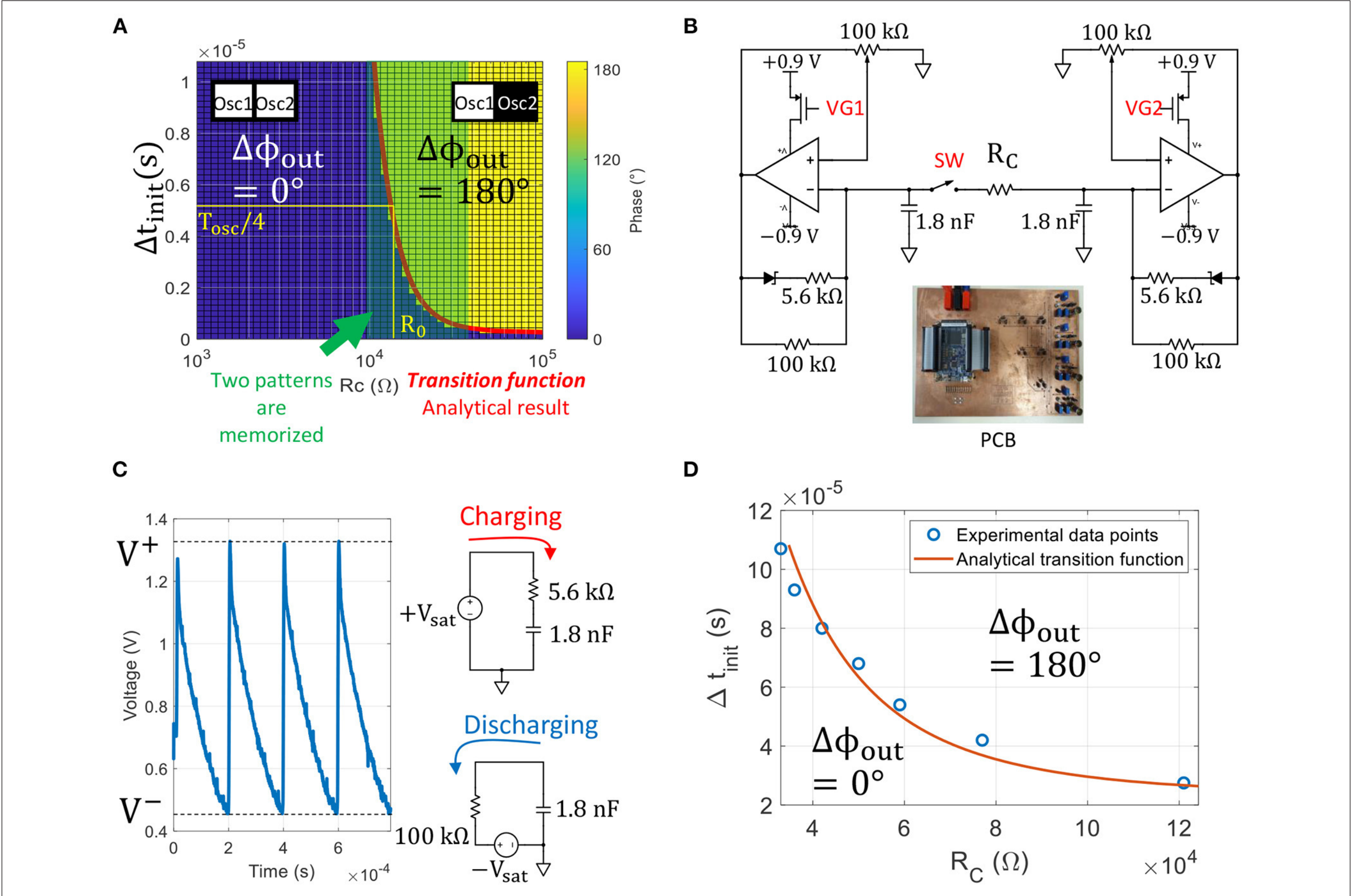


FIGURE 5 | (A) Plot showing the phase relation between two oscillators for every set of parameters (R_C, Δt_{init}). As expected, small coupling resistances tend to pull the oscillator phase together, whereas large coupling resistances push the phase away. The green region shows the coupling resistance range 10kΩ $< R_C <$ 40kΩ in which two patterns (0° and 0°) and (0° and 180°) are memorized and can be retrieved by adjusting the input delay. The red curve is our analytical model describing the *transition* between the two phase states in the plan (R_C, Δt_{init}), and plays a major role in the ONN ability to memorize patterns. **(B)** Experimental set-up of two coupled relaxation oscillators based on MCP6001 OPAs. We delay VG2 with respect to VG1 by Δt_{init} to set the initial phase, and we close SW after initialization. **(C)** Experimental oscillating waveform and equivalent circuits during charge and discharge of the output capacitor. **(D)** Experimental phase transition curve and analytical model in plain line.

Finally, based on the transition function, we predict the final phase relation as:

$$\Delta\phi_{out} = 180^\circ \left(sign\left(R_C - \zeta^{-1}(\Delta t_{init})\right) + 1\right)/2 \qquad (13)$$

Analogous to ANNs, Equation (13) can be thought of as oscillator's activation function. Because, it provides the oscillator's output phase based on its input phase (set by Δt_{init}; Equation 5) and the weight implemented by R_C.

2.1.8. Impact of VO_2 Parameters Variations on the Phase Transition Function

Fabricating reliable VO_2 devices is challenging (Corti et al., 2019) and ONN experiments with VO_2 are currently limited to few devices because of device variability (Shukla et al., 2016). Here, we study the impact of VO_2 variability on the ability to phase-lock and on the synaptic range. The transition function $\zeta(R_C)$ defines the boundary between two phase regions, and allows direct identification of the neutral coupling resistor R_0 corresponding to the weight $W = 0$ (Equation 12). Thus, we use ζ and R_0 as metrics to assess the impact of VO_2 parameters' variations. We apply relative variations on VO_2 parameters one at a time from –20% up to +20%, as shown in **Figure 7A**. Note that we vary V_H from –4% up to +4% only, as for larger positive variations oscillations do not occur (load line crosses the insulating branch and forms a fixed point). For all cases, we discretize the whole input space ($R_C, \Delta t_{init}$) and perform multiples transient simulations to extract the new phase regions. Then, we numerically solve (Equation 10) with the new sets of parameters and we verify that the transition function ζ matches the phases boundary obtained via transient simulations, as in **Figure 5A**.

Figure 7A shows the set of phase transition curves obtained when varying R_{met}, R_{ins}, V_L and V_H. Note that our current formalism assumes matched oscillators and hence, variations are

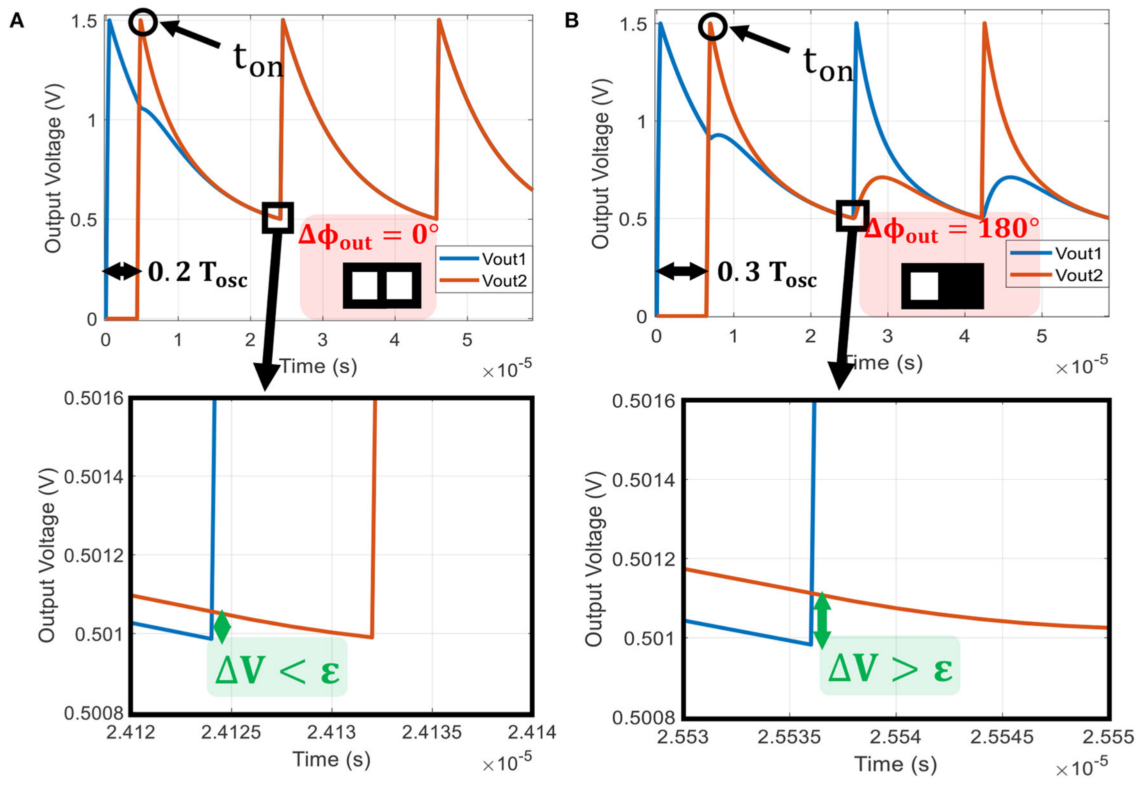


FIGURE 6 | Two identical oscillators coupled by $R_C = 12\text{k}\Omega$. **(A)** The second oscillator is turned on at 0.2 T_{osc} after the first one. By zooming on the waveform when the first oscillator reaches V^-, we observe a voltage difference $\Delta V < \epsilon$. Therefore, the oscillators converge to an in-phase state. **(B)** The second oscillator is turned-on 0.3 T_{osc} after the first one. In this case, we observe a voltage difference $\Delta V > \epsilon$ and the oscillators converge to an out-of-phase state.

applied to both coupled oscillators. For all curves, the maximum Δt_{init} value corresponds to an input delay of $T_{osc}/2$ and shows the oscillation period variation (highlighted in green in **Figure 7A**). Finally, we extract R_0 for each configuration (**Figure 7B**). We observe that variations on the IMT point (defined by R_{ins} and V_H) induce the largest T_{osc} and R_0 variations. With our biasing set by R_S and V_{DD} (**Table 1**), the most sensitive VO_2 parameter is V_H as +4 and –4% V_H variations induces +40 and –20% R_0 variations, respectively. As the dynamic of the voltage V across the VO_2 device is given by $C_P \ \ dV/dt = I_L - I$, we believe this sensitivity is mainly due to the load line that passes very close to the IMT point on the VO_2 *IV* characteristic (**Figure 7C**). In this case near IMT, $I_L(V_H) - I(V_H)$ is small and the voltage "slows down" and is very sensitive to any IMT variation. When applying –4% up to +4% V_H variations, the oscillating period T_{osc} almost doubles (same remark with –20 and +20% R_{ins} variations). Ideally, we would then place the load line at equal distances between MIT and IMT points ($I(V_L) - I_L(V_L) \approx I_L(V_H) - I(V_H)$) to homogenize the impact of VO_2 variations. However, we show in the next subsection that such biasing would prevent binary phase locking and that resistively coupled oscillators need a very asymmetric waveform to phase-lock to 180°.

2.1.9. Impact of Oscillators' Waveform Shape on ONN Phase-Locking

Oscillators' circuit parameters listed in **Table 1** influence the oscillating frequency, amplitude and waveform shape. The oscillating waveform shape has a major influence on ONN phase-locking capability and has been studied for PLL-based ONNs by Hoppensteadt and Izhikevich (2000). Here, we study the impact of the oscillating waveform shape on the capability for pairs of oscillators to lock to the 180° phase state. We characterize the oscillating waveform shape with the ratio τ_d/τ_c, where τ_d and τ_c are the discharging and charging time constant, respectively (defined in **Supplementary Material** Equations S6, S7). Our transition function ζ links ONN phase-locking properties to the metric τ_d/τ_c, as ζ only depends on oscillators' internal parameters.

We reproduce the previous simulation with two coupled oscillators to extract the output phase regions for different load resistances R_S that set τ_d/τ_c (**Figure 8A**). Note that we also could have varied VO_2 parameters such as R_{met}, but instead we consider the same device. For $\tau_d/\tau_c = 3.7$ ($R_S = 3\ \text{k}\Omega$), we observe that the two oscillators cannot lock to $\Delta\phi_{out} = 180°$ for small Δt_{init} values. In other words, the phase state $\Delta\phi_{out} = 180°$ stored by a large R_C cannot be fully recovered. This can be an issue for some

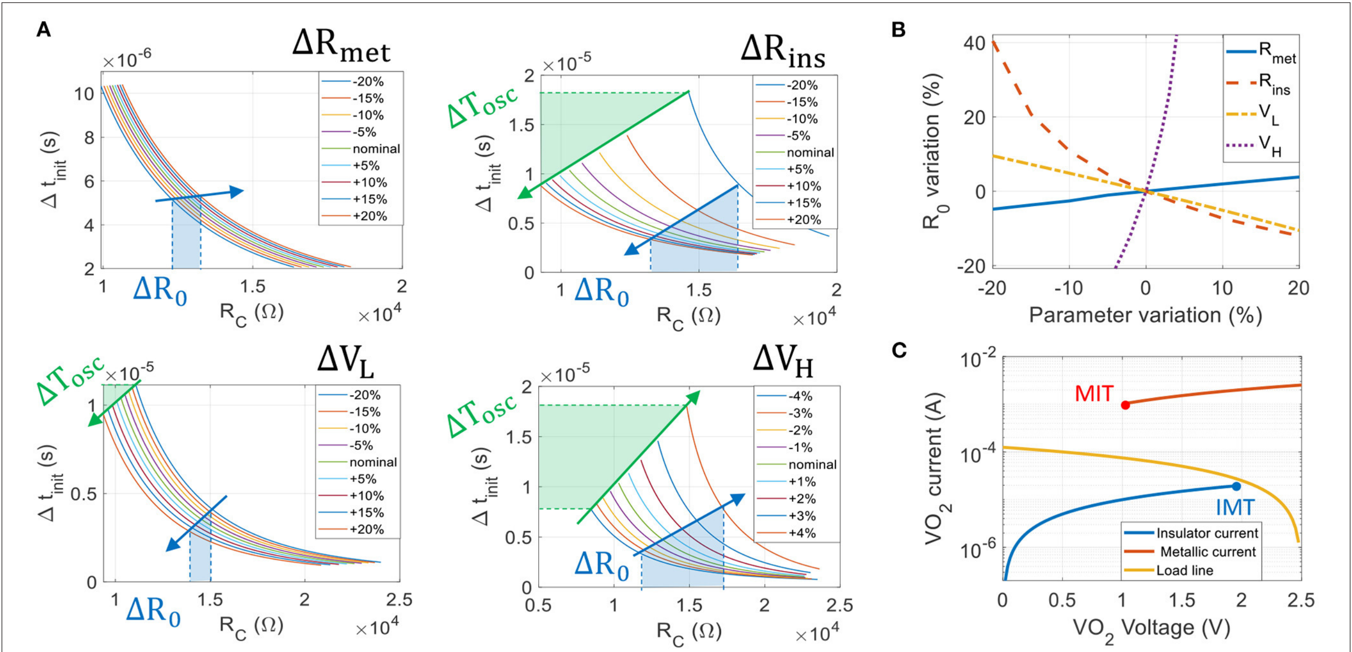


FIGURE 7 | (A) Phase transition curves $\zeta(R_C)$ when varying VO$_2$ parameters R_{met}, R_{ins}, V_L from –20 to +20%, and V_H from –4 to +4% for circuit parameters listed in **Table 1**. Left hand side of the transition curve corresponds to inputs $(\Delta t_{init}, R_C)$ inducing $\Delta\phi_{out} = 0°$ whereas right hand side corresponds to $\Delta\phi_{out} = 180°$ phase region. V_H and R_{ins} variations correspond to IMT point variations and have the most detrimental impact on the transition function variations. The oscillating period almost doubles due to IMT variations. **(B)** Variations of the neutral synaptic resistance R_0 with respect to VO$_2$ parameters' variations. R_0 is very sensitive to V_H as –4 and +4% V_H variations induce –20 and +40% R_0 variations, respectively. **(C)** The ONN sensitivity to IMT point is mainly due to the load line $I_L = (V_{DD} - V)/R_S$ placed close to IMT point. Any IMT variation greatly impacts the oscillators' dynamics defined by $C_P\, dV/dt = I_L - I$.

pairs of oscillators that need an out-of-phase relationship for any input delay.

If $\tau_d/\tau_c = 59$ ($R_S = 20$ kΩ), the charging time is much smaller than the discharging time and the oscillating waveform becomes very asymmetrical. Interestingly, this configuration enlarges the 180° phase region and $\Delta\phi_{out} = 180°$ is reachable for any Δt_{init} value for large R_C. Our analytical model $\zeta(R_C)$ predicts the correct boundary between the two phase regions (red plain lines in **Figure 8A**).

We study a simple case where 4 VO$_2$-oscillators are coupled by resistances to store a single pattern (**Figure 8B**). Based on transition functions obtained for 2 coupled oscillators, we compute coupling resistances R_{+1} and R_{-1} that correspond to synaptic coefficients +1 and –1, respectively. We set R_{+1} and R_{-1} around R_0 as $R_{+1} = \zeta^{-1}(T_{osc}/4 + T_{osc}/8)$ and $R_{-1} = \zeta^{-1}(T_{osc}/4 - T_{osc}/8)$, respectively. Then, we scale coupling resistances as 3xR_{+1} and 3xR_{-1} as every oscillator is connected to 3 others (**Figure 8B**). We notice that ONN with $\tau_d/\tau_c = 59$ retrieves the correct stored pattern whereas ONN with $\tau_d/\tau_c = 3.7$ produces a wrong output (**Figure 8C**). In the latter case, we observe that all oscillators converge to an in-phase relationship. We believe this wrong behavior is mainly due to the small τ_d/τ_c value for which it is less likely ONN converges to $\Delta\phi_{out} = 180°$, as described by the transition function ζ.

Figure 9 shows results of the same experiment for τ_d/τ_c varied from 1.8 up to 59 (obtained for 2 kΩ $\leq R_S \leq$ 20 kΩ). We observe that $\tau_d/\tau_c > 20$ is required to retrieve the correct pattern. Interestingly for $\tau_d/\tau_c \leq 20$, there are cases where the fourth oscillator locks to a phase state around 270°. 270° phase value is also obtained in the phase plot between two coupled oscillators, such as on the left-hand side of **Figure 8A**. This phenomenon would allow more than two phase values but is not captured by our current formalism. In contrast, we set τ_d/τ_c to high values (59 in this work) to ensure binary 0° and 180° phase locking.

To implement a large-scale neural network such as HNN with an ONN, we need a systematic approach to map the weights to ONN coupling resistances. In the next section, we exploit some HNN features and our knowledge of two coupled oscillators to propose a *mapping function*.

2.2. ONN Weight Mapping

2.2.1. Applying HNN Formalism to ONN

We exploit HNN formalism to build an analogous representation in ONN. For equivalence, we treat HNN neurons similar to ONN oscillators. Such as, we consider a neuron i with two possible states S_i that can be thought of as equivalent to ONN oscillators with 0° or 180° phase relations as:

$$S_i = \begin{cases} +1 \\ -1 \end{cases} \iff \Delta\phi_i = \begin{cases} 0° \\ 180° \end{cases} \tag{14}$$

In HNN, each neuron output state is dynamically determined by a sigmoid activation function $g(x) = \big(\tanh(\beta x) + 1\big)/2$ (with β a positive parameter) (Gerstner et al., 2014) and shown in

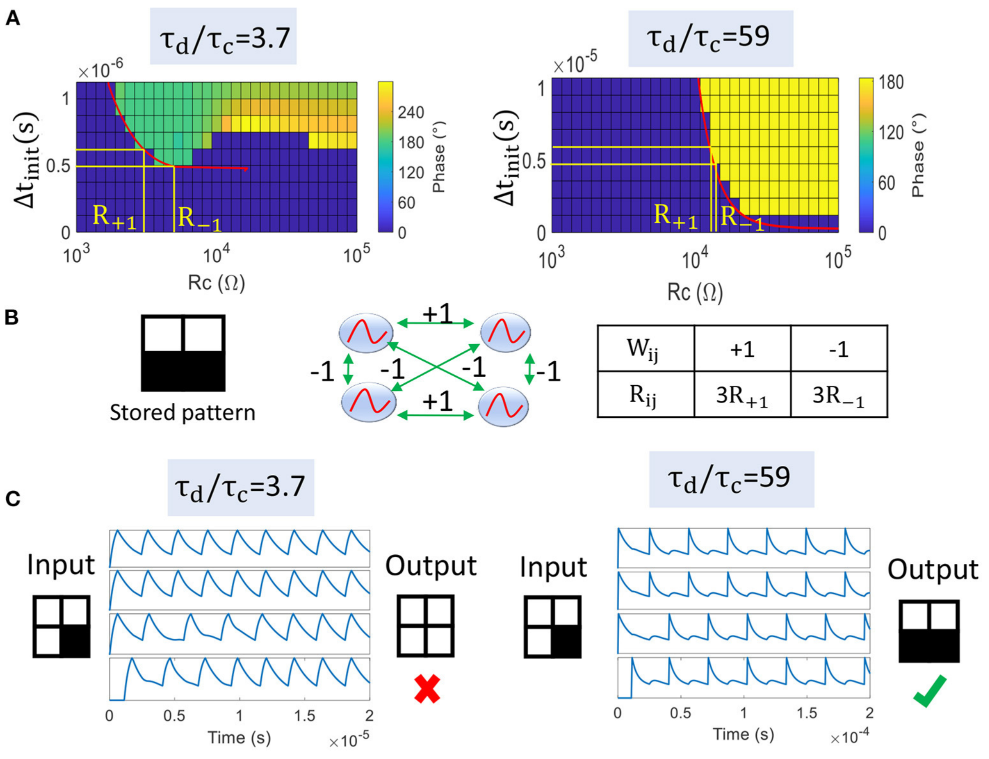


FIGURE 8 | (A) Phase plots showing $\Delta\phi_{out}$ with respect to Δt_{init} and coupling resistance R_C between two oscillators. For $\tau_d/\tau_c = 3.7$, 180° phase state is not reachable for low Δt_{init} values. In contrast for $\tau_d/\tau_c = 59$, 180° phase-locking can occur for any Δt_{init} value for large R_C. The red line is our analytical model $\zeta(R_C)$ and captures well the boundary between phase regions. **(B)** Four coupled oscillators store a pattern composed of 2 white and 2 black pixels. Positive and negative weights are mapped to 3xR_{+1} and 3xR_{-1}, respectively. **(C)** ONN inference for $\tau_d/\tau_c = 3.7$ and $\tau_d/\tau_c = 59$. The first configuration leads to a wrong in-phase relationship for all oscillators. In the latter case, ONN retrieves the correct stored pattern.

Figure 10. For a neuron i, g gives the probability to reach one of the two states at $t + \Delta t$ for a given input weighted sum $h_i(t)$ as

$$P\Big(S_i(t+\Delta t) = +1 \mid h_i(t)\Big) = g\big(h_i(t)\big) \tag{15}$$

with

$$h_i(t) = \sum_{j=1}^{N} W_{ij}\, S_j(t) \tag{16}$$

In ONNs, Equation (15) would represent the probability of oscillator i to be in-phase with the reference at time-step Δt. For two oscillators case, the weighted input sum of the second oscillator is given by:

$$h_2(t) = W_{21}\, S_1(t) = W_{21} \tag{17}$$

Then, the probability of the second oscillator to be in-phase with the reference can be derived by Equations (15) and (17), as:

$$P\Big(S_2(t+\Delta t) = +1 \mid h_2(t)\Big) = P_{inphase} = g(W_{21}) \tag{18}$$

2.2.2. Mapping Function

Here, we apply the above definitions to derive a mapping function using the HNN formalism, as:

$$\mu_N : W_{ij} \longrightarrow R_{ij} \tag{19}$$

where, normalized weights are $-1 \leq W_{ij} \leq 1$ and N is ONN size. Before scaling to N oscillators, we derive a mapping function μ_2 for two coupled oscillators. The unifying step between HNN and ONN is the recasting of the phase transition curve, ζ as the probability $P_{inphase}$ for a given coupling resistance R_C. In ONNs, the input delay Δt_{init} can be considered as a uniform random variable taking values between 0 and $T_{osc}/2$ and the transition function ζ would give the probability $P_{inphase}$ (for example for $R_C > 10k\Omega$):

$$P_{inphase} = \zeta(R_C)\,\frac{2}{T_{osc}} \tag{20}$$

and by Equations (18) and (20), we finally obtain

$$\mu_2(W_{21}) = R_C = \zeta^{-1}\left(\frac{T_{osc}}{2} g(W_{21})\right) \tag{21}$$

This mapping function is represented in **Figure 11** for three different values of the sigmoid parameter, β. We see that this

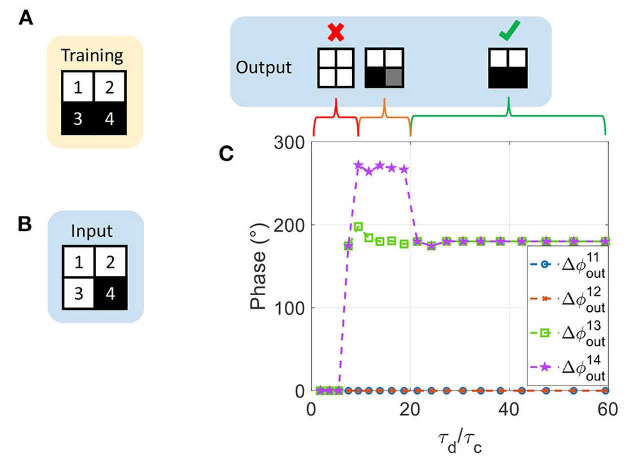


FIGURE 9 | (A) Training pattern stored by the ONN. **(B)** ONN input pattern. **(C)** Output phase with respect to τ_d/τ_c. For $\tau_d/\tau_c < 7$, all oscillators converge to a wrong in-phase relationship. For $7 \le \tau_d/\tau_c < 20$, the fourth oscillator locks to a 270° phase state. A very asymmetrical waveform such that $\tau_d/\tau_c \ge 20$ leads to a correct binary phase-locking.

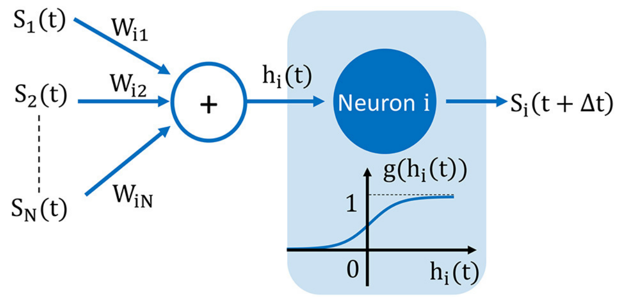


FIGURE 10 | Model of artificial neuron used to construct our mapping function μ_N. The neuron's output state $S_i(t)$ is either +1 or −1 and is dynamically updated at each time-step Δt according to the sigmoid activation function g. Here, g gives the probability to have one of the two states at $t + \Delta t$ for a given input weighted sum $h_i(t)$.

parameter sets the range of R_C and could be adapted for different ONN sizes. Interestingly, we notice that $|\Delta W_{21}/\Delta R_C|$ is quite large for a positive weight, whereas it is much smaller for a negative one. For example, we see in **Figure 11B** that the function ζ^{-1} is a logarithmic function; thus, any small variation in ΔR_C around 10kΩ is likely to change the final phase state outcome. Whereas, for large R_C, the two oscillators are almost always out-of-phase. This asymmetry in ζ^{-1} comes from the oscillator waveform type, as ζ^{-1} is derived from Equation (10), which is specific for relaxation oscillator waveform type. Hence, we expect some change for other types of waveforms, such as linear sawtooth, but the formulation of mapping (21) is general enough to be applied to any relaxation oscillators.

For large-scale ONN with N oscillators, we scale μ_2 (21) by a factor $N - 1$ to ensure the conservation of the current flow in coupling resistances. We finally obtain:

$$\mu_N(W_{ij}) = R_{ij} = (N-1)\,\zeta^{-1}\left(\frac{T_{osc}}{2}g(W_{ij})\right) \tag{22}$$

In next section, we demonstrate the effectiveness of the proposed mapping function (22) to design a 60-ONN architecture for pattern recognition as in HNN.

3. RESULTS

3.1. ONN Design for Pattern Recognition

3.1.1. ONN Training and Mapping

In the previous section, we presented the memory capability of two coupled oscillators. Here, we apply the analytical formulas to a larger ONN size. We develop a design flow as shown in **Figure 12A** for pattern recognition with ONNs where we have implemented the proposed mapping function. We first compute the weights associated with the M stored patterns using the Hebbian Rule (Hoppensteadt and Izhikevich, 2000), as:

$$W_{ij} = \frac{1}{N}\sum_{k=1}^{M}\xi_i^k\xi_j^k \tag{23}$$

We store $M = 6$ images representing digits "0", "1" to "5" as shown in **Figure 12B**. Next, we use our mapping function to compute the coupling resistances associated with the Hebbian coefficients. The mapping is represented in **Figure 12C** for different values of parameter β which sets the slope of $\mu_N(W_{ij})$. Increasing β induces a larger coupling resistance range. Because the Hebbian rule normalizes the weights by the network's size N (23), we scale β with N to keep a relative standard deviation of R_{ij} approximately constant when increasing the ONN size. By simulations, we found that the best accuracy is obtained for $\beta = N/32$, and we report the results in subsection 3.1.3.

3.1.2. ONN Inference

For every input, we set up the 60 ONN with black and white pixels encoded by −1 and +1, respectively. For pixels with black input, the corresponding oscillator is initialized with a delay $\Delta t_{init} = T_{osc}/2$ to set an initial out-of-phase relation (5) whereas, for a white pixel, no delay is introduced. A noisy gray pixel corresponds to an input delay between 0 and $T_{osc}/2$. After few oscillations, the ONN settles, retrieves the noiseless pattern and phase relations are measured. An example of ONN voltage dynamics is presented in **Figure 13A**, showing the initialization and the settling time before the ONN stabilizes. **Figures 13B,C** show two examples of input images where 15 pixels have been randomly altered by a uniform distribution taking values between −1 and +1. When the number of noisy input pixels is too large such as in **Figure 13D** (20 noisy pixels), ONN converges toward a wrong spurious state that is different from the stored patterns. The results are in accordance with original observations from Hopfield (1982), proving that our mapping can implement HNN with ONN.

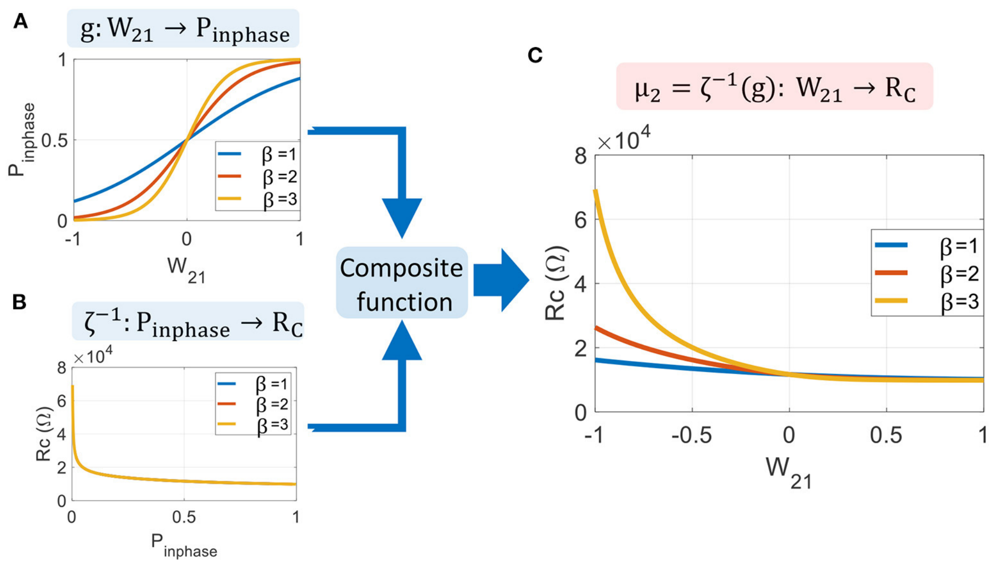


FIGURE 11 | Mapping function for two coupled oscillators. **(A)** Sigmoid activation function presents the probability $P_{inphase}$ for the two oscillators to be in phase. **(B)** Inverse of the transition function ζ^{-1} determines the coupling resistance R_C for a given probability $P_{inphase}$. **(C)** The mapping function μ_2 is obtained by the composite function $\zeta^{-1}(g)$.

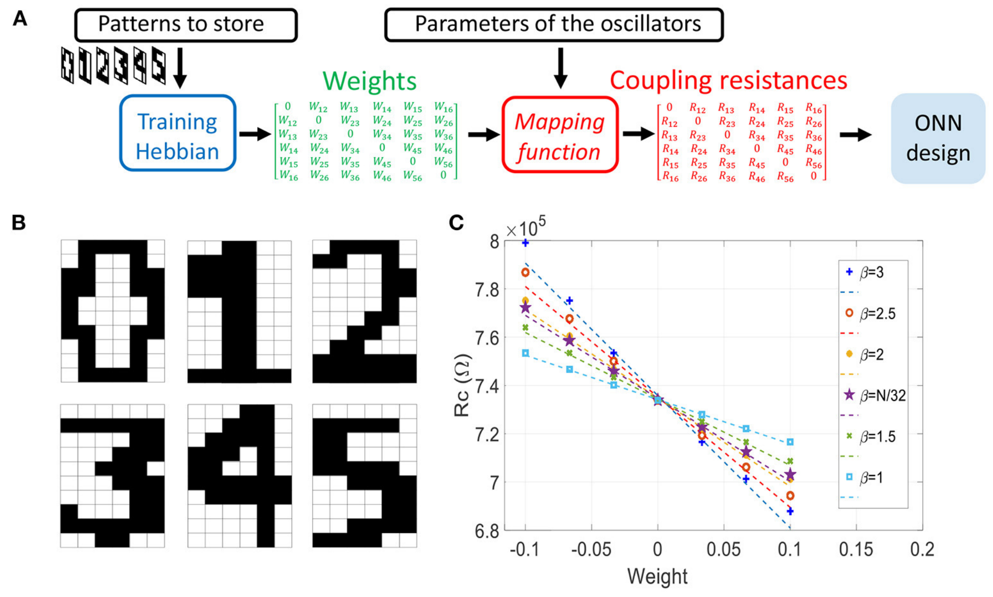


FIGURE 12 | **(A)** Illustration of the ONN design flow for the associative memory application. Patterns to store can be represented as black and white images from which we compute weights with the Hebbian rule during the training process. Then, we use the mapping function μ_N to get the coupling resistances, allowing a systematic ONN design. **(B)** Stored patterns. **(C)** Coupling resistances as a function of Hebbian weights, computed with the mapping function μ_N for different values of parameter β.

3.1.3. ONN Recognition Accuracy

Here, we perform simulations to compute the pattern recognition accuracy of the 60-ONN. We randomly apply noise to training patterns to generate a test set. It consists of 20 different subsets S_k, $k \in \{1, 2, .., 20\}$ in which 60 different test patterns have k randomly located fuzzy input pixels. We vary the mapping function parameter β to assess its influence on ONN accuracy. We notice from **Figure 14C** that ONN achieves the best accuracy for an optimum value $\beta = N/32 = 1.875$. In this case, ONN recognizes more than 80% of test images with up to 20% of

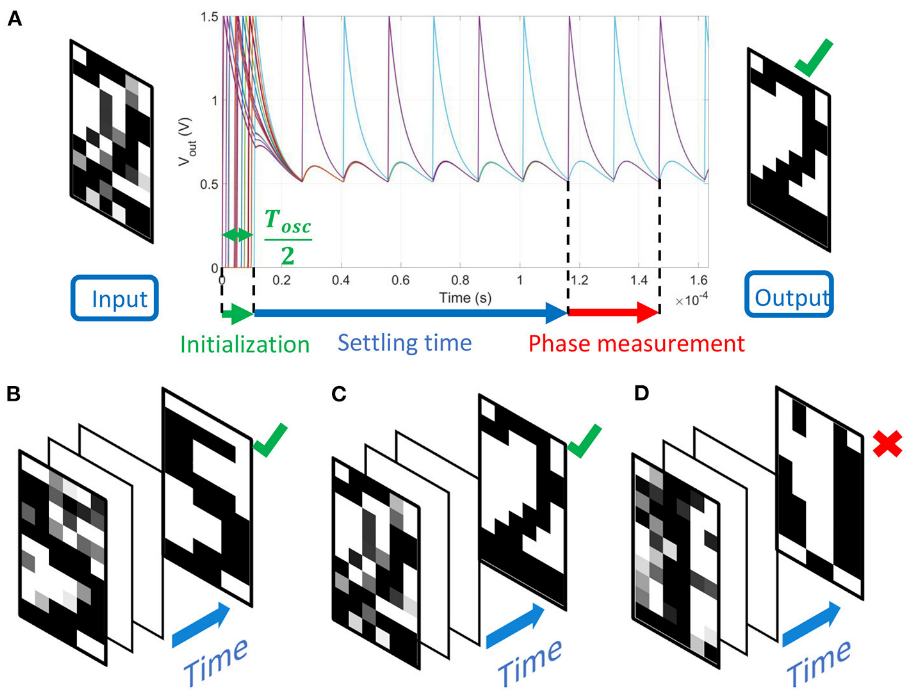


FIGURE 13 | (A) Noisy input image "2" with 15 random altered pixels and voltage waveforms of 60 oscillators. ONN is initialized during $T_{osc}/2$ with the noisy input image. After few oscillation cycles, ONN settles, and phases are measured. ONN retrieves the correct output image that corresponds to the stored pattern "2." **(B)** The input image "5" has been altered at 15 random pixel locations by a uniform distribution. ONN retrieves the corresponding stored pattern. **(C)** Similarly, 15 random pixels of the input image "2" are altered, and ONN retrieves the correct corresponding pattern. **(D)** In this example, 20 noisy input pixels are introduced to digit "1," and ONN converges toward a spurious state.

noise. As seen in **Figure 12C**, the slope parameter β sets the coupling resistance range, which in turn affects ONN accuracy. For instance, we observe that the set of coupling resistances obtained for $\beta = 2$ is similar to the case $\beta = 2.5$, but the accuracy is much lower in the latter case. In the next section, we quantify the coupling resistance accuracy that is required for synaptic design.

3.1.4. ONN Coupling Resistance Range

We study the impact of R_C's relative variations and R_C's mean value. R_C^{min} is the minimum resistance common to all coupling resistances, and ΔR is the additional series resistance to distinguish between weights (**Figure 14A**). Using the Hebbian rule, weights are located near "0" coefficient as in **Figure 12C** and our mapping function can be fitted linearly (dashed lines). Therefore, every coupling resistances can be approximated by $R_C \approx R_C^{min} + n\Delta R^{min}$ with $n \in \{0, 1, 2, .., M\}$. Using this linear approximation, we can verify ONN accuracy is similar to the nominal case of mapping function with $\beta = N/32$, as shown in **Figure 14C** with the magenta dashed line.

As observed in previous sections, ONN accuracy is quite sensitive to the coupling resistances. We obtain $\Delta R^{max} \approx$ 15% R_C^{min} for $\beta = 3$, and $\Delta R^{max} \approx$ 5% R_C^{min} for $\beta = 1$ (**Figure 14B**). For these two cases as shown in **Figure 14C**, ONN shows poor accuracy. It is rather for $\beta = N/32 = 1.875$ with $\Delta R^{max} \approx$ 10% R_C^{min} that ONN accuracy is above 90%.

To achieve the best ONN accuracy, a very good resistor matching is required, as we need a precision of $\Delta R^{min} =$ 1.7% R_C^{min} between two consecutive weights. To study the influence of the R_C's mean value only, we apply the same variation to all coupling resistances for $\beta = N/32$ and for 10 fuzzy input pixels (**Figure 14D**). We notice that the mean value of coupling resistances can vary from −10% up to +5% from its nominal value to achieve a similar accuracy.

4. DISCUSSION

Oscillatory neural networks are triggering great interest for parallel processing applications, but a remaining challenge is how to compute with ONNs. To do so, we build analogies with ANN to determine the mapping between Hebbian learning coefficients (weights) to coupling resistors, knowing that they are essential elements for the network functionality. In this work, we proposed a mapping function that translates Hebbian signed weights to coupling resistances in a VO_2-based ONN for systematic ONN analysis. Our simulations on 60-oscillators test case study highlighted a strong dependency between the ONN recognition accuracy and its coupling resistance range, set by mapping parameter β. Although we identified a suitable value $\beta = N/32$ to achieve good ONN accuracy, our mapping formulation provides coupling resistances that differ only with few percent. This would lead to significant

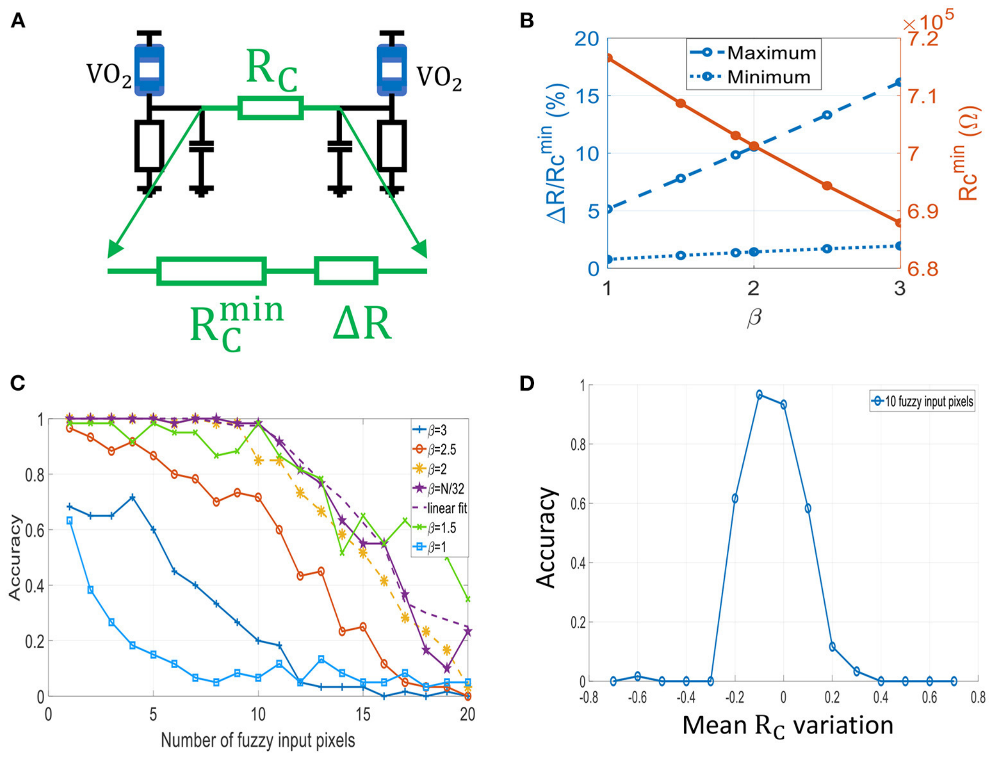


FIGURE 14 | (A) Two oscillators coupled by R_C, which can be decomposed in two series resistances: $R_C = R_C^{min} + \Delta R$. **(B)** Evolution of ΔR, R_C^{min} with respect to β. $\Delta R^{max} \approx 10\%\ R_C^{min}$ gives the best accuracy results. Note that $\Delta R^{min} = 1.7\%\ R_C^{min}$. **(C)** ONN recognition accuracy for different values of β. The dashed line is obtained for a linear fit of the mapping μ_N, i.e., with coupling resistances that are linearly spaced. **(D)** Impact of R_C's mean variation on recognition accuracy.

hardware design constraints, as resistor mismatches smaller than 1.7% would be required to emulate two consecutive weights. In our mapping formalism, we used the phase transition function ζ, which provides the coupling resistance range holding the ONN memory. As we only derived ζ from oscillator dynamics, we believe the oscillator design could be optimized to expand the coupling resistance range and relax synaptic design constraints. For example, oscillator biasing current and supply voltage are the knobs that could be adjusted to maximize the synaptic range.

Here, we reported on a *mapping function* to compute coupling resistances from signed Hebbian coefficients in a VO_2-based ONN. We first enhanced the ONN initialization control based on a simple architecture where every oscillator can be decoupled from the network via a switch. We were able to derive the phase *transition function* from the ONN dynamics, which is crucial for ONN memory. We then merged this analytical formulation with a sigmoid activation function from ANN formalism to build a mapping function. To demonstrate the ONN architecture's applicability with the proposed mapping function, we presented a test case of pattern recognition with 60 fully coupled oscillators. Finally, we showed that ONN recognition accuracy is very sensitive to relative variations between coupling resistances.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

CD and AT-S developed the analytical formulation of ONN mapping and wrote the article. CD implemented the ONN circuit-solver framework on Matlab and performed simulations. Both authors contributed to the article and approved the submitted version.

FUNDING

This work is supported by the European Union's Horizon 2020 research and innovation program, EU H2020 NEURONN (www.neuronn.eu) project under Grant No. 871501.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.694549/full#supplementary-material

REFERENCES

Acebrón, J. A., Bonilla, L. L., Pérez Vicente, C. J., Ritort, F., and Spigler, R. (2005). The kuramoto model: a simple paradigm for synchronization phenomena. *Rev. Mod. Phys*. 77, 137–185. doi: 10.1103/RevModPhys.77.137

Corti, E., Cornejo Jimenez, J. A., Niang, K. M., Robertson, J., Moselund, K. E., Gotsmann, B., et al. (2021). Coupled vo2 oscillators circuit as analog first layer filter in convolutional neural networks. *Front. Neurosci*. 15:19. doi: 10.3389/fnins.2021.628254

Corti, E., Gotsmann, B., Moselund, K., Stolichnov, I., Ionescu, A., and Karg, S. (2018). "Resistive coupled vo2 oscillators for image recognition," in *2018 IEEE International Conference on Rebooting Computing (ICRC)* (McLean, VA: IEEE), 1–7.

Corti, E., Gotsmann, B., Moselund, K., Stolichnov, I., Ionescu, A., Zhong, G., et al. (2019). "Vo2 oscillators coupling for neuromorphic computation," in *2019 Joint International EUROSOI Workshop and International Conference on Ultimate Integration on Silicon (EUROSOI-ULIS)* (Grenoble: IEEE), 1–4.

Corti, E., Khanna, A., Niang, K., Robertson, J., Moselund, K. E., Gotsmann, B., et al. (2020). Time-delay encoded image recognition in a network of resistively coupled vo on si oscillators. *IEEE Electron. Device Lett*. 41, 629–632. doi: 10.1109/LED.2020.2972006

Endo, T., and Takeyama, K. (1992). Neural network using oscillators. *Electron. Commun. Jpn*. 75, 51–59. doi: 10.1002/ecjc.4430750505

Fell, J., and Axmacher, N. (2011). The role of phase synchronization in memory processes. *Nat. Rev. Neurosci*. 12, 105–118. doi: 10.1038/nrn2979

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition, Chapter 17.2*. New York, NY; Cambdrige University Press.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. U.S.A*. 79, 2554–2558. doi: 10.1073/pnas.79.8.2554

Hoppensteadt, F. C., and Izhikevich, E. M. (2000). Pattern recognition via synchronization in phase-locked loop neural networks. *IEEE Trans. Neural Netw*. 11, 734–738. doi: 10.1109/72.846744

Izhikevich, E., and Kuramoto, Y. (2006). Weakly coupled oscillators. *Encyclopedia Math. Phys*. 448–53. doi: 10.1016/B0-12-512666-2/00106-1

Jackson, T., Pagliarini, S., and Pileggi, L. (2018). "An oscillatory neural network with programmable resistive synapses in 28 nm CMOS," in *2018 IEEE International Conference on Rebooting Computing (ICRC)* (McLean, VA: IEEE), 1–7.

Maffezzoni, P., Bahr, B., Zhang, Z., and Daniel, L. (2015a). Oscillator array models for associative memory and pattern recognition. *IEEE Trans. Circ. Syst. I Regul. Pap*. 62, 1591–1598. doi: 10.1109/TCSI.2015.2418851

Maffezzoni, P., Bahr, B., Zhang, Z., and Daniel, L. (2016). Analysis and design of boolean associative memories made of resonant oscillator arrays. *IEEE Trans. Circ. Syst. I Regul. Pap*. 63, 1964–1973. doi: 10.1109/TCSI.2016.2596300

Maffezzoni, P., Daniel, L., Shukla, N., Datta, S., and Raychowdhury, A. (2015b). Modeling and simulation of vanadium dioxide relaxation oscillators. *IEEE Trans. Circ. Syst. I Regul. Pap*. 62, 2207–2215. doi: 10.1109/TCSI.2015.2452332

Nez, J., Avedillo, M. J., Jimnez, M., Quintana, J. M., Todri-Sanial, A., Corti, E., et al. (2021). Oscillatory neural networks using vo2 based phase encoded logic. *Front. Neurosci*. 15:442. doi: 10.3389/fnins.2021.655823

Parihar, A., Shukla, N., Datta, S., and Raychowdhury, A. (2015). Synchronization of pairwise-coupled, identical, relaxation oscillators based on metal-insulator phase transition devices: a model study. *J. Appl. Phys*. 117, 054902. doi: 10.1063/1.4906783

Parihar, A., Shukla, N., Jerry, M., Datta, S., and Raychowdhury, A. (2017). Vertex coloring of graphs via phase dynamics of coupled oscillatory networks. *Sci. Rep*. 7, 911. doi: 10.1038/s41598-017-00825-1

Raychowdhury, A., Parihar, A., Smith, G. H., Narayanan, V., Csaba, G., Jerry, M., et al. (2019). Computing with networks of oscillatory dynamical systems. *Proc. IEEE* 107, 73–89. doi: 10.1109/JPROC.2018.2878854

Schmitt, O. H. (1938). A thermionic trigger. *J. Sci. Instrum*. 15, 24–26. doi: 10.1088/0950-7671/15/1/305

Sharma, A. A., Bain, J. A., and Weldon, J. A. (2015). Phase coupling and control of oxide-based oscillators for neuromorphic computing. *IEEE J. Exploratory Solid State Comput. Devices Circ*. 1, 58–66. doi: 10.1109/JXCDC.2015.2448417

Shukla, N., Parihar, A., Cotter, M., Barth, M., Li, X., Chandramoorthy, N., et al. (2014). "Pairwise coupled hybrid vanadium dioxide-mosfet (hvfet) oscillators for non-boolean associative computing," in *2014 IEEE International Electron Devices Meeting* (San Francisco, CA: IEEE), 28.7.1–28.7.4.

Shukla, N., Tsai, W.-Y., Jerry, M., Barth, M., Narayanan, V., and Datta, S. (2016). "Ultra low power coupled oscillator arrays for computer vision applications," in *2016 IEEE Symposium on VLSI Technology* (Honolulu, HI: IEEE), 1–2.

Todri-Sanial, A., Carapezzi, S., Delacour, C., Abernot, M., Gil, T., Corti, E., et al. (2021). *How Frequency Injection Locking Can Train Oscillatory Neural Networks to Compute in Phase*. Available online at: https://hal-lirmm.ccsd.cnrs.fr/lirmm-03164135 (accessed April 12, 2021).

Vassilieva, E., Pinto, G., de Barros, J., and Suppes, P. (2011). Learning pattern recognition through quasi-synchronization of phase oscillators. *IEEE Trans. Neural Netw*. 22, 84–95. doi: 10.1109/TNN.2010.2086476

Wang, H., Qi, M., and Wang, B. (2017). Ppv modeling of memristor-based oscillators and application to onn pattern recognition. *IEEE Trans. Circ. Syst. II Express Briefs* 64, 610–614. doi: 10.1109/TCSII.2016.2591961

Winfree, A. T. (1967). Biological rhythms and the behavior of populations of coupled oscillators. *J. Theor. Biol*. 16, 15–42. doi: 10.1016/0022-5193(67)90051-3

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2021 Delacour and Todri-Sanial. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

ORIGINAL RESEARCH
published: 16 November 2021
doi: 10.3389/fnins.2021.694170

A Scatter-and-Gather Spiking Convolutional Neural Network on a Reconfigurable Neuromorphic Hardware

Chenglong Zou[1,2], *Xiaoxin Cui*[1*], *Yisong Kuang*[1], *Kefei Liu*[1], *Yuan Wang*[1], *Xinan Wang*[2] *and Ru Huang*[1]

[1] *Institute of Microelectronics, Peking University, Beijing, China,* [2] *School of ECE, Peking University Shenzhen Graduate School, Shenzhen, China*

Artificial neural networks (ANNs), like convolutional neural networks (CNNs), have achieved the state-of-the-art results for many machine learning tasks. However, inference with large-scale full-precision CNNs must cause substantial energy consumption and memory occupation, which seriously hinders their deployment on mobile and embedded systems. Highly inspired from biological brain, spiking neural networks (SNNs) are emerging as new solutions because of natural superiority in brain-like learning and great energy efficiency with event-driven communication and computation. Nevertheless, training a deep SNN remains a main challenge and there is usually a big accuracy gap between ANNs and SNNs. In this paper, we introduce a hardware-friendly conversion algorithm called "scatter-and-gather" to convert quantized ANNs to lossless SNNs, where neurons are connected with ternary {−1,0,1} synaptic weights. Each spiking neuron is stateless and more like original McCulloch and Pitts model, because it fires at most one spike and need be reset at each time step. Furthermore, we develop an incremental mapping framework to demonstrate efficient network deployments on a reconfigurable neuromorphic chip. Experimental results show our spiking LeNet on MNIST and VGG-Net on CIFAR-10 datasetobtain 99.37% and 91.91% classification accuracy, respectively. Besides, the presented mapping algorithm manages network deployment on our neuromorphic chip with maximum resource efficiency and excellent flexibility. Our four-spike LeNet and VGG-Net on chip can achieve respective real-time inference speed of 0.38 ms/image, 3.24 ms/image, and an average power consumption of 0.28 mJ/image and 2.3 mJ/image at 0.9 V, 252 MHz, which is nearly two orders of magnitude more efficient than traditional GPUs.

Keywords: convolutional neural network, spiking neural network, network quantization, network conversion, neuromorphic hardware, network mapping

OPEN ACCESS

Edited by:
Angeliki Pantazi,
IBM Research-Zurich, Switzerland

Reviewed by:
Shuangming Yang,
Tianjin University, China
Timothée Masquelier,
Centre National de la Recherche Scientifique (CNRS), France

***Correspondence:**
Xiaoxin Cui
cuixx@pku.edu.cn

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *12 April 2021*
Accepted: *15 October 2021*
Published: *16 November 2021*

Citation:
Zou C, Cui X, Kuang Y, Liu K, Wang Y, Wang X and Huang R (2021) A Scatter-and-Gather Spiking Convolutional Neural Network on a Reconfigurable Neuromorphic Hardware. Front. Neurosci. 15:694170. doi: 10.3389/fnins.2021.694170

1. INTRODUCTION

Deep convolutional neural network (CNN) architectures such as VGG-Net (Simonyan and Zisserman, 2014) and ResNet (He et al., 2016) have achieved close to, even beyond human-level performance in many computer vision tasks such as image classification (Russakovsky et al., 2015) and object detection (Lin et al., 2014) in recent years. However, these large-scale models usually consist of tens of millions of parameters, and compute with massive high-precision (32/64 bits) fixed-point or floating-point multiply-accumulation (MAC) operations. Although network training can be implemented on a cloud server equipped with powerful CPUs or GPUs using backpropagation algorithm (Rumelhart et al., 1986), inference at edge still inevitably requires vast power and memory budget. Lots of works presented various compression (Deng et al., 2020) and quantization methods (Hubara et al., 2016) of neural network or concentrated on less memory access and pipeline optimizing in custom CNN accelerators (Lecun, 2019; Chen et al., 2020), which greatly improved computation efficiency and reduced power consumption.

Considering another kind of emerging approach to incorporate biological plausibility of brain-inspired models and efficient neuromorphic hardware primitives, spiking neural networks (SNNs) (Grning and Bohte, 2014) attract more attention. SNNs inherently communicate and compute with one-bit spike signals and low-precision synapses toward an event-driven information processing paradigm (consuming energy only when necessary) (Sheik et al., 2013; Deng et al., 2020). It has been proved that SNNs are very suitable to be implemented on large-scale distributed neuromorphic chip with impressive energy efficiency (Cassidy et al., 2013; Schuman et al., 2017). For example, a single TrueNorth chip (Akopyan et al., 2015) supports real-time running of 1 million neurons and 256 million synapses with only 70 mW power consumption. Tianjic chip (Deng et al., 2020) is composed of 156 functional neuromorphic core, and achieve several orders of magnitude of energy efficiency compared with common platforms like CPUs or GPUs.

However, training a high-accuracy SNN remains a main challenge due to discrete spike representation and non-differentiable threshold function (Tavanaei et al., 2018). To date, various methods have been applied to construct SNNs with comparable accuracy to conventional CNNs. Some works adopt bioinspired learning rules like unsupervised spike-timing dependent plasticity (STDP) (Falez et al., 2019; Lobov et al., 2020) for feature extraction. However, these layer-by-layer training algorithms usually perform less efficiently in deep architectures. For supervised learning like SpikeProp (Bohtea et al., 2002) and Tempotron (Gutig and Sompolinsky, 2006), they also fail to deal with practical tasks like CIFAR-10 (Krizhevsky and Hinton, 2009) classification. Recent works (Lee et al., 2016, 2020; Wu et al., 2018; Wei et al., 2020; Yang et al., 2021a) use different pseudo-derivative methods (also called surrogate gradient) to define the derivative of the threshold-triggered working mechanism. Thus, the SNNs could be optimized with gradient descent algorithms as artificial neural networks (ANNs) and achieve good accuracies with fast response speed, but a unified and effective surrogate function is the key problem for these methods.

ANN-to-SNN conversion is another popular solution, which tries to match firing rates of spiking neurons and analog activations of ANNs. Esser et al. (2016) presented a simple BNN-to-SNN conversion algorithm, where spike signals are coded within only one time step, so each neuron will fire at most once. Binary SNNs can achieve a great model compression rate with the least resource and power budgets and fastest inference speed with an acceptable loss of accuracy on MNIST (Lecun and Bottou, 1998) and CIFAR-10 (Krizhevsky and Hinton, 2009) datasets. A more common approach is to map the parameters of a ReLU-based ANN to that of an equivalent SNN. Studies (Bodo et al., 2017; Xu et al., 2017) have found that SNNs can be converted from trained high-accuracy CNNs efficiently by the means of data-based threshold or weight normalization. However, the network performances depend on empirical statistics of average firing rate, and require dozens even hundreds of time steps to get a stable accuracy. This may give a large energy and latency budget for hardware implementation. Besides, the final accuracy is still declining when compared with its ANN counterpart due to accumulated errors of spike approximation in higher layers (Bodo et al., 2017; Rueckauer and Liu, 2018; Yousefzadeh et al., 2019).

This work aims to overcome the aforementioned drawbacks in ANN-to-SNN conversion process and hardware implementation, i.e., to present a more accurate, general, and hardware-friendly conversion method, which is compatible with contemporary neuromorphic hardware. For this purpose, we first introduce an adjustable quantized algorithm in ANN training to minimize the spike approximation errors, which are commonly existed in ANN-to-SNN conversion and propose a scatter-and-gather conversion mechanism for SNNs. This work is based on our previous algorithm (Zou et al., 2020) and hardware (Kuang et al., 2021), and we extend it by (a) testing its robustness on input noise and larger dataset (CIFAR-100), (b) developing a incremental mapping framework to carry out an efficient network deployment on a typical crossbar-based neuromorphic chip, (c) detailed power and speed analyses are given to show its excellent application potential. All together, the main contributions of this article are summarized as follows:

1. Compared with existing ANN-to-SNN conversion methods, the proposed conversion algorithm with quantization constraint can be jointly optimized at training stage, which greatly eliminate the common spike approximation errors. The final accuracy can benefit from higher quantization level and upper bound. Our presented spiking LeNet and VGG-Net achieve great classification accuracies and source code can be available online[1];
2. An incremental mapping algorithm is presented to optimize network topology placement on a reconfigurable neuromorphic chip with maximum resource efficiency and sufficient flexibility. Besides, three novel evaluation criteria are proposed to analyze resource utilization on general crossbar-based neuromorphic hardware;
3. Experimental results show that our four-spike LeNet and VGG-Net can achieve about 99.37% and 91.91% test accuracy on MNIST and CIFAR-10 dataset, respectively, while our system can obtain nearly 0.38 and 3.24 ms/image real-time inference speed, and an average power consumption of 0.28 and 2.3 mJ/image accordingly. It should be noted that the presented spiking models can be also mapped onto many

[1]https://github.com/edwardzcl/Spatio_temporal_SNNs

large-scale neuromorphic platforms like TrueNorth (Akopyan et al., 2015) and BiCoSS (Yang et al., 2021b) built with integrate-and-fire (IF) neurons.

The rest of this article is organized as follows. section 2 introduces the principle of proposed median quantization and scatter-and-gather conversion. In section 3, we introduce a reconfigurable neuromorphic chip and present an incremental mapping workflow to complete model deployment. Experimental results including classification accuracy, resource utilization, and inference speed are presented in section 4. Finally, section 5 concludes this paper.

2. NETWORK CONVERSION

2.1. Background

Conventional CNNs are mainly composed of an alternate cascade of convolutional layer, ReLU (Glorot et al., 2011) activation function, and pooling layer. For improving final accuracy and learning efficiency in deep networks, there is usually an additional batch normalization layer located between the convolution layer and ReLU activation function, which achieves an output distribution of zero-mean and unit variance. Used as a standard module in most state of art CNNs, a general convolutional layer can be formulated as Equations (1)–(3):

$$Conv \qquad s = \sum_{i,j,k} w_{i,j,k} * x_{i,j,k} \tag{1}$$

$$BN \qquad r = \frac{s-\mu}{\sigma+\varepsilon} + \beta \tag{2}$$

$$ReLU \qquad y = \max(0, r) \tag{3}$$

where i, j, k indicate the width, height, and channel dimension of a convolutional kernel, s is inner product result of weight w and input x, μ and σ are the mean and standard deviation of s, β is the bias term, $\varepsilon = 10^{-6}$ for numerical stability. Note, we omit the scaling term in all equations involved with BN for the convenience of description. Because parameter-free pooling layer is used for simple down-sampling, most of the memory and power budgets come from intensive high-precision (32/64 bits) float-point or fixed-point MAC operations in convolutional layers.

For a spiking neural network built with IF neurons (Abbott, 1999), the membrane potential V of each neuron will change due to the spike integration x from other neurons i at every time step t as Equation 4, where w represents the synaptic strength. A neuron will emit a spike at some time when its membrane potential is greater than a pre-defined threshold in Equation 5. This discrete spiking dynamic behavior is quite different from ANNs, in which the activation function is continuous.

$$V(t+1) = V(t) + \sum_i x_i(t) * w_i \tag{4}$$

$$Spike = \begin{cases} 0 & \text{if} \quad V(t+1) < \theta \\ 1 & \text{if} \quad V(t+1) \geq \theta \end{cases} \tag{5}$$

To take advantage of end-to-end training process in deep learning, we are looking forward to an effective method which can convert a quantized and high-accuracy CNN to a spike-based SNN with nearly lossless accuracy. By comparison through the forward process between contemporary CNNs and SNNs, we summarize several key differences as follows:

1. SNN has no individual normalization layer and pooling layer but particular threshold terms θ.
2. SNN usually communicates with timed spike trains of binary value {0,1}, instead of continuous values.
3. If we try an ANN-to-SNN conversion method, how to ensure that firing rate of each spiking neuron is absolutely proportional to corresponding activation output of an ANN neuron without approximation errors.

In this work, we use convolutions with stride of 2 to replace pooling for structure unity, which was proved to be feasible (Springenberg et al., 2014; Esser et al., 2016). Therefore, the main problem is how to deal with incompatible batch normalization layer and continuous activation function, which are essential for a deep ANN training and final accuracy performance.

2.2. Training With Median Quantization

Previous works such as Lee et al. (2016) and Bodo et al. (2017) intend to maintain a balance between the synaptic weights and firing thresholds using a robust normalization method based on maximum value of weights or activations in each layer. However, there are always big spiking approximation errors accumulated in higher layer, which explains why it takes a longer time (dozens or hundreds of time steps) to achieve high correlations of ANN activations. Moreover, the final accuracy and real-time performance of spiking models will seriously suffer from this effect. In contrast, we choose to take these common approximation errors into consideration at model training stage with a median quantization constraint formulated as in Equation (6):

$$Quant(r) = clip(\frac{round(r * 2^k)}{2^k}, 0, B) \tag{6}$$

where r is the batch normalization output (Equation 2), and the quantization level k and upper bound B are two hyper-parameters, which determine the spike encoding precision. For example, when the quantization level $k = 0$ and upper bound $B = 4$, this quantized ReLU (**Figure 1**) can be formulated as follows:

$$y = \begin{cases} 2 & if \quad r \geq 1.75 \\ 1.5 & if \quad 1.25 \leq r < 1.75 \\ 1 & if \quad 0.75 \leq r < 1.25 \\ 0.5 & if \quad 0.25 \leq r < 0.75 \\ 0 & if \quad r < 0.25 \end{cases} \tag{7}$$

where y is the output of quantized ReLU. Then, we can further integrate batch normalization (Equation 2) into quantization

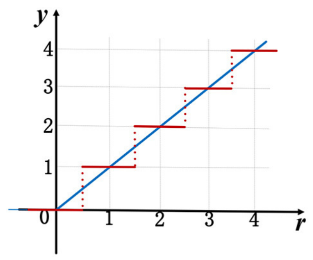


FIGURE 1 | A median quantization with $k = 0$ and $B = 4$ for ReLU. The blue line shows original ReLU function and the red line for the quantized ReLU.

(Equation 7) and modify it as:

$$y' = \begin{cases} 4 \; if \; s' \geq (1.75 - \beta)(\sigma' + \varepsilon) + \mu' \\ 3 \; if \; (1.25 - \beta)(\sigma' + \varepsilon) + \mu' \leq s' < (1.75 - \beta)(\sigma' + \varepsilon) + \mu' \\ 2 \; if \; (0.75 - \beta)(\sigma' + \varepsilon) + \mu' \leq s' < (1.25 - \beta)(\sigma' + \varepsilon) + \mu' \\ 1 \; if \; (0.25 - \beta)(\sigma' + \varepsilon) + \mu' \leq s' < (0.75 - \beta)(\sigma' + \varepsilon) + \mu' \\ 0 \; if \; s' < (0.25 - \beta)(\sigma' + \varepsilon) + \mu' \end{cases} \tag{8}$$

$$\mu' = 2 * \mu, \quad \sigma' = 2 * \sigma \tag{9}$$

where s' is the new inner product, together with mean μ' and standard deviation σ' need be scaled twice of original values in Equation (2). Intuitively, the amplitude of quantized ReLU exactly matches spike counts of SNNs. In this example, there are at most four spikes generated. It should be noted that both of the quantization level and upper bound are adjustable as a trade-off between final accuracy and firing rate. Higher quantization level or upper bound may result in a better classification performance but will bring more spikes, which will be discussed in section 4. To enable gradient-based training, we use a straight-through estimator (STE) previously introduced in Bengio et al. (2013), which replaces the piecewise ReLU (red line in **Figure 1**) with its continuous version (blue line in **Figure 1**) in backward pass process. Therefore, the above conversion coefficients and accuracy performances can be iteratively optimized with our proposed quantization constraints during training. More specially, the batch normalization operation (Equation 2), which is incompatible with SNNs, can be merged into ReLU activation function without any computing cost[2].

2.3. Conversion With Scatter-and-Gather

Based on quantized ANNs presented above, we develop a rate-based conversion method called scatter-and-gather for SNNs. For instance of network with quantization level $k = 1$ and upper bound $B = 2$, we need configure four SNN neurons with different thresholds described as in Equation (10) to match the activation output of one ANN neuron, and each spiking neuron will fire at most once within only one time step,

$$\begin{cases} V(t+1) = V(t) + \sum_i x_i(t) * w_i \\ \theta_1 = \mu' + (0.25 - \beta) * (\sigma' + \varepsilon) \\ \theta_2 = \mu' + (0.75 - \beta) * (\sigma' + \varepsilon) \\ \theta_3 = \mu' + (1.25 - \beta) * (\sigma' + \varepsilon) \\ \theta_4 = \mu' + (1.75 - \beta) * (\sigma' + \varepsilon) \end{cases} \tag{10}$$

where V is the shared membrane potential for four IF neurons, x is the incoming spike, w is the strength of corresponding synapse which is same as original ANN counterpart, θ is the threshold, and other variables are the batch normalization terms in Equation (8). This converted neuron model is really similar to the McCulloch and Pitts model (Hayman, 1999), where simple threshold gates are enabled and there is no temporal information integration. The only difference is that threshold choices of each neuron may be different. This scatter-and-gather mechanism is described in **Figure 2**. Four SNN neurons work synchronously, receive the same spike inputs, and share the same synaptic strengths, but fire with respective threshold (θ_1-θ_4). Hence, the total time step for one sample simulation will be always 1 and membrane potential will be reset after firing and prepare for next new sample. It should be noted that the proposed scatter-and-gather conversion is really different from AMOS algorithm (Stckl and Maass, 2019). AMOS needs to use different transmitting delays between intra- and inter-layer neurons to maintain information synchronization within multiple time steps. Besides, their conversion coefficients and thresholds are determined by fitting activation gates after ANN training, but our parameter determination method described in Equations (8)–(10) guarantees a lossless conversion from the corresponding quantized ANNs. Compared with Esser et al. (2016), our method can be seen as a generalization from a single spike to multi-spike conversion, to some extent.

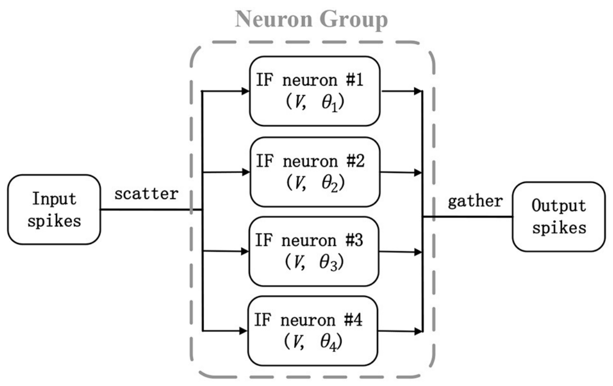

Neuron Group
IF neuron #1 (V, θ_1)
IF neuron #2 (V, θ_2)
IF neuron #3 (V, θ_3)
IF neuron #4 (V, θ_4)
Input spikes
scatter
gather
Output spikes

FIGURE 2 | A scatter-and-gather mechanism in artificial neural network (ANN)-to-spiking neural network (SNN) conversion: Four integrate-and-fire (IF) neurons (neuron group) work synchronously and replace an equivalent ANN neuron.

[2]In this paper, we focus on neural networks with batch normalization, but our quantization method can also support other architectures without that.

3. NETWORK MAPPING

In this section, we briefly describe the structure and function of a reconfigurable neuromorphic chip (Kuang et al., 2021), and then present an incremental mapping workflow to demonstrate efficient hardware deployments for our converted SNNs.

3.1. Neuromorphic Processor

This chip (Kuang et al., 2021) is designed as a neuro-synaptic processing core, which consists of 1,152 transmission axons, 1,024 basic LIF spiking neurons, and an 1,152 $*$ 1,024 synaptic crossbar (see **Figure 3**). There is a multicasting router working with an address event representation (AER) protocol (Boahen, 2000) in each chip. The AER router is responsible for receiving and sending signals, which includes general spike packets and programming and test packets. With four AER interfaces in the east, west, north, and south directions, multiple chips can be formed as an 8 $*$ 8 mesh network to support a larger-scale system.

Each basic spiking neuron has an individual programmable connectivity strength shared by connected 1,152 synapses, each of which can be additionally configured as on or off state. We can employ multiple basic neurons with different connectivity strengths, to make up a complete neuron and achieve a multi-bit (1, 2, 4, 8) weight representation. For example, for a combination of four basic neurons with respective connectivity strength $\{w_1 = 1,\ w_2 = 2,\ w_3 = 4,\ w_4 = -8\}$, we can achieve a 4-bit representation range of -8 to 7. Moreover, this neuro-synaptic crossbar supports a spatial axon extension at most 64 (1, 2, 4, 8, 16, 32, 64) times during a complete computing period, to take in a larger feature map input (fan-in) with the cost of decreasing the number of output neuron ports (fan-out) on chip. As illustrated in **Figure 4**, a spatial neuron is comprised of two complete neurons to support a double (1,152 $*$ 2) fan-in of feature receptive field and the output neuron ports halve. For an extreme instance, we can support a largest convolutional kernel of 3 $*$ 3 $*$ 2,048 and output only one feature point. All in all, these two reconfigurability functions improve the precision of synapses and enhance the ability for processing larger receptive field of convolution and pooling.

In the working mode, the dynamic LIF neuron dynamics behavior is performed and membrane potential is updated. It should be noticed that synaptic nodes which are not triggered by spike events will have no computation activity. Spike events in typical neuromorphic systems are generally discrete and sparse, which can be efficiently delivered by the AER router and multicast among multiple cores. For some larger-scale neural networks exceeding on-chip memory, two alternate SRAMs will work alternately like a ping-pong buffer to enhance the computing throughput. In other words, when memory controller is reading the current weight parameters and neuron states from one of them, a direct-memory-access (DMA) controller will take new programming data (weight parameter and scheduling information) from off-chip memory and writes them to the other one to update the synapse connectivity and neuron states. With the ability of ping-pong reuse, our chip should have potential to implement large-scale network architectures like VGG-Net (Simonyan and Zisserman, 2014) on one core, compared with many other large-scale neuromorphic chips (Akopyan et al., 2015; Yang et al., 2021b) in general.

3.2. Mapping Strategy

However, almost all contemporary neuromorphic hardwares, designed with 2D crossbar-based structure, have typical block-wise constraints for neuron connectivity (Bouvier et al., 2019). For a standard 2-D crossbar unit with finite inputs and outputs (256 $*$ 256 for TrueNorth), it is impossible to process a complete convolutional layer individually. Building with 256 $*$ 256 synaptic computing core, TrueNorth has to use group convolution (Esser et al., 2016) to cut a large convolutional layer into many slices. For the sake of description, we adopt a series of definitions in **Table 1** for different notations. Due to local speciality of convolution operation, a common approach is to partition 3-D input feature maps into a number of $m * n$ patches seeing (**Figure 5**) to ensure that the size of each patch is less than the number of input axons. In this case, each patch is a spatial topographic location involving all of input feature map channels in Equation (12). Adjacent patches have a specific overlapping region that depends on the kernel size and stride of convolution or pooling. In contrast, our chip could extend processing receptive field for a larger input patch with larger width or height by reusing 1,152 input axons for f times in Equation (11).

$$w_l * h_l * d_l \le \frac{1,152 * f}{k_{rep}},\ w_{l+1} * h_{l+1} * d_{l+1} \le \frac{1,024}{f * k_{wei} * k_{rep}} \quad (11)$$

$$d_l = D_l, \qquad d_{l+1} \le D_{l+1} \quad (12)$$

$$w_{l+1} = \frac{w_l - c_{l+1}}{s_{l+1}} + 1, \quad h_{l+1} = \frac{h_l - c_{l+1}}{s_{l+1}} + 1 \quad (13)$$

Accordingly, the size of resulted output patch can be calculated from the size of input patch as in Equation (13). As introduced in previous section, the output size may be greater than the number of fewer output neurons, because of higher weight precisions or spatial axon extension. For example, if we want to implement a 1-bit convolution ($k_{rep} = 1$, $k_{wei} = 2$) for an input feature maps of 4 $*$ 4 $*$ 128 ($W_l * H_l * D_l$) with a filter kernel of 2 $*$ 2 $*$ 128 $*$ 256 ($c_{l+1} * c_{l+1} * D_l * D_{l+1}$) and stride of 2 ($s_{l+1} = 2$), the size of output feature maps can be calculated as 2 $*$ 2 $*$ 256 ($D_{l+1} * H_{l+1} * D_{l+1}$) according to Equation (13). Then, there may be two mapping options: 1: four identical input patches of 4 $*$ 4 $*$ 128 ($w_l * h_l * d_l$) distributed on four computing cores, respectively; each of which contributes to an output patch of 2 $*$ 2 $*$ 64 ($w_{l+1} * w_{l+1} * w_{l+1}$); 2: two complementary input patches of 4 $*$ 2 $*$ 128 distributed on two computing cores, respectively; each of which contributes to an output patch of 2 $*$ 1 $*$ 256. Detailed mapping results are shown in **Table 2**.

Here, we define three practical evaluation criteria (Equations 14–16) to thoroughly figure out how many effective axons, neurons, and synapse connections are occupied in a standard neuro-synaptic crossbar. Higher utilization density means a

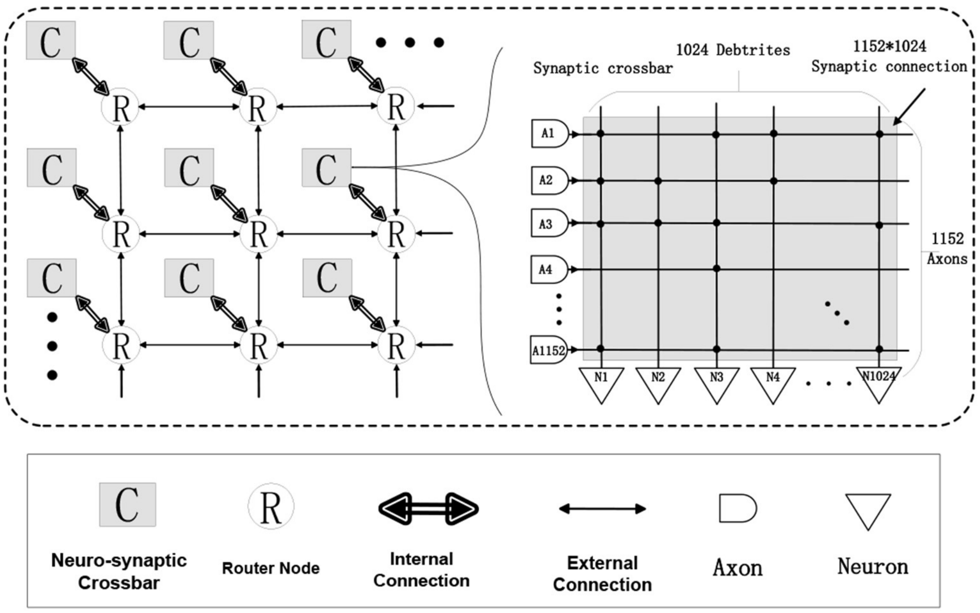


FIGURE 3 | A structural view of our neuromorphic chip: 8 * 8 chips form a multi-chip array, each chip consists of 1,024 LIF neurons, 1,152 axons and a connected synaptic crossbar of 1,152 * 1,024 size.

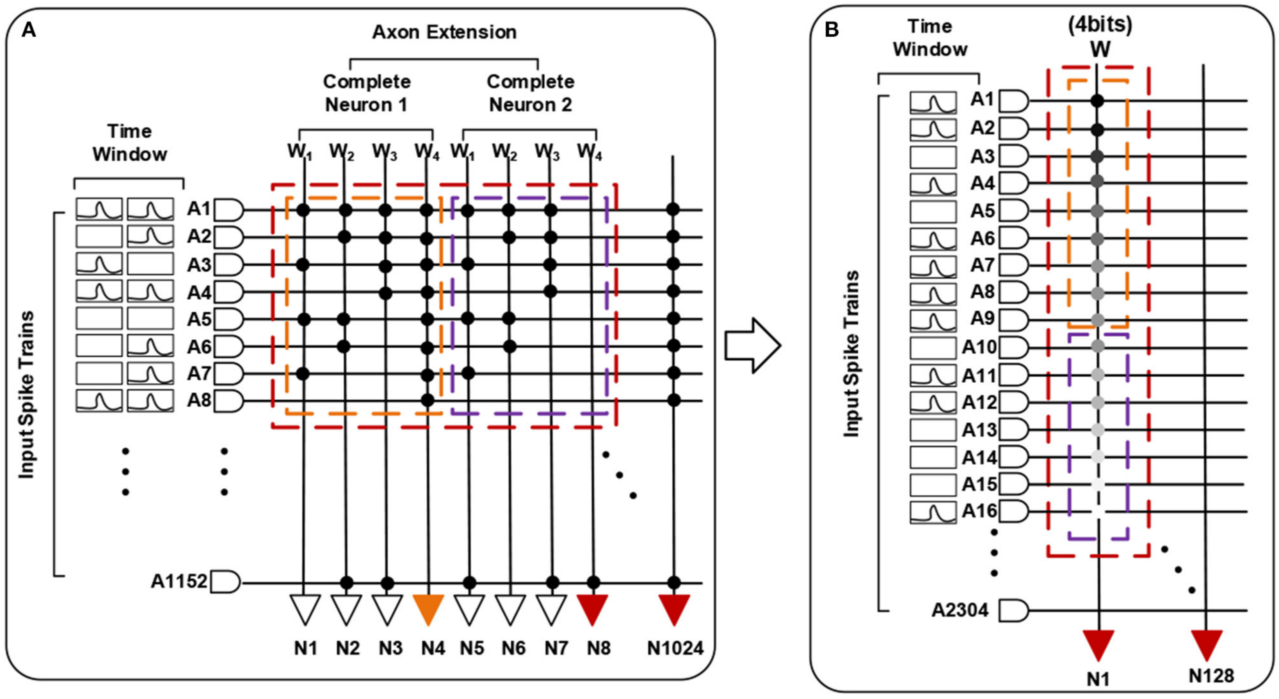


FIGURE 4 | A functional view of our neuromorphic chip. **(A)** describes a spatial neuron with axon extension $f = 2$ (two complete neurons) and a combination for 4-bit weights. **(B)** is the equivalent one.

more compact mapping with less resource consumption and reduces redundancy.

$$Density_{neuron} = \frac{k_{rep} * (w_{l+1} * h_{l+1} * d_{l+1}) * f * k_{wei}}{1,024} \tag{14}$$

$$Density_{axon} = \frac{k_{rep} * (w_l * h_l * d_l)}{1,152 * f} \tag{15}$$

$$Density_{synapse} = \frac{k_{rep} * (w_{l+1} * h_{l+1} * d_{l+1}) * (c_{l+1}^2 * d_l * k_{rep} * k_{wei})}{1,152 * 1,024} \tag{16}$$

We summarize the three criteria of two plans in **Table 3**. It can be found that both of the $Density_{neuron}$ and $Density_{axon}$ are the same, but $Density_{synapse}$ of the No. 2 is twice as high as that of the No.1. From a hardware perspective, worse utilization of crossbar will lead to a more resource budget and multicast communication workload for fixed sized feature maps. Hence, there is a tradeoff between the size of input patch and output patch. Larger width or height of patches does not mean better resource efficiency on a specific chip. For each patch on a neuro-synaptic crossbar, d_l,c_{l+1},k_{rep},k_{wei} are all constant, we need to selectively increase w_l,h_l,f or d_{l+1} for a maximum utilization of axon, neuron, and synapse. Learning from the example above, a progressive strategy is to give the priority to increase output channel d_{l+1} and do not increase w_l and h_l until d_{l+1} is up to D_{l+1}, while there are still available neurons for axon extension. This priority leads to the minimum overlapping chance of sliding windows along width and height and guarantees all of hardware modules are working with a high resource efficiency.

After the primary size of each patch is determined, another problem is how to choose the shape. We can take an intuitive understanding in **Figure 6**. It shows output patches with the same size ($2 * 2$ and $1 * 4$) may have multiple options to be generated from different sized input patches. Similarly, input patches with the same size but different shapes will generate different number

TABLE 1 | Summary of main notations.

Notation	Description
W	Width of input/output feature map
H	Height of input/output feature map
D	Depth of input/output feature map
w	Width of input/output patch
h	Height of input/output patch
d	Depth of input/output patch
l	*l*th layer of convolution/pooling
m	Number of horizontal patches
n	Number of vertical patches
p	Number of depth-oriented patches
c	Kernel size of convolution/pooling
s	Stride of convolution/pooling
k_{rep}[a]	How many SNN neurons replace an ANN neuron
k_{wei}[b]	Bit-width of weight parameter
f	Axon extension
core	Neuro-synaptic crossbar
patch	Partial feature map
time step	Computing time for all neurons on a core

[a] *For spatial conversion, an ANN neuron will be replaced by multiple SNN neurons, i.e.,* $k_{rep} = B*2^k$.
[b] k_{wei} *represents the quantization bit-width for weight parameters of convolution kernels. In this article, we fixed* $k_{wei} = 2$ *for a simple ternary quantization of {–1, 0, +1}.*

TABLE 2 | Mapping results of two plans.

Plan	w_l	h_l	d_l	w_{l+1}	h_{l+1}	d_{l+1}	f	core
No. 1	4	4	128	2	2	64	2	4
No. 2	4	2	128	2	1	256	1	2

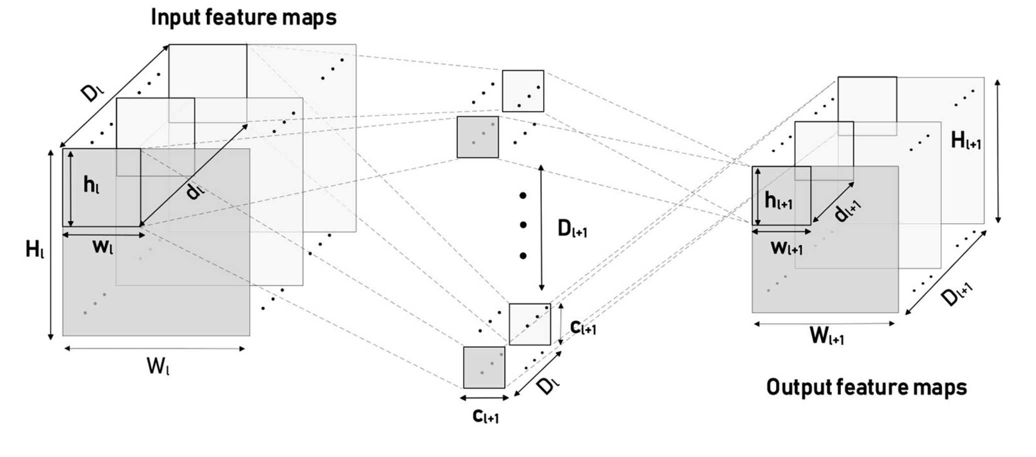


FIGURE 5 | Input and output patches on the corresponding feature maps.

of sliding windows, which means the size of output patches are not equal. We can consider the mean-value inequality for w_l,h_l seeing (Equation 17). The equality become valid only when $w_l = h_l$. The size of input and output patch is proportional to the product result on the left of Equations (17) and (18). If $w_l * h_l$ is a constant, maximizing $w_{l+1} * w_{l+1}$ must require $w_l = h_l$. This means a square patch is more compact than rectangular one and should be the first choice.

$$w_l * h_l \leq \frac{(w_l + h_l)^2}{4} \tag{17}$$

$$\begin{aligned} w_{l+1} * h_{l+1} &= (\frac{w_l - c_{l+1}}{s_{l+1}} + 1) * (\frac{h_l - c_{l+1}}{s_{l+1}} + 1) \\ &= \frac{w_l + h + (w_l + h_l) * (s_{l+1} - c_{l+1}) + (s_{l+1} - c_{l+1})^2}{s_{l+1}^2} \end{aligned} \tag{18}$$

For an overall consideration of patch size and shape, a channel-major and square-major mapping algorithm is described in **Algorithms 1**, **2** and **Figure 7**, respectively. We integrate above two priority principles into a progressive grid search strategy to obtain an optimal choices for undetermined parameters, i.e., a list of w_l, w_l, w_l, w_{l+1}, h_{l+1}, d_{l+1}, and f. In **Algorithm 1**, we first initialize each parameter with minimum, and then **Algorithm 1** would gradually increase the number of patch channels (d_{l+1}) but fix the patch width and height (w_{l+1}, h_{l+1}) until d_{l+1} equals D_{l+1} or the output neurons on a core are used up. Last but not least, if there are still remaining resources unused after **Algorithm 1** procedure, **Algorithm 2** will perform a step-by-step multi-path grid search process for potential and feasible mapping choices and output the maximum one for target crossbar-based neuromorphic chip.

TABLE 3 | Resource efficiency of two plans.

Plan	$Density_{neuron}$	$Density_{axon}$	$Density_{synapse}$
No. 1	100%	89%	22%
No. 2	100%	89%	44%

3.3. Spatial Mapping

As mentioned above, in this work, we mainly use a simple ternary-valued {−1,0,+1} weight quantization. Therefore, for an

Algorithm 1 Channel-major search.

This is the first procedure to generate an primary input and output patch size including (w_l, h_l, d_l, w_{l+1}, h_{l+1}, d_{l+1}, f). The output channel d_{l+1} will be less than or equal to D_{l+1}.

```
Require: quantization precisions ($k_{rep}$, $k_{wei}$), kernel size ($c_{l+1}$)
    and the number of output feature map channels ($D_{l+1}$)
Ensure: primary input and output patch size including ($w_l$, $h_l$,
    $d_l$, $w_{l+1}$, $h_{l+1}$, $d_{l+1}$, $f$)
 1: Initialize: $w_l = c_{l+1}$, $h_l = c_{l+1}$, $d_l = D_l$, $f = 1$;
 2: for $d_{l+1} = 1$ to $D_{l+1}$ do
 3:     if meet the left of constraint (Equation 11) then
 4:         ;
 5:     else
 6:         $f = f * 2$, jump to line 3;
 7:     end if
 8:     if not meet the right of constraint (Equation 11) then
 9:         output ($w_l$, $h_l$, $d_l$, $w_{l+1}$, $h_{l+1}$, $d_{l+1} - 1$, $f$);
10:     else
11:         ;
12:     end if
13:     if $d_{l+1} = D_{l+1}$ then
14:         output ($w_l$, $h_l$, $d_l$, $w_{l+1}$, $h_{l+1}$, $d_{l+1}$, $f$);
15:     end if
16: end for
```

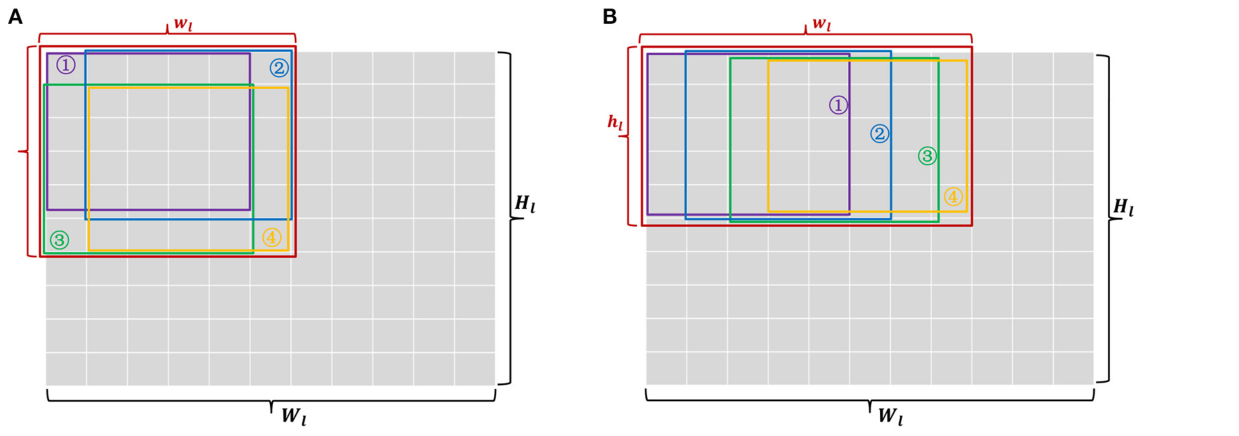


FIGURE 6 | Two kinds of input patches **(A,B)** which generate the same sized output patches. But the shape is square in **(A)** and rectangular in **(B)**. Each colored box denotes a 5 * 5 receptive field of convolution, except the red box that denotes a total input patch.

Algorithm 2 Square-major search.

This is the second procedure that should be executed after **Algorithm 1** and when $d_{l+1} = D_{l+1}$, and obtain a final input and output patch size including $(w_l, h_l, d_l, w_{l+1}, h_{l+1}, d_{l+1}, f)$.

Require: quantization precisions (k_{rep}, k_{wei}), kernel size (c_{l+1}) and the width and height of input feature map (W_l, H_l)

Ensure: final input and output patch size including $(w_l, h_l, d_l, w_{l+1}, h_{l+1}, d_{l+1}, f)$

```
1: for $w_l = c_{l+1}$ to $W_l$ or $h_l = c_{l+1}$ to $H_l$ do
2:     take a red step in Figure 7;
3:     if meet the left of constraint (Equation 11) then
4:         if meet the right of constraint (Equation 11) then
5:             ;
6:         else
7:             take a bluestep in Figure 7;
8:             if meet the right of constraint (Equation 11) then
9:                 mark $(w_l, h_l, d_l, w_{l+1}, h_{l+1}, d_{l+1}, f)$
10:            else
11:                take a green step in Figure 7;
12:                if meet the right of constraint (Equation 11) then
13:                    mark $(w_l, h_l, d_l, w_{l+1}, h_{l+1}, d_{l+1}, f)$
14:                else
15:                    take a blue step in Figure 7;
16:                    if meet the right of constraint (Equation 11) then
17:                        mark $(w_l, h_l, d_l, w_{l+1}, h_{l+1}, d_{l+1}, f)$
18:                    else
19:                        jump to line 11;
20:                    end if
21:                end if
22:            end if
23:        end if
24:        compare all marks and output the maximum, exit;
25:    else
26:        if $f == 64$ then
27:            output $(w_l, h_l, d_l, w_{l+1}, h_{l+1}, d_{l+1}, f)$;
28:        else
29:            $f = f * 2$, jump to line 3;
30:        end if
31:    end if
32: end for
```

SNN with different spike encoding precisions (k and B), we can configure k_{rep} and k_{wei} as follows:

$$k_{rep} = B * 2^k, k_{wei} = 2 \tag{19}$$

where k_{rep} means an ANN neuron is replaced by $B * 2^k$ SNN spatial neurons, each of which has the same synaptic connections and spike inputs but fire with different thresholds as discussed in section 2; k_{wei} means each complete neuron is composed of two basic spiking neurons with respective weight $\{w_1 = -1, w_2 = 1\}$ as in **Figure 8**. The number (f) of complete neurons contained in a spatial neuron is determined by the size of feature

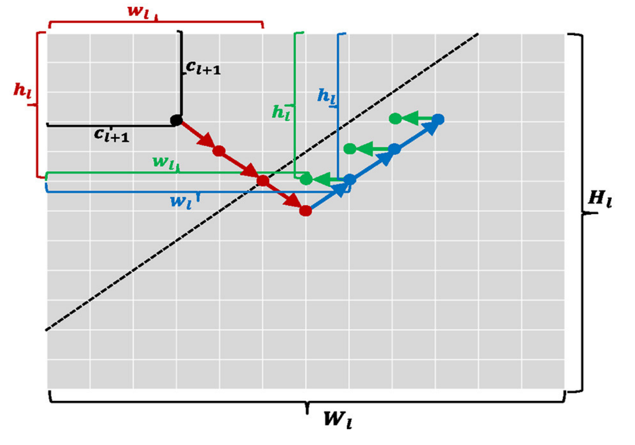


FIGURE 7 | A graphical description for **Algorithms 1** and **2**. A step-by-step grid search is performed and arrows with different colors denote different search directions. Each of colored dots is a candidate item of parameter configurations.

maps and k_{rep} according to **Algorithms 1**, **2**. For a complete convolutional or pooling layer, if we keep each patch equal, the numbers of horizontal, vertical and depth-oriented patches would be calculated as Equations (20)–(22), respectively.

$$W_l = w_l * m_l - (m_l - 1) * (c_{l+1} - s_{l+1}) \tag{20}$$

$$H_l = h_l * n_l - (n_l - 1) * (c_{l+1} - s_{l+1}) \tag{21}$$

$$D_{l+1} = p_l * d_{l+1} \tag{22}$$

Finally, we can distribute a total $m_l * n_l * p_l$ convolution patches onto our multi-chip $(8 * 8)$ system on schedule, together with ping-pong working mode. If resources are sufficient, fully unfolded mapping can achieve highest throughput and power efficiency, because the scatter-and-gather conversion ensures all spike signals are accessible at one computing time step for a layer. More importantly, expensive off-chip memory access budgets can be saved.

4. EXPERIMENTS

In this section, we first conduct an ablation study about quantization level k and upper bound B to evaluate the effectiveness of our proposed conversion and quantization algorithm on MNIST and CIFAR-10/100 dataset using LeNet and VGG-Net architecture, respectively. Then, we carry out practical mapping of above spiking networks onto our neuromorphic system and provide corresponding speed and power analysis results.

4.1. Benchmark Applications

1. *MNIST dataset*
   The MNIST dataset (Lecun and Bottou, 1998) of handwritten digit has been widely applied in image classification field,

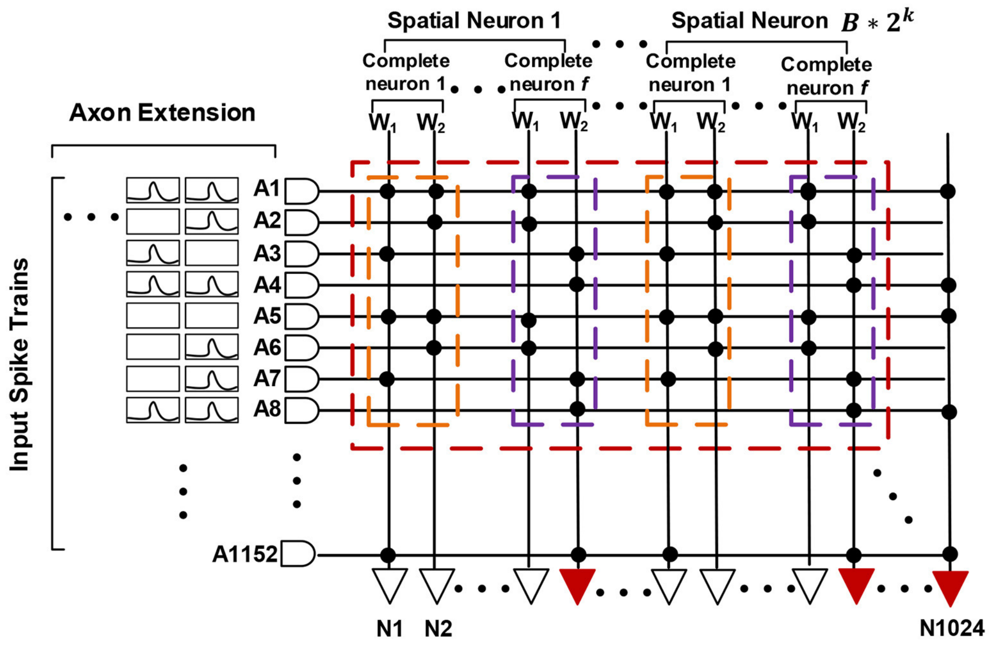


FIGURE 8 | Spatial mapping for spiking neural networks (SNNs) with scatter-and-gather conversion. An artificial neural network (ANN) neuron is replaced by $B * 2^k$ spatial SNN neurons, and each spatial neuron comprises *f* complete neurons with ternary-valued weights.

which was collected from postal codes, including a training set of 60,000 examples, and a test set of 10,000 examples. Each example is an individual 28 ∗ 28 pixel grayscale image labeled 0–9. Pixel values are integer (0–255), where 0 means background (white) and 255 means foreground (black). We adopt a classical LeNet (Lecun and Bottou, 1998) architecture (16*C*5-16*C*5-2*P*2-32*C*5-2*P*2-256*FC*-10*FC*)[3] for this task.

2. *CIFAR-10/100 dataset*
The CIFAR-10 dataset (Krizhevsky and Hinton, 2009) consists of 60,000 32 ∗ 32 pixel color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The CIFAR-100 dataset[4] is just like the CIFAR-10 but more challenging. It has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. A VGG-Net (Simonyan and Zisserman, 2014) variant with 13 layers (64*C*3-64*C*3-64*C*3-2*P*2-128*C*3-128*C*3-2*P*2-256*C*3-256*C*3-2P2 -512*C*3-512*C*3-10*FC*) is designed for these two image classification tasks. No data augmentation is used other than standard random image flipping and cropping for training. Test evaluation is based solely on central 24 ∗ 24 crop from test set (for both CIFAR-10 and CIFAR-100).

In our experiments, we use a ternary-valued {-1,0,1} weight quantization as in Li and Liu (2016), not full precision (16 or 32 bits) like many others (Lee et al., 2016, 2020; Bodo et al., 2017; Mostafa et al., 2017; Rueckauer and Liu, 2018; Wu et al., 2018; Yousefzadeh et al., 2019), to facilitate hardware deployment, because we find the weight quantization with more bit-width contributes very little to final accuracy, which is consistent with (Rastegari et al., 2016; Zhou et al., 2016). All convolutional networks are trained using standard ADAM rule (Kingma and Ba, 2014) with an initial learning rate set to 0.001 and 10 times decayed per 200 epochs, based on TensorLayer (Dong et al., 2017), a customized deep learning library. We did not use any weight or spike penalty or dropout (Srivastava et al., 2014) during training.

4.2. Quantization Precision

Here, we conduct a series of ablation experiments on two hyper-parameters, i.e., quantization level *k* and upper bound *B*, both of which jointly determine how many spikes each neuron will fire at most and relate to overall resource, latency, and power consumption on hardware. In fact, choosing a proper quantization level and upper bound for a specific network is completely subjective, because less spikes with a low-precision quantization inevitably result in a bigger accuracy loss.

Considering a successive combination of quantization level *k* in {0,1} and upper bound *B* in {1,2,4}, we report six different test accuracies for LeNet on MNIST and VGG-Net on CIFAR-10/100 (**Figure 9**). It should be noted we choose not quantize the first and last layer because they are usually used for an image-to-spike encoding and loss calculation as in Esser et al. (2016).

[3] *mCn* represents a convolutional layer with *m* filters and filter size of *n* ∗ *n*. *mPn* is a pooling layer with *m* ∗ *m* size and stride of *n*. It should be noted that we use convolution with stride of 2 to replace pooling. mFC is the fully connected layer with *m* neurons.
[4] http://www.cs.toronto.edu/~kriz/cifar.html

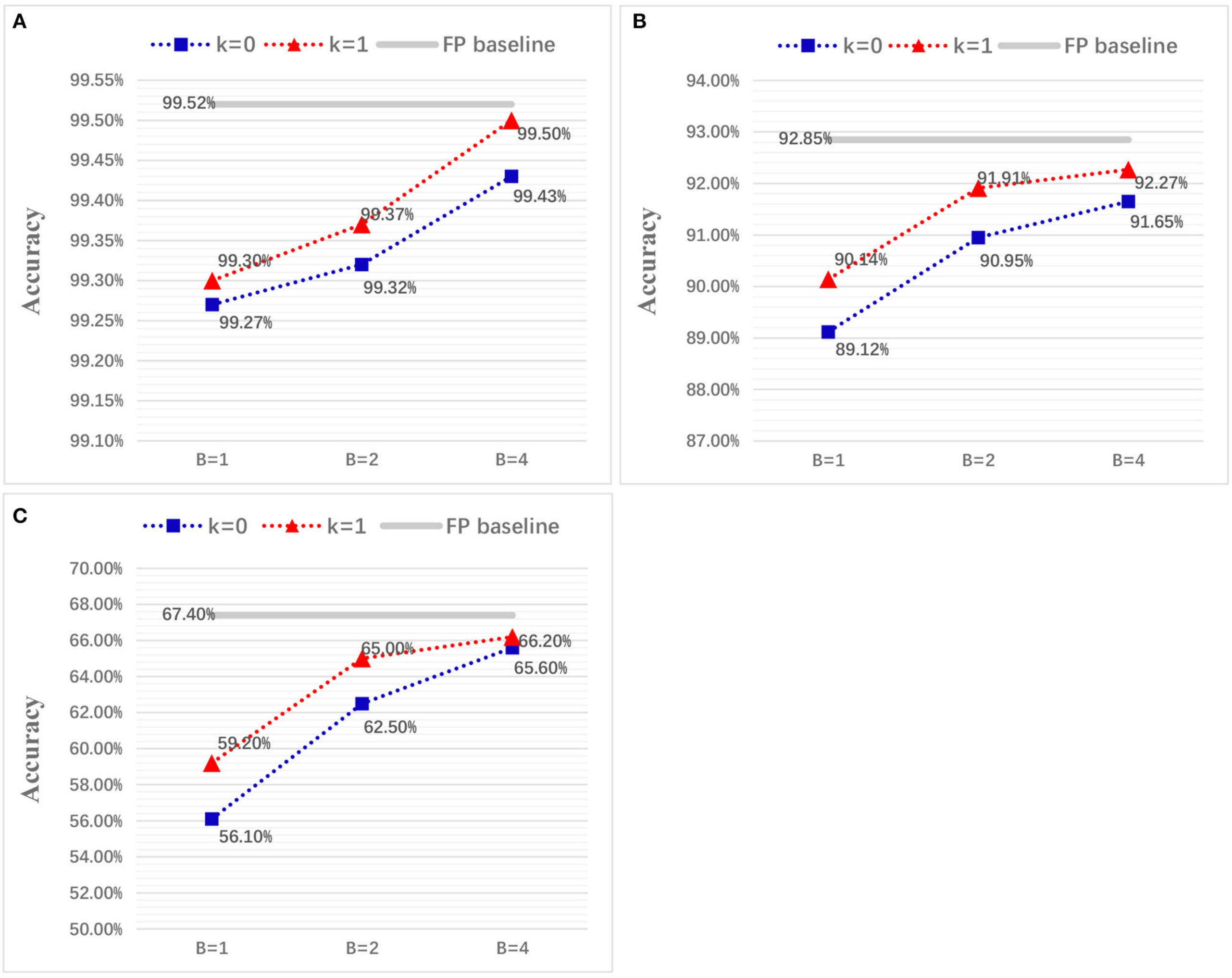


FIGURE 9 | Classification accuracy for LeNet on MNIST **(A)** and VGG-Net on CIFAR-10/100 dataset **(B,C)**, with different quantization precisions.

Experimental results show that the final accuracy can benefit from both of higher quantization level and upper bound. More importantly, we find that the spiking LeNet and VGG-Net with quantization level $k = 1$ and upper bound $B = 4$ are on a par with their full-precision (FP) baselines. For a comparison with other works, we summarize our results (for $k = 1$, $B = 2$) and many state-of-the-art works in **Tables 4–7**. It shows that our proposed spiking models are lossless with their quantized ANN counterparts and able to achieve great performance on MNIST among other works, and even better on CIFAR-10/100 dataset. In all experiments except for full-precision baseline, both of weights and activations adopt a low-precision quantization not full-precision (16 or 32 bits) like many others (Bodo et al., 2017; Xu et al., 2017). On the contrary, using this low-precision quantization does not harm to the final accuracy, but enables a cheap memory budget on many popular neuromorphic systems such as Akopyan et al. (2015), Davies et al. (2018), and Kuang et al. (2021). More specially, our networks complete simulation for one input sample within only one time step, compared with other conversion methods with dozens even hundreds of simulation time steps (Lee et al., 2016, 2020; Bodo et al., 2017; Mostafa et al., 2017; Xu et al., 2017; Rueckauer and Liu, 2018; Wu et al., 2018; Yousefzadeh et al., 2019).

For evaluation on robustness, we impose two different levels of noises (10%, 20%) on the neurons of input layer. More specifically, the IF neuron branches in **Figure 2** will be randomly shut down and never give spike outputs. This robustness evaluation is very similar to the Dropout technique (Srivastava et al., 2014), but we use it at network inference stage. We test LeNet on MNIST and VGG-Net CIFAR-10/100 with quantization precision $k = 1$ and $B = 2$ as in **Tables 4–7**. It shows our spiking networks are robust enough to tolerate broken neurons (input layer with noises) with maximum degradation of 3% when noise ratio is up to 20%. For noise at 10% level, our spiking LeNet even shows a slightly better accuracies.

TABLE 4 | Classification accuracy on MNIST.

	Activation quantization	ANN	SNN
This work (Full precision)	**N**	**99.52%**	**N/A**
This work (Rate coding)	***k* = 1, *B* = 2**	**99.37%**	**99.37%**
This work (10% noise)	***k* = 1, *B* = 2**	**99.37%**	**99.39%**
This work (20% noise)	***k* = 1, *B* = 2**	**99.37%**	**99.22%**
Mostafa et al. (2017) (Temporal coding)	N	98.50%	96.98%
Rueckauer and Liu (2018) (Temporal coding)	N	98.96%	98.57%
Wu et al. (2018) (Rate coding)	N	N/A	99.42%
Yousefzadeh et al. (2019) (Rate coding)	N	99.21%	99.19%
Bodo et al. (2017) (Rate coding)	N	99.44%	99.44%

The bold values are our experimental results.

TABLE 5 | Classification accuracy on CIFAR-10.

	Activation quantization	ANN	SNN
This work (Full precision)	**N**	**92.85%**	**N/A**
This work (Rate coding)	***k* = 1, *B* = 2**	**91.91%**	**91.91%**
This work (10% noise)	***k* = 1, *B* = 2**	**91.91%**	**90.32%**
This work (20% noise)	***k* = 1, *B* = 2**	**91.91%**	**89.65%**
Esser et al. (2016) (Rate coding)	1-bit	N/A	89.32%
Bodo et al. (2017) (Rate coding)	N	88.87%	88.82%
Lee et al. (2016) (Rate coding)	N	85.97%	83.54%
Lee et al. (2020) (Rate coding)	N	91.98%	90.54%

The bold values are our experimental results.

TABLE 6 | Classification accuracy on CIFAR-100.

	Activation quantization	ANN	SNN
This work (Full precision)	**N**	**67.4%**	**N/A**
This work (Rate coding)	***k* = 1, *B* = 2**	**65.0%**	**65.0%**
This work (10% noise)	***k* = 1, *B* = 2**	**65.0%**	**63.93%**
This work (20% noise)	***k* = 1, *B* = 2**	**65.0%**	**62.25%**
Esser et al. (2016) (Rate coding)	1-bit	N/A	65.48%

The bold values are our experimental results.

4.3. Mapping Results

For verifying effectiveness of our mapping algorithm, we carry out practical mapping for spiking LeNet and VGG-Net with various spike encoding precisions onto our neuromorphic chip. The mapping results of spiking LeNet and VGG-Net are summarized in **Tables 6**, **7**. As a convention, we denote the networks with the configurations of $\{k = 0, B = 1\}$, $\{k = 0, B = 2\}$, and $\{k = 1, B = 2\}$ as single-spike, two-spike, and four-spike model, respectively. From the two tables, it can be found either different quantization precisions or model sizes show different resource utilization while both spiking LeNet and VGG-Net with higher spike encoding precisions bring linearly better resource efficiency. For spatial mapping of scatter-and-gather SNNs, it is easy to understand that the input and output spike representation with higher precision mean a smaller patch height h_l and width w_l and increase effective synaptic connections for each output neuron, but total patch number, i.e., $m_l * n_l * p_l$ would be bigger.

For LeNet convolutional layer with dozens of channels, the height (h_l) and width (w_l) of each patch are much bigger than the kernel size, because all of the output channels (d_{l+1}) can be placed on one neuro-synaptic core according to **Algorithm 1**. Also, it can be seen that the shape of patches are not square occasionally, which can be explained by a balance between w_l,h_l and d_{l+1} discussed in **Algorithm 2**. For example, the first layer of two-spike LeNet choose an input patch of $8 * 6 * 16$ instead of $7 * 7 * 16$ or $6 * 6 * 16$ as the final mapping plan to attain fine-tuned resource efficiency. This is because the same area (the product of w_l, h_l) of input patch with unequal height and width ($w_l \neq h_l$) results in a smaller area (the product of w_{l+1},h_{l+1}) for output patch but allow a bigger capacity to hold all of output channel ($d_{l+1} = D_{l+1}$). This tradeoff is more explicit in VGG-Net with hundreds of channels. **Table 8** shows that the channel-major and square-major priorities acquire a more symmetric mapping, where the height and width of each patch are usually equal to kernel size (c_{l+1}). Although each patch cannot contain all output channels ($d_{l+1} < D_{l+1}$), the mapping algorithm improves the overall resource efficiency ($Density_{synapse}$) compared with LeNet.

Moreover, total component neurons and spiking sparsity of LeNet and VGG-Net running on chip are listed in **Figure 10**. Higher spike precisions significantly bring more spikes and neuron occupations. However, spiking sparsity (spiking times

TABLE 7 | Mapping results for LeNet.

	Single-spike						Two-spike						Four-spike					
	Input patch	Output patch	f	A[b]	N	S	Input patch	Output patch	f	A	N	S	Input patch	Output patch	f	A	N	S
16C5[a]	9 * 8 * 16	5 * 4 * 16	1	1.0	0.63	0.22	8 * 6 * 16	4 * 2 * 16	2	0.67	1.0	0.35	6 * 6 * 16	2 * 2 * 16	2	1.0	1.0	0.69
2P2	8 * 8 * 16	4 * 4 * 16	1	0.89	0.50	0.03	6 * 6 * 16	3 * 3 * 16	1	1.0	0.56	0.06	4 * 4 * 16	2 * 2 * 16	1	0.89	0.5	0.11
16C5	8 * 8 * 16	4 * 4 * 32	1	0.89	1.0	0.35	6 * 6 * 16	2 * 2 * 32	1	1.0	0.5	0.35	6 * 5 * 16	2 * 1 * 32	2	0.83	1.0	0.69
2P2	4 * 8 * 32	2 * 4 * 32	1	0.89	0.5	0.03	4 * 4 * 32	2 * 2 * 32	1	0.89	0.5	0.06	4 * 2 * 32	2 * 1 * 32	1	0.89	0.5	0.22
256FC	4 * 4 * 32	1 * 1 * 256	1	0.44	0.5	0.22	4 * 4 * 32	1 * 1 * 256	1	0.89	1.0	0.89	4 * 4 * 32	1 * 1 * 64	2	0.89	1.0	0.89
10FC	1 * 1 * 256	1 * 1 * 10	1	0.22	0.02	0.01	1 * 1 * 256	1 * 1 * 10	1	0.44	0.04	0.02	1 * 1 * 256	1 * 1 * 10	1	0.89	0.08	0.07

The first and last layer are usually processed off chip and not considered here.
[a]Layer is described as output channels-layer type-kernel size, where C is convolution, P is pooling and FC is the fully connected layer.
[b]The initial abbreviation A, N, and S refer to three evaluation criteria including $Density_{axon}$, $Density_{neuron}$ and $Density_{synapse}$ introduced in section 3.

TABLE 8 | Mapping results for VGG-Net.

	Single-spike						Two-spike						Four-spike					
	Input patch	Output patch	f	A	N	S	Input patch	Output patch	f	A	N	S	Input patch	Output patch	f	A	N	S
64C3	4 * 4 * 64	2 * 2 * 64	1	0.89	0.5	0.25	4 * 3 * 64	2 * 1 * 64	2	0.67	1.0	0.5	3 * 3 * 64	1 * 1 * 64	2	1.0	1.0	1.0
64C3	4 * 4 * 64	2 * 2 * 64	1	0.89	0.5	0.25	4 * 3 * 64	2 * 1 * 64	2	0.67	1.0	0.5	3 * 3 * 64	1 * 1 * 64	2	1.0	1.0	1.0
2P2	4 * 4 * 64	2 * 2 * 64	1	0.89	0.5	0.11	4 * 2 * 64	2 * 1 * 64	1	0.89	0.5	0.22	2 * 2 * 64	1 * 1 * 64	1	0.89	0.5	0.44
128C3	4 * 4 * 64	2 * 2 * 128	1	0.89	1.0	0.5	3 * 3 * 64	1 * 1 * 128	1	1.0	0.5	0.5	3 * 3 * 64	1 * 1 * 64	2	1.0	1.0	1.0
128C3	4 * 3 * 128	2 * 1 * 128	2	0.67	1.0	0.5	3 * 3 * 128	1 * 1 * 128	2	1.0	1.0	0.5	3 * 3 * 128	1 * 1 * 32	4	1.0	1.0	1.0
2P2	4 * 2 * 128	2 * 1 * 128	1	0.89	0.5	0.22	2 * 2 * 128	1 * 1 * 128	1	0.89	0.5	0.44	2 * 2 * 128	1 * 1 * 64	2	0.89	1.0	0.89
256C3	3 * 3 * 128	1 * 1 * 256	1	1.0	0.5	0.5	3 * 3 * 128	1 * 1 * 128	2	1.0	1.0	1.0	3 * 3 * 128	1 * 1 * 32	4	1.0	1.0	1.0
256C3	3 * 3 * 256	1 * 1 * 256	2	1.0	1.0	1.0	3 * 3 * 256	1 * 1 * 64	4	1.0	1.0	1.0	3 * 3 * 256	1 * 1 * 16	8	1.0	1.0	1.0
2P2	2 * 2 * 256	1 * 1 * 256	1	0.89	0.5	0.44	2 * 2 * 256	1 * 1 * 128	2	0.89	1.0	0.89	2 * 2 * 256	1 * 1 * 32	4	0.89	1.0	1.0
512C3	3 * 3 * 256	1 * 1 * 256	2	1.0	1.0	1.0	3 * 3 * 256	1 * 1 * 64	4	1.0	1.0	1.0	3 * 3 * 256	1 * 1 * 16	8	1.0	1.0	1.0
512C3	3 * 3 * 512	1 * 1 * 128	4	1.0	1.0	1.0	3 * 3 * 512	1 * 1 * 32	8	1.0	1.0	1.0	3 * 3 * 512	1 * 1 * 8	16	1.0	1.0	1.0
10FC	1 * 1 * 512	1 * 1 * 10	1	0.44	0.02	0.01	1 * 1 * 512	1 * 1 * 10	1	0.89	0.04	0.03	1 * 1 * 512	1 * 1 * 10	2	0.89	0.16	0.14

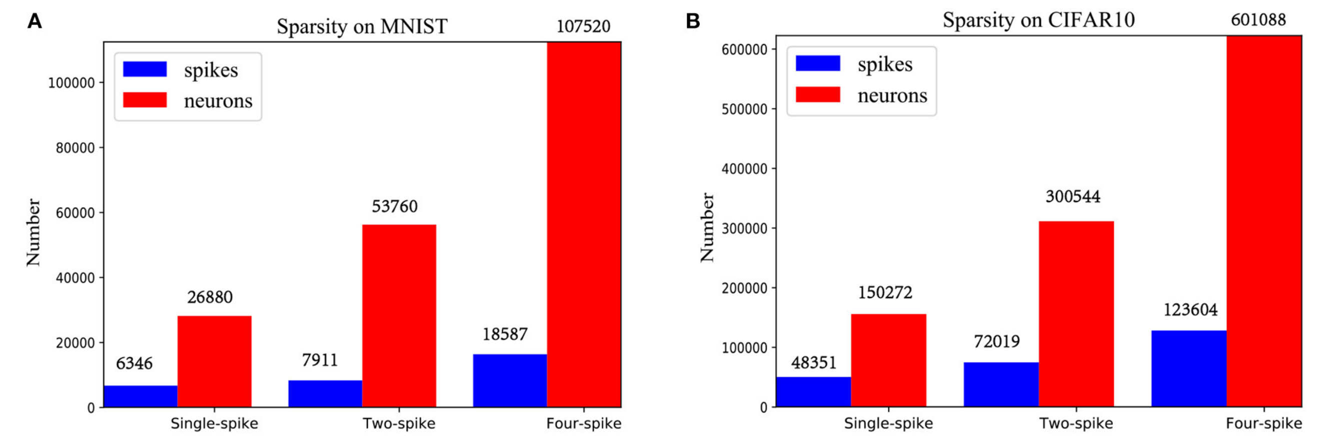


FIGURE 10 | Spiking sparsity of spiking LeNet **(A)** and VGG-Net **(B)** with different precisions.

per neuron) is gradually decreasing, from about 0.23 to 0.17 for LeNet and 0.32 to 0.21 for VGG-Net. This result corresponds to the fact that neurons with higher thresholds in an IF neuron group **Figure 2** is more difficult to generate spikes.

4.4. Speed and Power Analysis

Since this kind of multi-chip system is quite difficult to be instrumented to measure total power, such testing tools are presently undergoing development. For total performance

TABLE 9 | Chip utilization and latency for LeNet.

	Single-spike		Two-spike		Four-spike	
	Cores	**Latency (ms)**	**Cores**	**Latency (ms)**	**Cores**	**Latency (ms)**
16C5	36	0.0549	72	0.1097	144	0.1646
2P2	9	0.0549	16	0.0549	36	0.0549
16C5	4	0.0549	16	0.0549	32	0.0549
2P2	2	0.0549	4	0.0549	8	0.0549
256FC	1	0.0549	1	0.0549	1	0.0549
Total	52	0.2745	109	0.3293	221	0.3842

TABLE 10 | Chip utilization and latency for VGG-Net.

	Single-spike		Two-spike		Four-spike	
	Cores	**Latency (ms)**	**Cores**	**Latency (ms)**	**Cores**	**Latency (ms)**
64C3	144	0.1646	288	0.2743	576	0.4937
64C3	144	0.1646	288	0.2743	576	0.4937
2P2	36	0.0549	72	0.1097	144	0.1646
128C3	36	0.0549	144	0.1646	288	0.2743
128C3	72	0.1097	144	0.1646	576	0.4937
2P2	18	0.0549	36	0.0549	72	0.1097
256C3	36	0.0549	72	0.1097	288	0.2743
256C3	36	0.0549	144	0.1646	576	0.4937
2P2	9	0.0549	18	0.0549	72	0.1097
512C3	18	0.0549	72	0.1097	288	0.2743
512C3	4	0.0549	16	0.0549	64	0.0549
Total	553	0.8781	1294	1.5362	3520	3.2366

evaluation including network inference speed and power consumption with various model workloads, we adopt a mixed software–hardware methodology as in (Esser et al., 2016; Deng et al., 2020). We run an $8 * 8$ chip array in software simulation environment while refer to a actual single-chip performance. For a convolutional or pooling layer less than the capacity of $8 * 8$ multi-chip system, such as LeNet, our system can achieve fully unfolded running of this layer within only one computing period. If the size of a layer exceeds the system capacity such as VGG-Net, the direct-memory-access (DMA) controller needs to take data from off-chip memory and write it to the other on-chip SRAM and perform a ping-pong simulation. As provided in Kuang et al. (2021), our chip is operated at a power-supply voltage of 0.9 V, 252 MHz, and achieves up to 21.5 GSOPs and 0.57 pJ/SOP computational performances (idle power contributions are included). The inference latency of each layer of spiking LeNet and VGG-Net for different spike precisions is summarized in **Tables 9**, **10** It can be seen that all inference latency is at millisecond level but is not linearly proportional to the resource budgets (cores). This is because different spike precisions or model sizes bring different resource utilization as discussed in the last section.

Furthermore, we count the average number of synaptic operations (SOPs) of one sample simulation for spiking LeNet on MNIST and VGG-Net on CIFAR-10 with different spike precisions (see **Figure 11**). Each synaptic operation delivers a 1-bit spike event through a unique 1-bit non-zero synapse and adds it to membrane potential. It should be noted that for SNNs on our neuromorphic system, no multiplication operation is performed and only low-bit addition is required. Moreover, there are no computation budgets for a synaptic node without spike input, a neuron need update its state only when a spike from the previous layer is coming, so the active power would be proportional to firing activity, i.e., the number of synaptic operations. Total power consumption P_{total} is the sum of (a) leakage power P_{leak}, which is scaled by measuring idle power for single chip, and (b) active power P_{active}, which can be calculated with the number of SOPs during network inference.

In all cases, the first (the transduction layer) and last layer (the classification layer) are computed off-chip to convert multivalued image inputs into a series of binary spike trains and obtain the final decoding output, respectively. **Table 11** shows our results for the evaluated spiking LeNet and VGG-Net on MNIST and CIFAR-10 dataset, with their corresponding accuracies, throughput, power and classifications per energy (FPS per Watt). It can be seen that higher spike precisions for both LeNet and VGG-Net bring higher classification accuracy but larger inference power and latency. The four-spike LeNet and VGG-Net on chips achieve a real-time inference speed of 0.38 ms/image, 3.24 ms/image, and an average power consumption of 0.28 and 2.3 mJ/image, respectively, at 0.9 V, 252 MHz. Compared with GPUs (Titan Xp and Tesla V100) computing with the default FP32 precision, our system can obtain comparable accuracies but nearly two orders of magnitude power efficiency improvements. On the other hand, our results show that we can achieve a close classification speed on CIFAR-10 compared with TrueNorth (Esser et al., 2016) and even faster than Tianjic chip (Deng et al., 2020). The weakness in power efficiency (FPS/W) results from heavy communication workloads for off-chip memory access and inter-chip routing because of the relatively smaller ($8 * 8$) system capacity for ours, while the other two adopt quite large-scale multi-core design (4,096 for TrueNorth, 156 for Tianjic) and asynchronous communication protocol (TrueNorth).

5. CONCLUSION AND DISCUSSION

In this work, we introduce an adjustable quantization and training algorithm for ANNs to minimize common spike approximation errors, and propose a scatter-and-gather rate-based conversion method for SNNs built with simple IF neurons. Besides, we develop an incremental and resource-efficient mapping framework for these SNNs on a reconfigurable neuromorphic ASIC. Experimental results show that our spiking LeNet on MNIST and VGG-Net on CIFAR-10/100 dataset yield great classification accuracies. Meanwhile, the employment with our presented mapping algorithm is able to flexibly manage network topology placement on target neuromorphic chip with maximum resource efficiency. The four-spike LeNet on MNIST and VGG-Net CIFAR-10 on our system achieve millisecond-level speed and millijoule-level power. It should be noted that

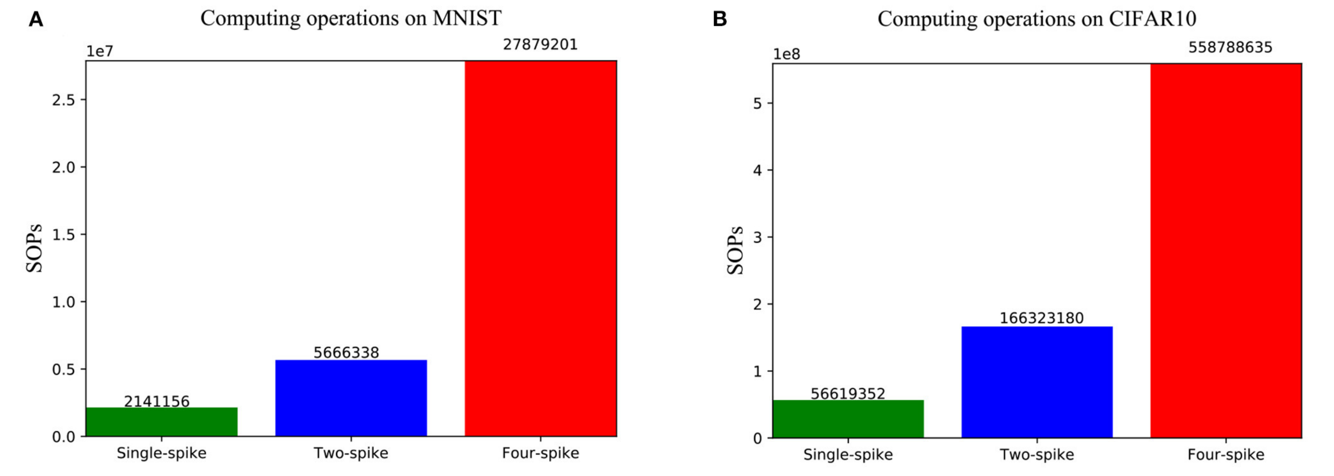


FIGURE 11 | The average number of synaptic operations (SOPs) of one input sample when running spiking LeNet **(A)** and VGG-Net **(B)** with different precisions.

TABLE 11 | Summary of main performance.

	Models	Accuracy	FPS[a]	mJ[b]	FPS/W
MNIST	Single-spike (**This work**)	99.27%	3642	0.1897	5271
	Two-spike (**This work**)	99.32%	3036	0.2293	4361
	Four-spike (**This work**)	99.37%	2602	0.2751	3635
	TrueNorth (Esser et al., 2015)	99.42%	1000	0.121	8264
	Tianjic (Deng et al., 2020)	99.48%	2126	0.069	14555
	Titan Xp (FP32 precision)	99.52%	1433	35	29
	V100 (FP32 precision)	99.52%	2185	22	45
CIFAR-10	Single-spike (**This work**)	89.12%	1138	0.6148	1626
	Two-spike (**This work**)	90.95%	650	1.0854	921
	Four-spike (**This work**)	91.91%	308	2.3013	434
	TrueNorth (Esser et al., 2016)	83.41%	1249	0.1637	6109
	Tianjic (Deng et al., 2020)	93.52%	1751	0.12	8217
	Titan Xp (FP32 precision)	92.85%	617	67	15
	V100 (FP32 precision)	92.85%	1181	42	24

[a]FPS is denoted as frames/second and FPS/W is fames/second per Watt. [b]The average energy consumption for one input frame inference.

in this power and speed evaluation stage, we treat the inter-chip communication identical with the intra-chip one. However, for a normal multi-chip system, the inter-chip communication is usually more expensive. Hence, integrating multiple computing cores into a single chip to reduce inter-chip communication is a main future work. Besides, a more thoughtful mapping scheme with the consideration of overall resource and communication can also help to alleviate cross-chip overhead. For more complicated applications, future works will concentrate on the conversion and mapping function on other architecture such as ResNet and RNN. A more rewarding work is to try training and mapping of hybrid-precision models. This may bring a further performance improvement on this neuromorphic chip.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

CZ and XC proposed the idea, designed, and conducted the experiments. YK and KL helped to complete hardware measurement and software simulation. CZ, XC, and YW wrote the manuscript, then RH revised it. XC, YW, and XW directed the project and provided overall guidance. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported by the National Key Research and Development Program of China (Grant No. 2018YFB2202605), National Natural Science Foundation of China (Grant No. 61421005), the 111 Project (B18001), and R&D Project of Shenzhen Science and Technology Innovation Committee (Grant Nos. JCYJ20200109120404043 and KQTD20200820113105004).

REFERENCES

Abbott, L. F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Res. Bull.* 50, 303–304. doi: 10.1016/S0361-9230(99)00161-6

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., and Modha, D. S. (2015). Truenorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Bengio, Y., Nicholas, L., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *eprint arxiv*.

Boahen, K. A. (2000). Point-to-point connectivity between neuromorphic chips using address events. *IEEE Trans. Circ. Syst. II Anal. Digit. Signal Process*. 47, 416–434. doi: 10.1109/82.842110

Bodo, R., Iulia-Alexandra, L., Hu, Y., Michael, P., and Liu, S. C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci*. 11:682. doi: 10.3389/fnins.2017.00682

Bohtea, S. M., Kokac, J. N., and Poutréab, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E., et al. (2019). Spiking neural networks hardware implementations and challenges. *ACM J. Emerg. Technol. Comput. Syst*. 15. doi: 10.1145/3304103

Cassidy, A. S., Merolla, P., Arthur, J. V., Esser, S. K., Jackson, B., Alvarez-Icaza, R., et al. (2013). "Cognitive computing building block: a versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 1–10.

Chen, Y., Xie, Y., Song, L., Chen, F., and Tang, T. (2020). A survey of accelerator architectures for deep neural networks. *Engineering* 6, 264–274. doi: 10.1016/j.eng.2020.01.007

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Deng, L., Li, G., Han, S., Shi, L., and Xie, Y. (2020). Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108, 485–532. doi: 10.1109/JPROC.2020.2976475

Deng, L., Wang, G., Li, G., Li, S., Liang, L., Zhu, M., et al. (2020). Tianjic: a unified and scalable chip bridging spike-based and continuous neural computation. *IEEE J. Solid State Circ*. 55, 2228–2246. doi: 10.1109/JSSC.2020.2970709

Dong, H., Supratak, A., Mai, L., Liu, F., Oehmichen, A., Yu, S., et al. (2017). "Tensorlayer: a versatile library for efficient deep learning development," in *Proceedings of the 25th ACM International Conference on Multimedia, MM '17* (New York, NY; Association for Computing Machinery), 1201–1204.

Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V., and Modha, D. S. (2015). "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems, Vol. 28*, eds C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett (Curran Associates, Inc.).

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A*. 113, 11441–11446. doi: 10.1073/pnas.1604850113

Falez, P., Tirilly, P., Bilasco, I. M., Devienne, P., and Boulet, P. (2019). Unsupervised visual feature learning with spike-timing-dependent plasticity: how far are we from traditional feature learning approaches? *Pattern Recognit*. 93, 418–429. doi: 10.1016/j.patcog.2019.04.016

Glorot, X., Bordes, A., and Bengio, Y. (2011). "Deep sparse rectifier neural networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 315–323.

Grning, A., and Bohte, S. (2014). "Spiking neural networks: principles and challenges," in *2014 European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

Gutig, R., and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nat. Neurosci*. 9, 420–428. doi: 10.1038/nn1643

Hayman, S. (1999). "The mcculloch-pitts model," in *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339), Vol. 6*, 4438–4439.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Quantized neural networks: training neural networks with low precision weights and activations. *J. Mach. Learn. Res*. 18.

Kingma, D., and Ba, J. (2014). Adam: A method for stochastic optimization.

Krizhevsky, A., and Hinton, G. (2009). "Learning multiple layers of features from tiny images," in *Handbook of Systemic Autoimmune Diseases*.

Kuang, Y., Cui, X., Zhong, Y., Liu, K., Zou, C., Dai, Z., et al. (2021). "A 28-nm 0.34-pj/sop spike-based neuromorphic processor for efficient artificial neural network implementations," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)* (IEEE), 1–5.

Lecun, Y. (2019). "1.1 deep learning hardware: Past, present, and future," in *2019 IEEE International Solid- State Circuits Conference-(ISSCC)* (San Francisco, CA: IEEE), 12–19.

Lecun, Y., and Bottou, L. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci*. 14:119. doi: 10.3389/fnins.2020.00119

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci*. 10:508. doi: 10.3389/fnins.2016.00508

Li, F., and Liu, B. (2016). Ternary weight networks. *eprint arxiv*.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., and Zitnick, C. L. (2014). "Microsoft coco: common objects in context," in *Computer Vision ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, Vol. 8693*, eds D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars (Cham: Springer).

Lobov, S. A., Mikhaylov, A. N., Shamshin, M., Makarov, V. A., and Kazantsev, V. B. (2020). Spatial properties of stdp in a self-learning spiking neural network enable controlling a mobile robot. *Front. Neurosci*. 14:88. doi: 10.3389/fnins.2020.00088

Mostafa, H., Pedroni, B. U., Sheik, S., and Cauwenberghs, G. (2017). "Fast classification using sparsely active spiking networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (Baltimore, MD: IEEE), 1–4.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). "Xnor-net: imagenet classification using binary convolutional neural networks," in *Computer Vision ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, Vol. 9908*, eds B. Leibe, J. Matas, N. Sebe and M. Welling (Cham: Springer).

Rueckauer, B., and Liu, S. (2018). "Conversion of analog to spiking neural networks using sparse temporal coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (Florence: IEEE), 1–5.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back propagating errors. *Nature* 323, 533–536. doi: 10.1038/323533a0

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis*. 115, 211–252. doi: 10.1007/s11263-015-0816-y

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., et al. (2017). A survey of neuromorphic computing and neural networks in hardware. *arXiv[Preprint].arXiv:1705.06963*.

Sheik, S., Pfeiffer, M., Stefanini, F., and Indiveri, G. (2013). "Spatio-temporal spike pattern classification in neuromorphic systems," in *Proceedings of the Second International Conference on Biomimetic and Biohybrid Systems (ICBBS), Living Machines'13* (Berlin; Heidelberg: Springer-Verlag), 262–273.

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *Comput. Sci*.

Springenberg, J., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: the all convolutional net. *arXiv[Preprint].arXiv:1412.6806*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res*. 15, 1929–1958.

Stckl, C., and Maass, W. (2019). Recognizing images with at most one spike per neuron. *arXiv[Preprint].arXiv:2001.01682*.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2018). Deep learning in spiking neural networks. *Neural Netw*. 111, 47–63. doi: 10.1016/j.neunet.2018.12.002

Wei, F., Zhaofei, Y., Yanqi, C., Timothee, M., Tiejun, H., and Yonghong, T. (2020). Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *arXiv [Preprint].arXiv:2007.05785*.

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci*. 12:331. doi: 10.3389/fnins.2018.00331

Xu, Y., Tang, H., Xing, J., and Li, H. (2017). "Spike trains encoding and threshold rescaling method for deep spiking neural networks," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)* (Honolulu, HI: IEEE), 1–6.

Yang, S., Gao, T., Wang, J., Deng, B., Lansdell, B., and Linares-Barranco, B. (2021a). Efficient spike-driven learning with dendritic event-based processing. *Front. Neurosci*. 15:97. doi: 10.3389/fnins.2021.601109

Yang, S., Wang, J., Hao, X., Li, H., Wei, X., Deng, B., et al. (2021b). Bicoss: toward large-scale cognition brain with multigranular neuromorphic architecture. *IEEE Trans. Neural Netw. Learn. Syst*. 1–15. doi: 10.1109/TNNLS.2020.3045492

Yousefzadeh, A., Hosseini, S., Holanda, P., Leroux, S., Werner, T., Serrano-Gotarredona, T., et al. (2019). "Conversion of synchronous artificial neural network to asynchronous spiking neural network using sigma-delta quantization," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Hsinchu: IEEE), 81–85.

Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., and Zou, Y. (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv[Preprint].arXiv:1606.06160*.

Zou, C., Cui, X., Ge, J., Ma, H., and Wang, X. (2020). "A novel conversion method for spiking neural network using median quantization," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* (Seville: IEEE), 1–5.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2021 Zou, Cui, Kuang, Liu, Wang, Wang and Huang. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

ORIGINAL RESEARCH
published: 09 December 2021
doi: 10.3389/fnins.2021.773954

Revisiting Batch Normalization for Training Low-Latency Deep Spiking Neural Networks From Scratch

Youngeun Kim and Priyadarshini Panda*

Department of Electrical Engineering, Yale University, New Haven, CT, United States

Spiking Neural Networks (SNNs) have recently emerged as an alternative to deep learning owing to sparse, asynchronous and binary event (or spike) driven processing, that can yield huge energy efficiency benefits on neuromorphic hardware. However, SNNs convey temporally-varying spike activation through time that is likely to induce a large variation of forward activation and backward gradients, resulting in unstable training. To address this training issue in SNNs, we revisit Batch Normalization (BN) and propose a temporal Batch Normalization Through Time (BNTT) technique. Different from previous BN techniques with SNNs, we find that varying the BN parameters at every time-step allows the model to learn the time-varying input distribution better. Specifically, our proposed BNTT decouples the parameters in a BNTT layer along the time axis to capture the temporal dynamics of spikes. We demonstrate BNTT on CIFAR-10, CIFAR-100, Tiny-ImageNet, event-driven DVS-CIFAR10 datasets, and Sequential MNIST and show near state-of-the-art performance. We conduct comprehensive analysis on the temporal characteristic of BNTT and showcase interesting benefits toward robustness against random and adversarial noise. Further, by monitoring the learnt parameters of BNTT, we find that we can do temporal early exit. That is, we can reduce the inference latency by $\sim 5-20$ time-steps from the original training latency. The code has been released at https://github.com/Intelligent-Computing-Lab-Yale/BNTT-Batch-Normalization-Through-Time.

Keywords: spiking neural network, batch normalization, image recognition, event-based processing, energy-efficient deep learning

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine, United States

Reviewed by:
Jason Eshraghian,
University of Michigan, United States
Elena Cerezuela,
Sevilla University, Spain
Yujie Wu,
Tsinghua University, China

****Correspondence:***
Youngeun Kim
youngeun.kim@yale.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *10 September 2021*
Accepted: *08 November 2021*
Published: *09 December 2021*

Citation:
Kim Y and Panda P (2021) Revisiting Batch Normalization for Training Low-Latency Deep Spiking Neural Networks From Scratch. Front. Neurosci. 15:773954. doi: 10.3389/fnins.2021.773954

1. INTRODUCTION

Artificial Neural Networks (ANNs) have shown state-of-the-art performance across various computer vision tasks. Nonetheless, huge energy consumption incurred for implementing ANNs on conventional von-Neumann hardware limits their usage in low-power and resource-constrained Internet of Things (IoT) environment, such as mobile phones, drones among others. In the context of low-power machine intelligence, Spiking Neural Networks (SNNs) have received considerable attention in the recent past (Cao et al., 2015; Diehl and Cook, 2015; Roy et al., 2019; Comsa et al., 2020; Panda et al., 2020). Inspired by biological neuronal mechanisms, SNNs process visual information with discrete spikes or events over multiple time-steps. Recent works have shown that the event-driven behavior of SNNs can be implemented on emerging neuromorphic hardware to yield 1–2 order of magnitude energy efficiency over ANNs (Akopyan et al., 2015; Davies et al., 2018).

Despite the energy efficiency benefits, SNNs have still not been widely adopted due to inherent training challenges. The training issue arises from the non-differentiable characteristic of a spiking neuron, generally, Integrate-and-Fire (IF) type (Burkitt, 2006), that makes SNNs incompatible with gradient descent training.

To address the training issue of SNNs, several methods, such as, *Conversion* and *Surrogate Gradient Descent* have been proposed. In ANN-SNN conversion (Diehl et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019; Han et al., 2020), off-the-shelf trained ANNs are converted to SNNs using normalization methods to transfer ReLU activation to IF spiking activity. The advantage here is that training happens in the ANN domain leveraging widely used machine learning frameworks like, PyTorch, that yield short training time and can be applied to complex datasets. But the ANN-SNN conversion method requires large number of time-steps ($\sim 500 - 1,000$) for inference to yield competitive accuracy, which significantly increases the latency and energy consumption of the SNN. On the other hand, directly training SNNs with a surrogate gradient function (Wu et al., 2018; Neftci et al., 2019; Lee et al., 2020) exploits temporal dynamics of spikes, resulting in lesser number of time-steps ($\sim 100 - 150$). However, the discrepancy between forward spike activation function and backward surrogate gradient function during backpropagation restricts the training capability. Therefore, naive SNNs without additional optimization techniques are difficult to be trained on large-scale datasets (e.g., CIFAR-100 and Tiny-ImageNet). Recently, a hybrid method (Rathi et al., 2020) that combines the conversion method and the surrogate gradient-based method shows state-of-the-art performance at reasonable latency ($\sim$ 250 time-steps). However, the hybrid method incurs sequential processes, i.e., training ANN from scratch, conversion of ANN to SNN, and training SNNs using surrogate gradient descent, that increases the total computation cost to obtain the final SNN model. Overall, training high-accuracy and low-latency SNNs from scratch still remains an open problem.

In this paper, we investigate the temporal characteristics of Batch Normalization (BN) for more advanced SNN training. The BN layer (Ioffe and Szegedy, 2015) has been used extensively in deep learning to accelerate the training process of ANNs. It is well known that BN reduces internal covariate shift (or soothing optimization landscape Santurkar et al., 2018) mitigating the problem of exploding/vanishing gradients. In SNN literature, there are a few recent works that leverage BN layers during training and have shown competitive performance for image classification tasks with low latency. Ledinauskas et al. (2020) use a standard BN layer and show the scalability of SNNs toward deep architectures with BN layers. Fang et al. (2020) propose a learnable membrane time constant with a standard BN layer. Zheng et al. (2020) present the advantage of scaling BN parameter according to the neuronal firing threshold. Even though the previous BN approaches show performance/latency improvement, we assert that there is need to explore the advantage of BN in the temporal dimension since SNNs convey information through time. The previous BN works with SNNs use a single BN parameter across all time-steps. We are essentially motivated by the question, *Can a single learnable parameter in the BN layer learn the temporal characteristics of the input spikes that vary across different time-steps?*

Different from previous works, we highlight the importance of temporal characterization of BN technique. To this end, we propose a new SNN-crafted batch normalization layer called Batch Normalization Through Time (BNTT) that decouples the parameters in the BN layer across different time-steps. BNTT is implemented as an additional layer in SNNs and is trained with surrogate gradient backpropagation. To investigate the effect of our BNTT, we compare the statistics of spike activity of BNTT with previous approaches: Conversion (Sengupta et al., 2019) and standard Surrogate Gradient Descent (Neftci et al., 2019), as shown in **Figure 1**. Interestingly, different from the conversion method and surrogate gradient method (without BNTT) that maintain reasonable spike activity during the entire time period across different layers, spike activity of layers trained with BNTT follows a gaussian-like trend. BNTT imposes a variation in spiking across different layers, wherein, each layer's activity peaks in a particular time-step range and then decreases. Moreover, the peaks for early layers occur at initial time-steps and latter layers peak at later time-steps. This phenomenon implies that learnable parameters in BNTT enable the networks to pass the visual information temporally from shallow to deeper layers in an effective manner.

The newly observed characteristics of BNTT brings several advantages. First, similar to BN, the BNTT layer enables SNNs to be trained stably from scratch even for large-scale datasets. Second, learnable parameters in BNTT enable SNNs to be trained with low latency ($\sim 25 - 50$ time-steps) and impose optimum spike activity across different layers for low-energy inference. Finally, the distribution of the BNTT learnable parameter (i.e., γ) is a good representation of the temporal dynamics of spikes. Hence, relying on the observation that low γ value induces low spike activity and vice-versa, we further propose a temporal early exit algorithm. Here, an SNN can predict at an earlier time-step and does not need to wait till the end of the time period to make a prediction.

In summary, our key contributions are as follows: (i) We explore the temporal characteristics of BN for SNNs and propose a temporally adaptive BN approach, called BNTT. (ii) BNTT allows SNNs to be implemented in a low-latency and low-energy environment. (iii) We further propose a temporal early exit algorithm at inference time by monitoring the learnable parameters in BNTT. (iv) To ascertain that BNTT captures the temporal characteristics of SNNs, we mathematically show that proposed BNTT has similar effect as controlling the firing threshold of the spiking neuron at every time step during inference.

2. BATCH NORMALIZATION

Batch Normalization (BN) reduces the internal covariate shift (or variation of loss landscape Santurkar et al., 2018) caused by the distribution change of input signal, which is a known problem of deep neural networks (Ioffe and Szegedy, 2015). Instead of calculating the statistics of total dataset, the intermediate

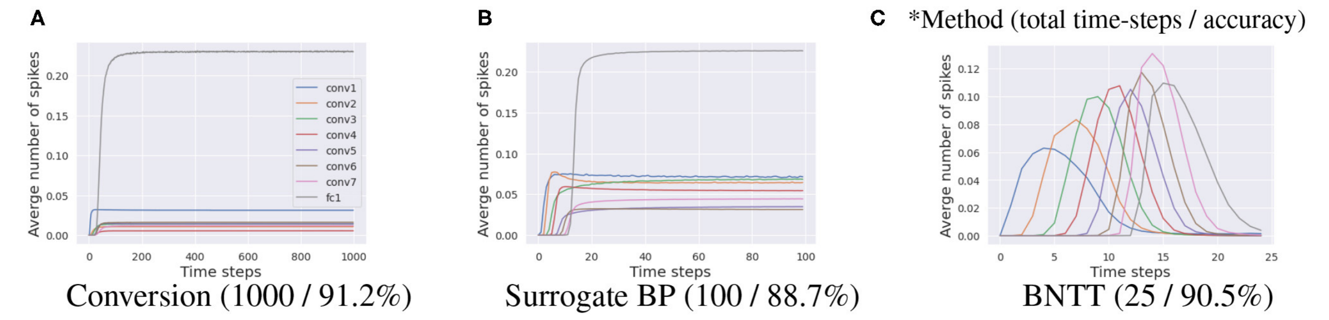


FIGURE 1 | Visualization of the average number of spikes in each layer with respect to time-steps. Compared to **(A)** ANN-SNN conversion and **(B)** surrogate gradient-based backpropagation, our **(C)** BNTT captures the temporal dynamics of spike activation with learnable parameters, enabling low-latency (i.e., small time-steps) and low-energy (i.e., less number of spikes) training. All experiments are conducted on CIFAR-10 with VGG9.

representations are standardized with a mini-batch to reduce the computation complexity. Given a mini-batch $\mathcal{B} = \{x_{1,\ldots,m}\}$, the BN layer computes the mean and variance of the mini-batch as:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{b=1}^{m} x_b; \qquad \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{b=1}^{m}(x_b - \mu_{\mathcal{B}})^2. \tag{1}$$

Then, the input features in the mini-batch are normalized with calculated statistics as:

$$\widehat{x}_b = \frac{x_b - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \tag{2}$$

where, ϵ is a small constant for numerical stability. To further improve the representation capability of the layer, learnable parameters γ and β are used to transform the input features that can be formulated as $BN(x_i) = \gamma\widehat{x}_i + \beta$. At inference time, BN uses the running average of mean and variance obtained from training. In this work, different from the static BN, we explore the temporal characteristics of BN with SNNs by enabling temporally-varying parameters in BN.

3. METHODOLOGY

3.1. Spiking Neural Networks

Different from conventional ANNs, SNNs transmit information using binary spike trains. To leverage the temporal spike information, Leaky-Integrate-and-Fire (LIF) model (Dayan and Abbott, 2001) is widely used to emulate neuronal functionality in SNNs, which can be formulated as a differential equation:

$$\tau_m \frac{dU_m}{dt} = -U_m + RI(t), \tag{3}$$

where, U_m represents the membrane potential of the neuron that characterizes the internal state of the neuron, τ_m is the time constant of membrane potential decay. Also, R and $I(t)$ denote the input resistance and the input current at time t, respectively. Following the previous work (Wu et al., 2019), we convert this continuous dynamic equation into a discrete equation for digital simulation. For a single post-synaptic neuron i, we can represent the membrane potential u_i^t at time-step t as:

$$u_i^t = \lambda u_i^{t-1} + \sum_j w_{ij} o_j^t. \tag{4}$$

Here, j is the index of a pre-synaptic neuron, λ is a leak factor with value less than 1, o_j is the binary spike activation, and w_{ij} is the weight of the connection between pre- and post-neurons. From Equation (4), the membrane potential of a neuron decreases due to leak and increases due to the weighted sum of incoming input spikes.

If the membrane potential u exceeds a pre-defined firing threshold θ, the LIF neuron i generates a binary spike output o_i. After that, we perform a soft reset, where the membrane potential u_i is reset by reducing its value by the threshold θ. Compared to a hard reset (resetting the membrane potential u_i to zero after neuron i spikes), the soft reset minimizes information loss by maintaining the residual voltage and carrying it forward to the next time step, thereby achieving better performance (Han et al., 2020). **Figure 2A** illustrates the membrane potential dynamics of a LIF neuron.

For the output layer, we discard the thresholding functionality so that neurons do not generate any spikes. We allow the output neurons to accumulate the spikes over all time-steps by fixing the leak parameter (λ in Equation 4) as one. This enables the output layer to compute probability distribution after softmax function without information loss. As with ANNs, the number of output neurons in SNNs is identical to the number of classes C in the dataset. From the accumulated membrane potential, we can define the cross-entropy loss for SNNs as:

$$L = -\sum_i y_i log(\frac{e^{u_i^T}}{\sum_{k=1}^{C} e^{u_k^T}}), \tag{5}$$

where, y is the ground-truth label, and T represents the total number of time-steps. Then, the weights of all layers are updated by backpropagating the loss value with gradient descent.

To compute the gradients of each layer l, we use back-propagation through time (BPTT), which accumulates the gradients over all time-steps (Wu et al., 2018; Neftci et al., 2019).

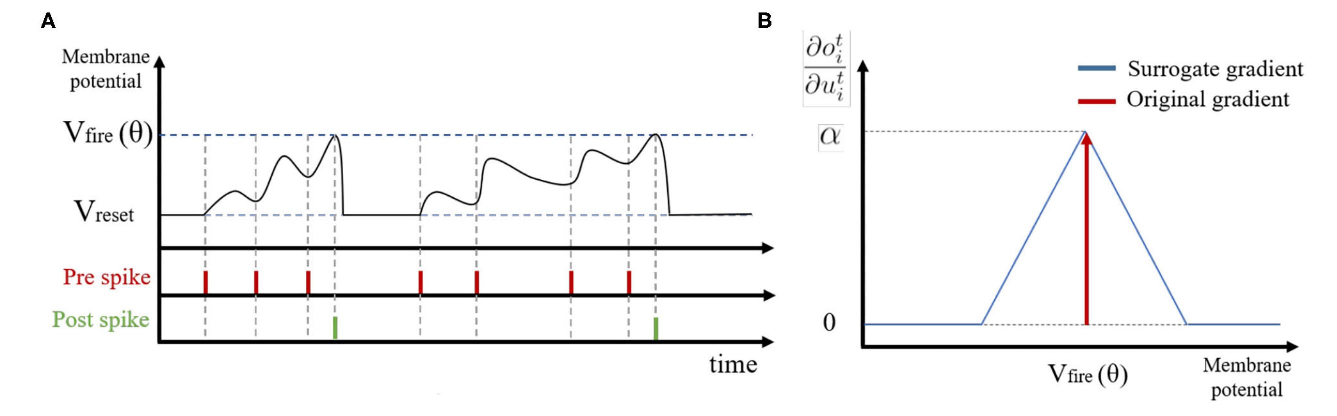


FIGURE 2 | (A) Illustration of spike activities in Leaky-Integrate-and-Fire neurons. **(B)** The approximated gradient value with respect to the membrane potential.

These approaches can be implemented with auto-differentiation tools, such as PyTorch (Paszke et al., 2017), that enable backpropagation on the unrolled network. To this end, we compute the loss function at time-step T and use gradient descent optimization. Mathematically, we can define the accumulated gradients at the layer l by chain rule as:

$$\frac{\partial L}{\partial W_l} = \begin{cases} \sum_t \left(\frac{\partial L}{\partial O_l^t}\frac{\partial O_l^t}{\partial U_l^t} + \frac{\partial L}{\partial U_l^{t+1}}\frac{\partial U_l^{t+1}}{\partial U_l^t}\right)\frac{\partial U_l^t}{\partial W_l}, & \text{if } l = \text{hidden layer} \\ \sum_t \frac{\partial L}{\partial U_l^T}\frac{\partial U_l^T}{\partial W_l}. & \text{if } l = \text{output layer} \end{cases} \tag{6}$$

Here, O_l and U_l are output spikes and membrane potential at layer l, respectively. For the output layer, we get the derivative of the loss L with respect to the membrane potential u_i^T at final time-step T:

$$\frac{\partial L}{\partial u_i^T} = \frac{e^{u_i^T}}{\sum_{k=1}^{C} e^{u_k^T}} - y_i. \tag{7}$$

This derivative function is continuous and differentiable for all possible membrane potential values. On the other hand, LIF neurons in hidden layers generate spike output only if the membrane potential u_i^t exceeds the firing threshold, leading to non-differentiability. To deal with this problem, we introduce an approximate gradient (**Figure 2B**):

$$\frac{\partial o_i^t}{\partial u_i^t} = \alpha \max\{0, 1 - |\frac{u_i^t - \theta}{\theta}|\}, \tag{8}$$

where, α is a damping factor for back-propagated gradients. Note, a large α value causes unstable training as gradients are summed over all time-steps. Hence, we set α to 0.3. Overall, we update the network parameters at the layer l based on the gradient value (Equation 6) as $W_l = W_l - \eta\Delta W_l$.

3.2. Batch Normalization Through Time (BNTT)

In this work, we present a new temporally-variant Batch Normalization for accelerating SNN training. We first visualize the distribution of the input signal of standard BN at layer 5 in VGG9 SNN with surrogate-gradients based training (**Figure 3**). The results show that the input signal to the BN layer varies with time. Therefore, we assert that if we enable temporal flexibility to BN parameters (e.g., global mean μ, global variation σ, and learnable parameter γ), the representation power of the networks might be improved.

To this end, we vary the internal parameters in a BN layer through time, that we define as, BNTT. Similar to the digital simulation of LIF neuron across different time-steps, one BNTT layer is expanded temporally with a local learning parameter associated with each time-step. This allows the BNTT layer to capture temporal statistics (see section 3.3 for mathematical analysis). The proposed BNTT layer is easily applied to SNNs by inserting the layer after convolutional/linear operations as:

$$\begin{aligned} u_i^t &= \lambda u_i^{t-1} + BNTT_{\gamma^t}(\sum_j w_{ij} o_j^t) \\ &= \lambda u_i^{t-1} + \gamma_i^t\left(\frac{\sum_j w_{ij} o_j^t - \mu_i^t}{\sqrt{(\sigma_i^t)^2 + \epsilon}}\right). \end{aligned} \tag{9}$$

During the training process, we compute the mean μ_i^t and variance σ_i^t from the samples in a mini-batch $\mathcal{B}$ for each time step t, as shown in **Algorithm 1**. Note, for each time-step t, we apply an exponential moving average to approximate global mean $\bar{\mu}_i^t$ and variance $\bar{\sigma}_i^t$ over training iterations. These global statistics are used to normalize the test data at inference. Also, we do not utilize β as in conventional BN, since it adds redundant voltage to the membrane potential of SNNs.

Adding the BNTT layer to LIF neurons changes the gradient calculation for backpropagation. Given that $x_i^t = \sum_j w_{ij} o_j^t$ is an input signal to the BNTT layer, we can calculate the gradient value passed through lower layers by the BNTT layer as:

$$\frac{\partial L}{\partial x_b^t} = \frac{1}{m\sqrt{(\sigma^t)^2 + \epsilon}}\left(m\frac{\partial L}{\partial \hat{x}_b^t} - \sum_{k=1}^{m}\frac{\partial L}{\partial \hat{x}_k^t} - \hat{x}_b^t \sum_{k=1}^{m}\frac{\partial L}{\partial \hat{x}_k^t}\hat{x}_k^t\right). \tag{10}$$

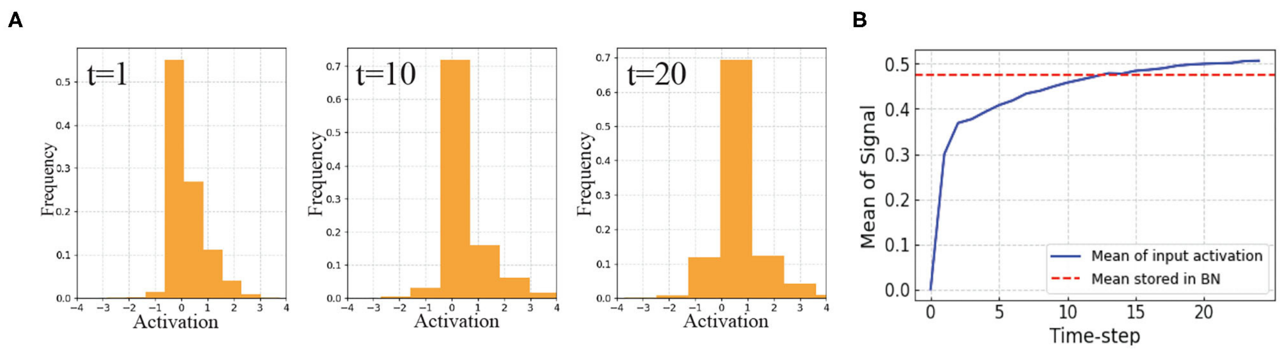


FIGURE 3 | SNNs with standard BN: **(A)** Distributions of the input activation of BN at time-step 1, 10, and 20. **(B)** While the mean of input activation varies with time, stored mean in standard BN layer has constant value at inference. This will create discrepancy and inhibit the BN layer to learn well. This suggests a temporally varying BN technique.

Here, we omit a neuron index i for simplicity. Also, m and b denote the batch size and batch index (see **Supplementary Material A** for more detail). Thus, for every time-step t, gradients are calculated based on the time-specific statistics of input signals. This allows the networks to take into account temporal dynamics for training weight connections. Moreover, a learnable parameter γ is updated to restore the representation power of the batch normalized signal. Since we use different γ^t values across all time-steps, γ^t finds an optimum over each time-step for efficient inference. We update gamma $\gamma^t = \gamma^t - \eta\Delta\gamma^t$ where:

$$\Delta\gamma^t = \frac{\partial L}{\partial \gamma^t} = \frac{\partial L}{\partial u^t}\frac{\partial u^t}{\partial \gamma^t} = \sum_{k=1}^{m} \frac{\partial L}{\partial u_k^t}\hat{x}_k^t. \tag{11}$$

3.3. Mathematical Analysis

In this section, we discuss the connections between BNTT and the firing threshold of a LIF neuron. Specifically, we formally prove that using BNTT has a similar effect as varying the firing threshold over different time-steps, thereby ascertaining that BNTT captures temporal characteristics in SNNs. Recall that BNTT normalizes the input signal using stored approximated global average $\bar{\mu}_i^t$ and standard deviation $(\bar{\sigma}_i{}^t)^2$ at inference. From Equation (9), we can calculate a membrane potential at time-step $t = 1$, given that initial membrane potential u_i^0 has a zero value:

$$\begin{aligned} u_i^1 &= \gamma_i^1 \left(\frac{\sum_j w_{ij}o_j^1 - \bar{\mu}_i^1}{\sqrt{(\bar{\sigma}_i^1)^2 + \epsilon}}\right) \\ &\approx \frac{\gamma_i^1}{\sqrt{(\bar{\sigma}_i^1)^2 + \epsilon}} \sum_j w_{ij}o_j^1 = \frac{\gamma_i^1}{\sqrt{(\bar{\sigma}_i^1)^2 + \epsilon}} \tilde{u}_i^1. \end{aligned} \tag{12}$$

Here, we assume $\bar{\mu}_i^1$ can be neglected with small signal approximation due to the spike sparsity in SNNs, and $\tilde{u}_i^1 = \sum_j w_{ij}o_j^1$ is membrane potential at time-step $t = 1$ without BNTT (obtained from Equation 4). We can observe that the membrane potential with BNTT is proportional to the membrane potential without BNTT at $t = 1$. For time-step $t > 1$, we should take into account the membrane potential from the previous time-step, which is multiplied by leak λ. To this end, by substituting (Equation 12) in the BNTT equation (Equation 9), we can formulate the membrane potential at $t = 2$ as:

$$\begin{aligned} u_i^2 &\approx \lambda u_i^1 + \frac{\gamma_i^2}{\sqrt{(\sigma_i^2)^2 + \epsilon}} \sum_j w_{ij}o_j^2 \\ &= \left(\frac{\lambda\gamma_i^1}{\sqrt{(\sigma_i^1)^2 + \epsilon}}\right)\tilde{u}_i^1 + \frac{\gamma_i^2}{\sqrt{(\sigma_i^2)^2 + \epsilon}} \sum_j w_{ij}o_j^2 \\ &\approx \frac{\gamma_i^2}{\sqrt{(\sigma_i^2)^2 + \epsilon}} \{\lambda\tilde{u}_i^1 + \sum_j w_{ij}o_j^2\} = \frac{\gamma_i^2}{\sqrt{(\sigma_i^2)^2 + \epsilon}} \tilde{u}_i^2. \end{aligned} \tag{13}$$

In the third line, the learnable parameter γ_i^t and σ_i^t have similar values in adjacent time intervals ($t = 1, 2$) because of continuous time property. Hence, we can approximate γ_i^1 and σ_i^1 as γ_i^2 and σ_i^2, respectively. Finally, we can extend the equation of BNTT to the time-step t:

$$u_i^t \approx \frac{\gamma_i^t}{\sqrt{(\sigma_i^t)^2 + \epsilon}} \tilde{u}_i^t. \tag{14}$$

Considering that a neuron produces an output spike activation whenever the membrane potential $\tilde{u}_i^t$ exceeds the pre-defined firing threshold θ, the spike firing condition with BNTT can be represented $u_i^t \geq \theta$. Comparing with the threshold of a neuron without BNTT, we can reformulate the firing condition as:

$$\tilde{u}_i^t \geq \frac{\sqrt{(\sigma_i^t)^2 + \epsilon}}{\gamma_i^t}\theta. \tag{15}$$

Thus, we can infer that using a BNTT layer changes the firing threshold value by $\sqrt{(\sigma_i^t)^2 + \epsilon}/\gamma_i^t$ at every time-step. In practice, BNTT results in an optimum γ during training that improves the representation power, producing better performance and

Algorithm 1: BNTT layer

```
Input: mini-batch $\mathcal{B}$ at time step $t$ ($x^t_{\{1...m\}}$), learnable parameter ($\gamma^t$), update factor ($\alpha$)
Output: $\{y^t = \mathrm{BNTT}_{\gamma^t}(x^t)\}$
1: $\mu^t \leftarrow \frac{1}{m}\sum_{b=1}^{m} x^t_b$
2: $(\sigma^t)^2 \leftarrow \frac{1}{m}\sum_{b=1}^{m}(x^t_b - \mu^t)^2$
3: $\hat{x}^t = \frac{x^t-\mu^t}{\sqrt{(\sigma^t)^2+\epsilon}}$
4: $y^t \leftarrow \gamma^t\hat{x}^t \equiv \mathrm{BNTT}_{\gamma^t}(x^t)$
5: % Exponential moving average
6: $\bar{\mu}^t \leftarrow (1-\alpha)\bar{\mu}^t + \alpha\mu^t$
7: $\bar{\sigma}^t \leftarrow (1-\alpha)\bar{\sigma}^t + \alpha\sigma^t$
```

Algorithm 2: Training process with BNTT

```
Input: mini-batch ($X$); label set ($Y$); max_timestep ($T$)
Output: updated network weights
1: for $i \leftarrow 1$ to $max\_iter$ do
2:     fetch a mini batch X
3:     for $t \leftarrow 1$ to $T$ do
4:         O ← PoissonGenerator(X)
5:         for $l \leftarrow 1$ to $L-1$ do
6:             $(O^t_l, U^t_l) \leftarrow (\lambda, U^{t-1}_l, \mathrm{BNTT}_{\gamma^t}(W_l, O^{t-1}_{l-1}))$
7:         end for
8:         % For the final layer $L$, stack the voltage
9:         $U^t_L \leftarrow (U^{t-1}_L, \mathrm{BNTT}_{\gamma^t}(W_l, O^{t-1}_{L-1}))$
10:    end for
11:    % Calculate the loss and back-propagation
12:    $L \leftarrow (U^T_L, Y)$
13: end for
```

low-latency SNNs.This observation allows us to consider the advantages of time-varying learnable parameters in SNNs. This implication is in line with previous work (Han et al., 2020), which insists that manipulating the firing threshold improves the performance and latency of the ANN-SNN conversion method. However, Han et al. change the threshold value in a heuristic way without any optimization process and fix the threshold value across all time-steps. On the other hand, our BNTT yields time-specific γ^t which can be optimized via back-propagation.

3.4. Early Exit Algorithm

The main objective of early exit is to reduce the latency during inference (Panda et al., 2016; Teerapittayanon et al., 2016). Most previous methods (Wu et al., 2018; Sengupta et al., 2019; Han et al., 2020; Lee et al., 2020; Rathi et al., 2020) accumulate output spikes till the end of the time-sequence, at inference, since all layers generate spikes across all time-steps as shown in **Figures 1A,B**. On the other hand, learnable parameters in BNTT manipulate the spike activity of each layer to produce a peak value, which falls again (a gaussian-like trend), as shown in **Figure 1C**. This phenomenon shows that SNNs using BNTT convey little information at the end of spike trains.

Inspired by this observation, we propose a temporal early exit algorithm based on the value of γ^t. From Equation (15), we know that a low γ^t value increases the firing threshold, resulting in low spike activity. A high γ^t value, in contrast, induces more spike activity. It is worth mentioning that $(\sigma^t_i)^2$ shows similar values across all time-steps and therefore we only focus on γ^t. Given that the intensity of spike activity is proportional to γ^t, we can infer that spikes will hardly contribute to the classification result once γ^t values across every layer drop to a minimum value. Therefore, we measure the average of γ^t values in each layer l at every time-step, and terminate the inference when γ^t value in every layer is below a pre-determined threshold. For example, as shown in **Figure 4**, we observe that all averaged γ^t values are lower than threshold 0.1 after $t > 20$. Therefore, we define the early exit time at $t = 20$. Note that we can determine the optimum time-step for early exit before forward propagation without any additional computation. In summary, the temporal early exit method enables us to find the earliest time-step during inference that ensures integration of crucial information, in turn reducing the inference latency without significant loss of accuracy.

3.5. Overall Optimization

Algorithm 2 summarizes the whole training process of SNNs with BNTT. Our proposed BNTT acts as a regularizer, unlike previous methods (Lee et al., 2016, 2020; Sengupta et al., 2019; Rathi et al., 2020) that use dropout to perform regularization. Our training scheme is based on widely used rate coding where the spike generator produces a Poisson spike train (see **Supplementary Material B**) for each pixel in the image with frequency proportional to the pixel intensity (Roy et al., 2019). For all layers, the weighted sum of the input signal is passed through a BNTT layer and then is accumulated in the membrane potential. If the membrane potential exceeds the firing threshold, the neuron generates an output spike. For last layer, we accumulate the input voltage over all time-steps without leak, that we feed to a softmax layer to output a probability distribution. Then, we calculate a cross-entropy loss function and gradients for weight of each layer with the approximate gradient function. During the training phase, a BNTT layer computes the time-dependent statistics (i.e., μ^t and σ^t) and stores the moving-average global mean and variance. At inference, we first define the early exit time-step based on the value of γ in BNTT. Then, the networks classify the test input (note, test data normalized with pre-computed global $\bar{\mu}^t$,$\bar{\sigma}^t$ BNTT statistics) based on the accumulated output voltage at the pre-computed early exit time-step.

4. EXPERIMENTS

In this section, we carry out comprehensive experiments on public classification datasets. We first compare our BNTT with previous SNNs training methods. Then, we quantitatively and qualitatively demonstrate the effectiveness of our proposed BNTT.

4.1. Experimental Setup

We evaluate our method on three static datasets (i.e., CIFAR-10, CIFAR-100, Tiny-ImageNet), one neuromophic dataset (i.e., DVS-CIFAR10), and one sequential dataset (i.e., Sequential

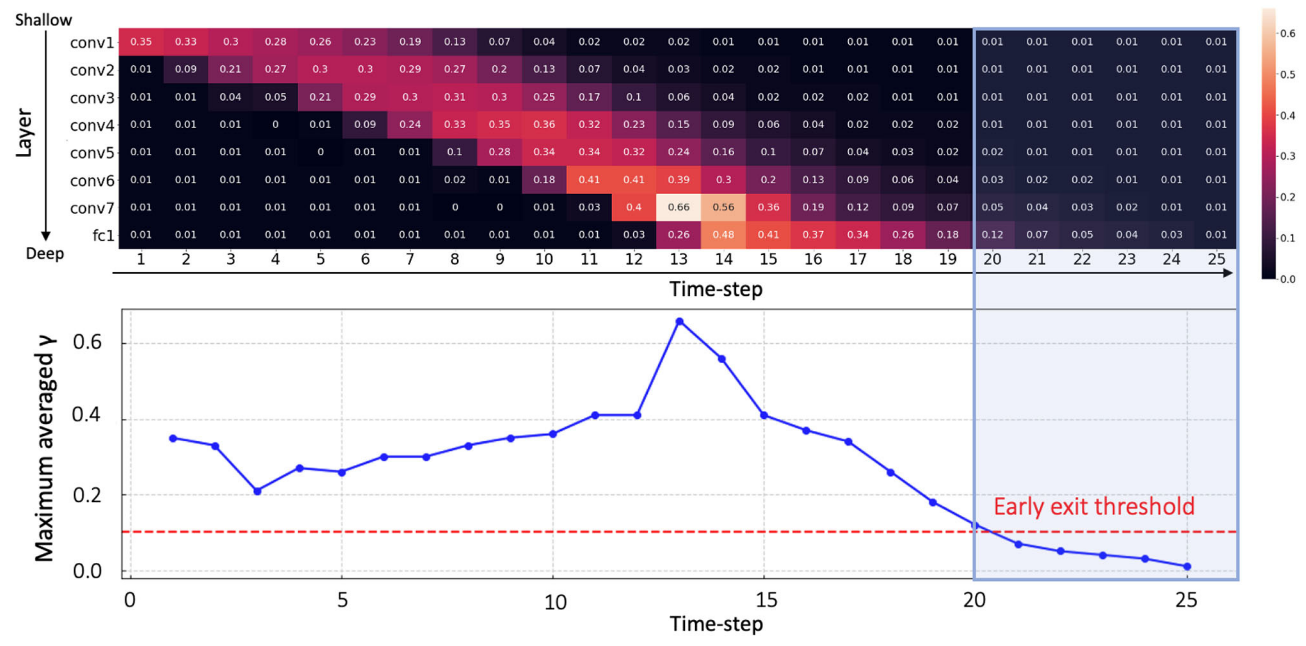


FIGURE 4 | The average value of γ^t at each layer over all time-steps (upper panel). Maximum averaged γ^t for each time-step (lower panel). Early exit time can be calculated as $t = 20$ since γ^t values at every layer have lower value than threshold 0.1 after time-step 20 (blue shaded area). Here, we use a VGG9 architecture on CIFAR-10.

MNIST). **CIFAR-10** (Krizhevsky and Hinton, 2009) consists of 60,000 images (50,000 for training/10,000 for testing) with 10 categories. All images are RGB color images whose size are 32 × 32. **CIFAR-100** has the same configuration as CIFAR-10, except it contains images from 100 categories. **Tiny-ImageNet** is the modified subset of the original ImageNet dataset. Here, there are 200 different classes of ImageNet dataset (Deng et al., 2009), with 100,000 training and 10,000 validation images. The resolution of the images is 64×64 pixels. **DVS-CIFAR10** (Li et al., 2017) has the same configuration as CIFAR-10. This discrete event-stream dataset is collected by moving the event-driven camera. We follow the similar data pre-processing protocol and a network architecture used in previous work (Wu et al., 2019) (details in **Supplementary Material C**). **Sequential MNIST** (Le et al., 2015) is the variant of MNIST (LeCun et al., 1998). Instead of showing the whole image to the networks, this dataset presents each pixel in an image pixel by pixel. Our implementation is based on Pytorch (Paszke et al., 2017). We train the networks with standard SGD with momentum 0.9, weight decay 0.0005 and also apply random crop and horizontal flip to input images. The base learning rate is set to 0.3 and we use step-wise learning rate scheduling with a decay factor 10 at 50, 70, and 90% of the total number of epochs. Here, we set the total number of epochs to 120, 240, 90, and 60 for CIFAR-10, CIFAR-100, Tiny-ImageNet, and DVS-CIFAR10, respectively.

4.2. Comparison With Previous Methods

On public datasets, we compare our proposed BNTT method with previous rate-coding based SNN training methods, including ANN-SNN conversion (Cao et al., 2015; Sengupta et al., 2019; Han et al., 2020), surrogate gradient back-propagation (Lee et al., 2020), and hybrid (Rathi et al., 2020) methods. From **Table 1**, we can observe some advantages and disadvantages of each training method. The ANN-SNN conversion method performs better than the surrogate gradient method across all datasets. However, they require large number of time-steps for training and testing, which is energy-inefficient and impractical in a real-time application. The hybrid method aims to resolve this high-latency problem, but it still requires over hundreds of time-steps. The surrogate gradient method (denoted as *Baseline*) suffers from poor optimization and hence cannot be scaled to larger datasets such as CIFAR-100 and Tiny-ImageNet. The results show that the performance improvement of SNN models is because of BNTT, and not because of applying the loss to the membrane potential which can improve the performance of SNNs (Eshraghian et al., 2021). Using standard BN with surrogate gradient training (i.e., *Baseline + standard BN*) improves the optimization capability of SNNs enabling us to train deep SNNs for complex datasets, however, there is performance degradation. Increasing the number of time-steps to $> 100 - 150$ does improve the performance, but that would also lead to increased computation. Our BNTT is based on the surrogate gradient method (i.e., Baseline + BNTT), and it enables SNNs to achieve high performance even for more complicated datasets. At the same time, we reduce the latency due to the inclusion of learnable parameters and temporal statistics in the BNTT layer. As a result, BNTT can be trained with 25 time-steps on a simple CIFAR-10 dataset, while preserving state-of-the-art accuracy. For CIFAR-100, we achieve about 40× and 2× faster inference speed compared to the conversion

TABLE 1 | Classification accuracy (%) on CIFAR-10, CIFAR-100, and Tiny-ImageNet.

	Dataset	Training method	Architecture	Time-steps	Accuracy (%)
Cao et al. (2015)	CIFAR-10	ANN-SNN Conversion	3Conv, 2Linear	400	77.4
Sengupta et al. (2019)	CIFAR-10	ANN-SNN Conversion	VGG16	2500	91.5
Lee et al. (2020)	CIFAR-10	Surrogate Gradient	VGG9	100	90.4
Rathi et al. (2020)	CIFAR-10	Hybrid	VGG16	200	92.0
Han et al. (2020)	CIFAR-10	ANN-SNN Conversion	VGG16	2048	93.6
Baseline	CIFAR-10	Surrogate Gradient	VGG9	100	88.7
Baseline + standard BN	CIFAR-10	Surrogate Gradient	VGG9	25	84.3
Baseline + BNTT (ours)	CIFAR-10	Surrogate Gradient	VGG9	25	90.5
Baseline + BNTT + Early Exit (ours)	CIFAR-10	Surrogate Gradient	VGG9	20	90.3
Sengupta et al. (2019)	CIFAR-100	ANN-SNN Conversion	VGG16	2500	70.9
Rathi et al. (2020)	CIFAR-100	Hybrid	VGG16	125	67.8
Han et al. (2020)	CIFAR-100	ANN-SNN Conversion	VGG16	2048	70.9
Baseline	CIFAR-100	Surrogate Gradient	VGG11	n/a	n/a
Baseline + standard BN	CIFAR-100	Surrogate Gradient	VGG11	50	43.0
Baseline + BNTT (ours)	CIFAR-100	Surrogate Gradient	VGG11	50	66.6
Baseline + BNTT + Early Exit (ours)	CIFAR-100	Surrogate Gradient	VGG11	30	65.8
Sengupta et al. (2019)	Tiny-ImageNet	ANN-SNN Conversion	VGG11	2500	54.2
Baseline	Tiny-ImageNet	Surrogate Gradient	VGG11	n/a	n/a
Baseline + standard BN	Tiny-ImageNet	Surrogate Gradient	VGG11	30	32.7
Baseline + BNTT (ours)	Tiny-ImageNet	Surrogate Gradient	VGG11	30	57.8
Baseline + BNTT + Early Exit (ours)	Tiny-ImageNet	Surrogate Gradient	VGG11	25	56.8

methods and the hybrid method, respectively. Interestingly, for Tiny-ImageNet, BNTT achieves better performance and shorter latency compared to previous conversion method. Note that ANN with VGG11 architecture used for ANN-SNN conversion achieves 56.3% accuracy. Moreover, using an early exit algorithm further reduces the latency by ∼ 20%, which enables the networks to be implemented with lower-latency and energy-efficiency. It is worth mentioning that surrogate gradient method without BNTT (*Baseline* in **Table 1**) only converges on CIFAR-10. For neuromorphic DVS-CIFAR10 dataset (**Table 2**), using BNTT improves the stability of training compared to a surrogate gradient baseline, and achieves state-of-the-art performance. These results show that our BNTT technique is very effective on event-driven data and hence well-suited for neuromorphic applications. We also compare the performance of BNTT with previous works on Sequential MNIST in **Table 3**. Here, we use 3-layer SNN architecture: FC(1,256)-FC(256,256)-FC(256,10). Without BNTT, *Baseline* has difficulty in capturing the sequential pattern of input data, resulting in low performance. Adding BNTT to *Baseline* enhances the training capability of SNNs, resulting in a slightly better performance than the state-of-the-art (Bellec et al., 2018).

TABLE 2 | Classification accuracy (%) on DVS-CIFAR10.

Method	Type	Accuracy (%)
Orchard et al. (2015)	Random forest	31.0
Lagorce et al. (2016)	HOTS	27.1
Sironi et al. (2018)	HAT	52.4
Sironi et al. (2018)	Gabor-SNN	24.5
Wu et al. (2019)	Surrogate gradient	60.5
Baseline	Surrogate gradient	n/a
Baseline + BNTT (ours)	Surrogate gradient	63.2

TABLE 3 | Classification accuracy (%) on sequential MNIST.

Method	Accuracy (%)
LIF (Bellec et al., 2018)	63.3
LSNN (Bellec et al., 2018)	93.7
DEEP R LSNN (Bellec et al., 2018)	96.4
Baseline	36.2
Baseline + BNTT (ours)	96.6

4.3. Comparison With the Previous BN Techniques for SNNs

We compare our temporal BNTT technique with the previous BN approaches for SNN in **Table 4**. The approaches with the standard BN (Fang et al., 2020; Ledinauskas et al., 2020) do not show scalability to complicated datasets such as CIFAR-100 and Tiny-ImageNet. Compared to this, our approach enables training SNNs with low latency on such datasets. Zheng et al. (2020) show the advantage of scaling BN parameter according to the firing threshold, which shows good performance for large-scale

TABLE 4 | Comparison between different BN techniques for SNNs.

	Method	CIFAR-10	CIFAR-100	Tiny-imageNet	ImageNet
Ledinauskas et al. (2020)	Standard BN	90.2	58.5	-	-
Fang et al. (2020)	Standard BN	93.5	-	-	-
Zheng et al. (2020)	Threshold-dependent BN	93.2	-	-	67.1
BNTT (ours)	Temporal BN	90.5	66.6	57.8	-

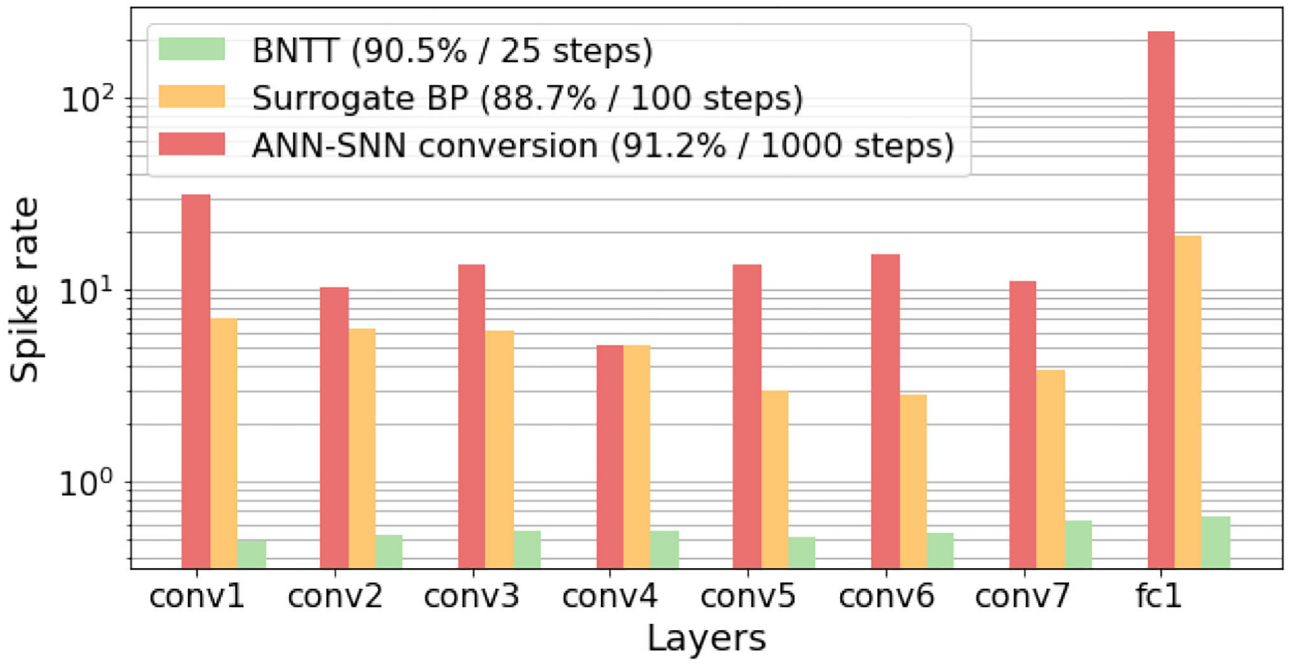


FIGURE 5 | Visualization layer-wise spike activity (log scale) in VGG9 on CIFAR-10 dataset.

datasets, including ImageNet. Our objective is to study the effect of BN in temporal domain, not enhance the capability of BN itself, which is different from their approach. Combining these two orthogonal approaches in order to achieve further performance gain can be a good topic for future work.

4.4. Spike Activity Analysis

We compare the layer-wise spiking activities of our BNTT with two widely-used methods, i.e., ANN-SNN conversion method (Sengupta et al., 2019) and surrogate gradient method (without BNTT) (Neftci et al., 2019). Specifically, we calculate the spike rate of each layer l, which can be defined as the total number of spikes at layer l over total time-steps T divided by the number of neurons in layer l (see **Supplementary Material D** for the equation of spike rate). In **Figure 5**, converted SNNs show a high spike rate for every layer as they forward spike trains through a larger number of time-steps compared to other methods. Even though the surrogate gradient method uses less number of time-steps, it still requires nearly hundreds of spikes for each layer. Compared to these methods, we can observe that BNTT significantly improves the spike sparsity across all layers. In addition, we conduct further energy comparison on Neuromorphic architecture in **Supplementary Material E**.

4.5. Analysis on Learnable Parameters in BNTT

The key observation of our work is the change of γ across time-steps. To analyze the distribution of the learnable parameters in our BNTT, we visualize the histogram of γ in conv1, conv4, and conv7 layers in VGG9 as shown in **Figure 6**. Interestingly, all layers show different temporal evolution of gamma distributions. For example, conv1 has high γ values at the initial time-steps which decrease as time goes on. On the other hand, starting from small values, the γ values in conv4 and conv7 layers peak at $t = 9$ and $t = 13$, respectively, and then shrink to zero at later time-steps. Notably, the peak time is delayed as the layer goes deeper, implying that the visual information is passed through the network sequentially over a period of time similar to **Figure 1C**. This gaussian-like trend with rise and fall of γ across different time-steps can support the explanation of overall low spike activity compared to other methods (**Figure 5**).

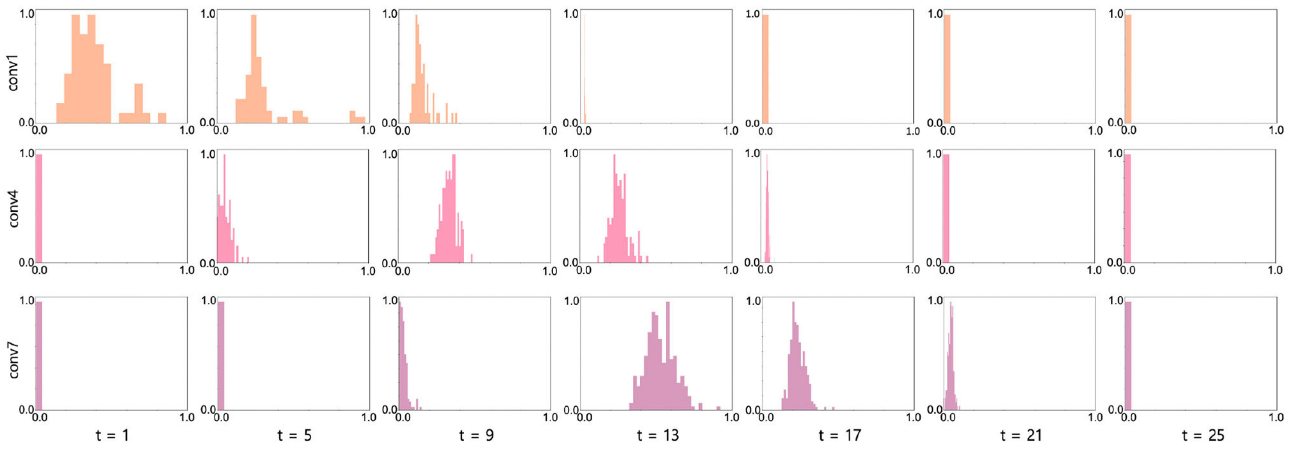


FIGURE 6 | Histogram visualization (x axis: γ value, y axis: frequency) at conv1 (row1), conv4 (row2), and conv7 (row3) layers in VGG9 across all time-steps. We normalize the frequency into range [0, 1] for better visualization. The experiments are conducted on CIFAR-10 with 25 time-steps.

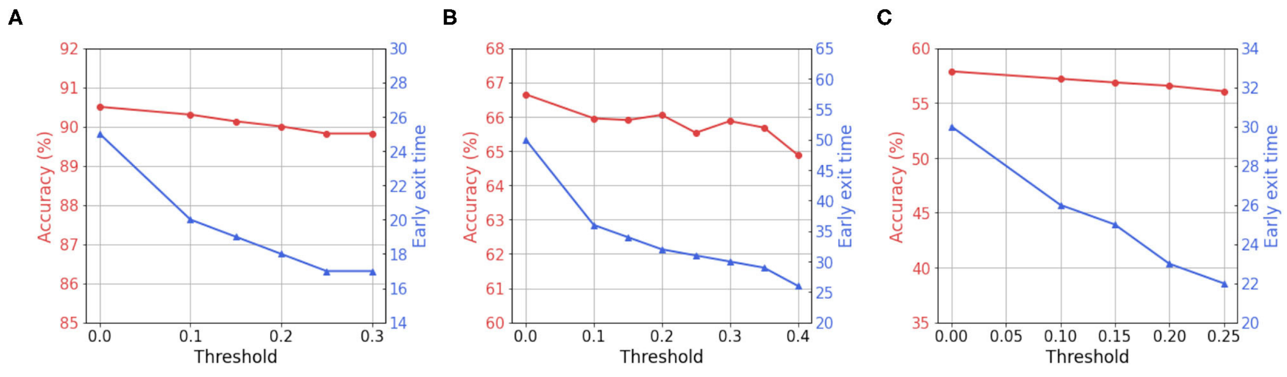


FIGURE 7 | Visualization of accuracy and early exit time with respect to the threshold value for γ. **(A)** CIFAR-10. **(B)** CIFAR-100. **(C)** Tiny-ImageNet.

4.6. Analysis on Early Exit

Recall that we measure the average of γ values in each layer at every time-step, and stop the inference when all γ values in every layer is lower than a predetermined threshold. To further investigate this, we vary the predetermined threshold and show the accuracy and exit time T_{exit} trend. As shown in **Figure 7**, we observe that high threshold enables the networks to infer at earlier time-steps. Although we use less number time-steps during inference, the accuracy drops marginally. This implies that BNTT rarely sends crucial information at the end of spike train (see **Figure 1C**). Note that the temporal evolution of learnable parameter γ with our BNTT allows us to exploit the early exit algorithm that yields a huge advantage in terms of reduced latency at inference. Such strategy has not been proposed or explored in any prior works that have mainly focused on reducing the number of time-steps during training without effectively using temporal statistics.

4.7. Analysis on Robustness

Finally, we highlight the advantage of BNTT in terms of the robustness to noisy input. To investigate the effect of our BNTT for robustness, we evaluate the performance change in the SNNs as we feed in inputs with varying levels of noise. We generate the noisy input by adding Gaussian noise $(0, \sigma^2)$ to the clean input image. From **Figure 8A**, we observe the following: (i) The accuracy of conversion method degrades considerably for $\sigma > 0.4$. (ii) Compared to ANNs, SNNs trained with surrogate gradient back-propagation shows better performance at higher noise intensity. Still, they suffer from large accuracy drops in presence of noisy inputs. (iii) BNTT achieves significantly higher performance than the other methods across all noise intensities. This is because using BNTT decreases the overall number of time-steps which is a crucial contributing factor toward robustness (Sharmin et al., 2020). These results imply that, in addition to low-latency and energy-efficiency, our BNTT method also offers improved robustness for suitably implementing SNNs in a real-world scenario.

In order to further validate the robustness of BNTT, we conduct experiments on adversarial inputs. We use FGSM (Goodfellow et al., 2014) to generate adversarial samples for ANN. For a given image x, we compute the loss function $\mathcal{L}(x, y)$ with the ground truth label y. The objective of FGSM attack is to

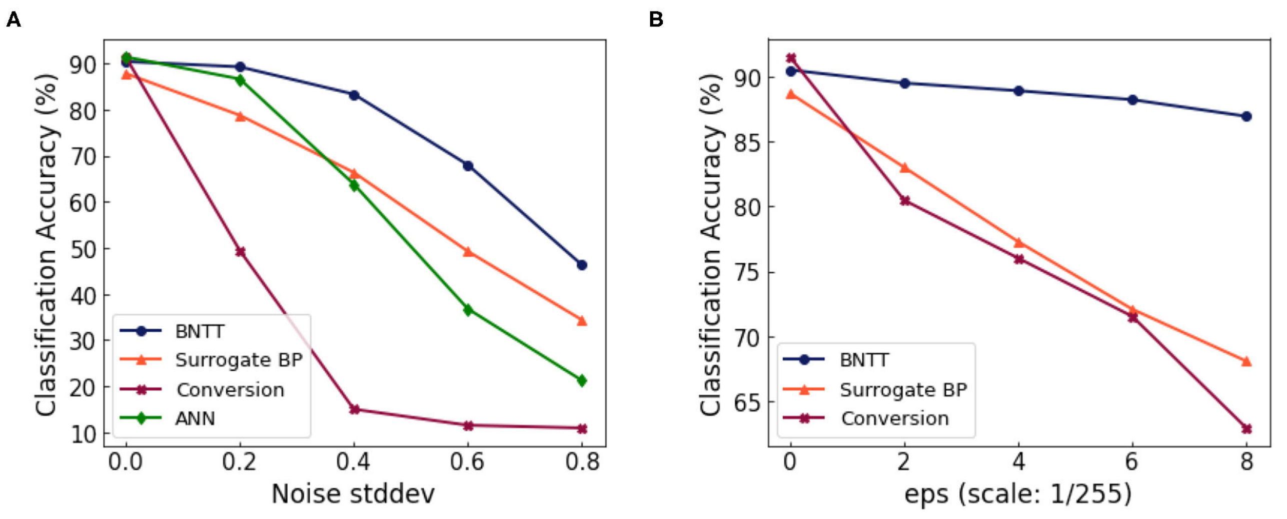


FIGURE 8 | (A) Performance change with respect to the standard deviation of the Gaussian noise. **(B)** Performance change with respect to the attack intensity (ϵ, denoted in x-axis) of the FGSM attack.

change the pixel intensity of the input image that maximizes the cost function:

$$x_{adv} = x + \epsilon \times sign(\nabla_x \mathcal{L}(x, y)). \tag{16}$$

We call x_{adv} as "adversarial sample." Here, ϵ denotes the strength of the attack. To conduct the FGSM attack for SNN, we use the SNN-crafted FGSM method proposed in Sharmin et al. (2020). In **Figure 8B**, we show the classification performance for varying intensities of FGSM attack. The SNN approaches (e.g., BNTT and Surrogate BP) show more robustness than ANN due to the temporal dynamics and stochastic neuronal functionality. We highlight that our proposed BNTT shows much higher robustness compared to others. Thus, we assert that BNTT improves robustness of SNNs in addition to energy efficiency and latency.

4.8. Comparison With Layer Norm

Layer Normalization (LN) (Ba et al., 2016) is an optimization method for recurrent neural networks (RNNs). The authors asserted that directly applying BN layers is hardly applicable since RNNs vary with the length of the input sequence. To this end, an LN layer calculates the mean and the variance for every single layer. As SNNs also take time-sequence data as input, we compare our BNTT with Layer Normalization in **Table 5**. For all experiments, we use a VGG9 architecture. Also, we set a base learning rate to 0.3 and we use step-wise learning rate scheduling as described in section 4.1. The results show that BNTT is more suitable structure to capture the temporal dynamics of Poisson encoded spikes.

TABLE 5 | Comparison with layer normalization on CIFAR-10 dataset.

Method	Acc (%)
Layer normalization (Ba et al., 2016)	75.4
BNTT	90.5

5. CONCLUSION

In this paper, we revisit the batch normalization technique and propose a novel mechanism for training low-latency, energy-efficient, robust, and accurate SNNs from scratch. Our key idea is to investigate the temporal characteristics of Batch Normalization (BN) with time-specific learnable parameters and statistics. Note, BN is known as an effective way of addressing vanishing/exploding gradients problem in ANNs. We discover that optimizing time-dependent learnable parameters γ captures the temporally varying input distribution so that it stabilizes the backward gradients during training and enables better learning of SNN representations. Our experiments reveal interesting benefits of BNTT for temporal early exit during inference as well as sturdy robustness against adversarial attacks. As previous SNN-based BN works (Fang et al., 2020; Ledinauskas et al., 2020; Zheng et al., 2020), this work showcases the importance of incorporating dynamic time-dependent parameters during surrogate gradient-based training to enable large-scale SNN implementations. By showing the importance of addressing the unstable gradient problem in SNN, we suggest future direction for better SNN training. Today, SNNs have few advanced optimization techniques (such as, weight initialization, skip connection that are common in ANN optimization suite) for addressing such issues. Our proposed BNTT can be considered to be one SNN-crafted optimization technique that can relieve the gradient problem, resulting in performance improvement. We hope this work fosters future work on advanced SNN optimization.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

YK and PP conceived the work and contributed to the writing of the manuscript. YK carried out experiments. Both authors contributed to the article and approved the submitted version.

FUNDING

This work was supported in part by the Center for Brain-inspired Computing (C-BRIC) which is a JUMP center sponsored by DARPA and SRC, the National Science Foundation (Grant#1947826), the Technology Innovation Institute, Abu Dhabi and the Amazon Research Award.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.773954/full#supplementary-material

REFERENCES

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst*. 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint* arXiv:1607.06450.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv preprint* arXiv:1803.09574.

Burkitt, A. N. (2006). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biol. Cybern*. 95, 1–19. doi: 10.1007/s00422-006-0068-6

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis*. 113, 54–66. doi: 10.1007/s11263-014-0788-3

Comsa, I. M., Fischbacher, T., Potempa, K., Gesmundo, A., Versari, L., and Alakuijala, J. (2020). "Temporal coding in spiking neural networks with alpha synaptic function," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Barcelona: IEEE), 8529–8533.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dayan, P., and Abbott, L. F. (2001). *Theoretical Neuroscience.* vol. 806.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "Imagenet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL: IEEE), 248–255.

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci*. 9:99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8.

Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., et al. (2021). Training spiking neural networks using lessons from deep learning. *arXiv preprint* arXiv:2109.12894.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. (2020). Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *arXiv preprint* arXiv:2007.05785.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint* arXiv:1412.6572.

Han, B., Srinivasan, G., and Roy, K. (2020). "Rmp-snn: residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE), 13558–13567.

Ioffe, S., and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv preprint* arXiv:1502.03167.

Krizhevsky, A., and Hinton, G. (2009). Learning multiple layers of features from tiny images.

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. (2016). Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell*. 39, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint* arXiv:1504.00941.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Ledinauskas, E., Ruseckas, J., Juršėnas, A., and Buračas, G. (2020). Training deep spiking neural networks. *arXiv preprint* arXiv:2006.04436.

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci*. 14:119. doi: 10.3389/fnins.2020.00119

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci*. 10:508. doi: 10.3389/fnins.2016.00508

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Front. Neurosci*. 11:309. doi: 10.3389/fnins.2017.00309

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *IEEE Signal. Process. Mag*. 36, 61–63. doi: 10.1109/MSP.2019.2931595

Orchard, G., Meyer, C., Etienne-Cummings, R., Posch, C., Thakor, N., and Benosman, R. (2015). Hfirst: a temporal approach to object recognition. *IEEE Trans. Pattern Anal. Mach. Intell*. 37, 2028–2040. doi: 10.1109/TPAMI.2015.2392947

Panda, P., Aketi, S. A., and Roy, K. (2020). Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Front. Neurosci*. 14:653. doi: 10.3389/fnins.2020.00653

Panda, P., Sengupta, A., and Roy, K. (2016). "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)* (Dresden: IEEE), 475–480.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in pytorch," in *NIPS-W*. Long Beach, CA.

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint* arXiv:2005.01807.

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11:682. doi: 10.3389/fnins.2017.00682

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). "How does batch normalization help optimization?" in *Advances in Neural Information Processing Systems* (Montreal, CA), 2483–2493.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: Vgg and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095

Sharmin, S., Rathi, N., Panda, P., and Roy, K. (2020). Inherent adversarial robustness of deep spiking neural networks: effects of discrete input encoding and non-linear activations. *arXiv preprint* arXiv:2003.10399. doi: 10.1007/978-3-030-58526-6_24

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. (2018). "Hats: histograms of averaged time surfaces for robust event-based object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT: IEEE), 1731–1740.

Teerapittayanon, S., McDanel, B., and Kung, H.-T. (2016). "Branchynet: fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)* (Cancun: IEEE), 2464–2469.

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). Direct training for spiking neural networks: faster, larger, better. *Proc. AAAI Conf. Artif. Intell.* 33, 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. (2020). Going deeper with directly-trained larger spiking neural networks. *arXiv preprint* arXiv:2011.05280.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2021 Kim and Panda. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

ORIGINAL RESEARCH
published: 11 January 2022
doi: 10.3389/fnins.2021.759807

Accelerating DNN Training Through Selective Localized Learning

Sarada Krithivasan[1]*, *Sanchari Sen*[2], *Swagath Venkataramani*[2] *and Anand Raghunathan*[1]

[1] Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States, [2] IBM Research, Yorktown Heights, NY, United States

Training Deep Neural Networks (DNNs) places immense compute requirements on the underlying hardware platforms, expending large amounts of time and energy. We propose $\mathrm{LoCal{+}SGD}$, a new algorithmic approach to accelerate DNN training by *selectively combining localized or Hebbian learning within a Stochastic Gradient Descent (SGD) based training framework*. Back-propagation is a computationally expensive process that requires 2 Generalized Matrix Multiply (GEMM) operations to compute the error and weight gradients for each layer. We alleviate this by selectively updating some layers' weights using localized learning rules that require only 1 GEMM operation per layer. Further, since localized weight updates are performed during the forward pass itself, the layer activations for such layers do not need to be stored until the backward pass, resulting in a reduced memory footprint. Localized updates can substantially boost training speed, but need to be used judiciously in order to preserve accuracy and convergence. We address this challenge through a *Learning Mode Selection Algorithm*, which gradually selects and moves layers to localized learning as training progresses. Specifically, for each epoch, the algorithm identifies a *Localized→SGD* transition layer that delineates the network into two regions. Layers before the transition layer use localized updates, while the transition layer and later layers use gradient-based updates. We propose both static and dynamic approaches to the design of the learning mode selection algorithm. The static algorithm utilizes a pre-defined scheduler function to identify the position of the transition layer, while the dynamic algorithm analyzes the dynamics of the weight updates made to the transition layer to determine how the boundary between SGD and localized updates is shifted in future epochs. We also propose a low-cost weak supervision mechanism that controls the learning rate of localized updates based on the overall training loss. We applied $\mathrm{LoCal{+}SGD}$ to 8 image recognition CNNs (including ResNet50 and MobileNetV2) across 3 datasets (Cifar10, Cifar100, and ImageNet). Our measurements on an Nvidia GTX 1080Ti GPU demonstrate upto 1.5× improvement in end-to-end training time with ∼0.5% loss in Top-1 classification accuracy.

Keywords: Deep Neural Networks (DNNs), localized learning, runtime efficiency, graphics process unit (GPU), stochastic gradient decent algorithm

OPEN ACCESS

Edited by:
*Bipin Rajendran,
King's College London,
United Kingdom*

Reviewed by:
*Enea Ceolini,
Leiden University, Netherlands
Bo Yuan,
Rutgers, The State University of New Jersey, United States*

***Correspondence:**
*Sarada Krithivasan
skrithiv@purdue.edu*

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *17 August 2021*
Accepted: *14 October 2021*
Published: *11 January 2022*

Citation:
Krithivasan S, Sen S, Venkataramani S and Raghunathan A (2022) Accelerating DNN Training Through Selective Localized Learning. Front. Neurosci. 15:759807. doi: 10.3389/fnins.2021.759807

1. INTRODUCTION

Deep Neural Networks (DNNs) have achieved continued success in many machine learning tasks involving images (Krizhevsky et al., 2017), videos (Ng et al., 2015), text (Zhou et al., 2015), and natural language (Goldberg and Hirst, 2017). However, training state-of-the-art DNN models is highly computationally expensive, often requiring exa-FLOPs of compute as the models are complex and need to be trained using large datasets. Despite rapid improvements in the capabilities of GPUs and the advent of specialized accelerators, training state-of-the-art models using current platforms is still quite expensive and often takes days to weeks. In this work, we aim to reduce the computational complexity of DNN training through a new algorithmic approach called LoCal+SGD[1], which alleviates the key performance bottlenecks in Stochastic Gradient Descent (SGD) through *selective use of localized learning*.

Computational Bottlenecks in DNN Training. DNNs are trained in a supervised manner using gradient-descent based cost minimization techniques such as SGD (Bottou, 2010) or Adam (Kingma and Ba, 2015). The training inputs, typically grouped into minibatches, are iteratively forward propagated (*FP*) and back propagated (*BP*) through the DNN layers to compute weight updates that push the network parameters in the direction that decreases the overall classification loss. Back-propagation is computationally expensive, accounting for 65–75% of the total training time on GPUs. This is attributed to two key factors: (i) *BP* involves 2 Generalized Matrix Multiply (GEMM) operations per layer, one to propagate the error and the other to compute the weight gradients, and (ii) when training on distributed systems using data/model parallelism (Dean et al., 2012; Krizhevsky et al., 2012), aggregation of weight gradients/errors across devices incurs significant communication overhead.

Prior Efforts on Efficient DNN Training. Prior research efforts to improve DNN training time can be grouped into a few directions. One group of efforts enable larger scales of parallelism in DNN training through learning rate tuning (Goyal et al., 2017; You et al., 2017a,b) and asynchronous weight updates (Dean et al., 2012). Another class of efforts employ importance-based sample selection during training, wherein "easier" training samples are selectively discarded to improve runtime (Jiang et al., 2019; Zhang et al., 2019). Finally, model quantization (Sun et al., 2019) and pruning (Lym et al., 2019) can lead to significant runtime benefits during training by enabling the use of reduced-bitwidth processing elements.

LoCal+SGD**: Combining SGD with Localized Learning.** Complementary to the aforementioned efforts, we propose a new approach, LoCal+SGD, to alleviate the performance bottlenecks in DNN training, while preserving model accuracy. Our hybrid approach combines Hebbian or localized learning (Hebb, 1949) with SGD by selectively applying it in specific layers and epochs. Localized learning rules (Hebb, 1949; Oja, 1982; Zhong, 2005) utilize a single feed-forward weight update to learn the feature representations, eschewing the *BP* step. Careful formulation of the localized learning rule can result in substantial computation savings compared to SGD. Further, it also reduces memory footprint as activations from *FP* need not be retained until *BP*. The reduction in memory footprint can in turn allow increasing the batch size during training, which leads to further runtime savings due to better compute utilization and reduced communication costs. It is worth noting that localized learning has been extensively explored in the context of unsupervised learning (van den Oord et al., 2018; Hénaff et al., 2019; Chen et al., 2020). Further, the formulation of new neuro-inspired learning rules remains an active area of research (Lee et al., 2015; Nøkland, 2016). Our work is orthogonal to such efforts and represents a new application of localized learning in a fully supervised context, wherein we selectively employ it within an SGD framework to achieve computational savings.

Preserving model accuracy and convergence with LoCal+SGD requires localized updates to be applied judiciously i.e., only to selected layers in certain epochs. We address this challenge through the design of a *learning mode selection algorithm*. At the start training, the algorithm initializes the learning mode of all layers to SGD. As training progresses, it identifies layers that will be moved to localized learning. Specifically, for each epoch, the algorithm identifies a *Localized→SGD* transition layer, which delineates the network into two regions. Layers before the transition layer use localized updates, while subsequent layers use gradient-based updates. This allows *BP* to stop at the transition layer, as layers before it have no need for the back-propagated errors. We explore both static and dynamic learning mode selection algorithms. The static algorithm utilizes a suitably chosen pre-defined function to determine the position of the transition layer every epoch. The dynamic algorithm analyzes the dynamics of the weight updates of the *Localized→SGD* transition layer in deciding the new position of the boundary. Further, we provide weak supervision by modulating the learning rate of locally updated layers based on the overall training loss.

To the best of our knowledge, LoCal+SGD is the first effort that combines localized learning (an unsupervised learning technique) within a supervised SGD context to reduced computational costs while maintaining classification accuracy. Across 8 image recognition CNNs (including ResNet50 and MobileNet) and 3 datasets (Cifar10, Cifar100, and ImageNet), we demonstrate that LoCal+SGD achieves up to 1.5× improvement in training time with ∼0.5% Top-1 accuracy loss on a Nvidia GTX 1080Ti GPU.

2. MATERIALS AND METHODS: LOCAL+SGD

The key idea in LoCal+SGD is to apply localized learning to selected layers and epochs during DNN training to reduce the overall training time with minimal loss in accuracy. The following design choices are critical to the effectiveness of LoCal+SGD:

- **Localized Learning Rule Formulation.** Eliminating *BP* would be of little help if it is replaced with an equally expensive learning rule. It is critical to choose a computationally efficient

[1]In addition to combining localized and SGD based learning, LoCal+SGD is Low-Calorie SGD or SGD with reduced computational requirements.

rule that still enables learning in the contexts where it is invoked.

- **Learning Mode Selection.** It is well known that universal use of localized learning rules results in an accuracy that is much lower than SGD. The key is to figure out when (which epochs) and where (which layers) to apply localized learning to best balance efficiency and accuracy. We refer to this as learning mode selection.
- **Weak Supervision.** Since we are operating within an overall supervised learning context where some layers are using global information, it is natural to ask whether such information can be used in a lightweight manner to improve the efficacy of localized learning. To this end, we propose a weak supervision technique, which modulates the learning rates of localized learning based on the overall classification loss.

In the following sub-sections, we describe how we address these design choices in greater detail.

2.1. Efficient Localized Learning

There has been growing interest toward the design of biologically plausible learning algorithms, in part to address the high computational requirements of stochastic gradient descent and in part to realize bio-plausible artificial intelligence systems. Learning rules such as feedback alignment Nøkland (2016) resolve the weight transport problem (Liao et al., 2015) by allowing for asymmetry in the weight values during forward and backward propagation. Similarly, target propagation (Lee et al., 2015) encourages neural activity to reach desired target activations evaluated during forward propagation, instead of utilizing loss gradients. Other learning rules such as equilibrium propagation (Scellier and Bengio, 2017) update the weights by evaluating gradients of locally defined objective functions, thereby avoiding gradient propagation across the network. However, many of these bio-plausible learning algorithms end up being computationally more expensive than SGD, such as feedback alignment (Nøkland, 2016). As the focus of our work is primarily on improving training runtime while achieving state-of-the-art accuracies, we propose the selective use of computationally lightweight localized learning rules in conjunction with SGD.

Localized learning has been extensively explored in the context of unsupervised learning, demonstrating success on small (<= 3 layer) networks using relatively simpler datasets (e.g., MNIST, Cifar-10) (Krizhevsky et al., 2009; Deng, 2012) with an accuracy gap that is yet to be bridged on larger datasets (e.g., ResNet50 or MobileNetV2 on ImageNet; Deng et al., 2009). First proposed in Hebb (1949), the key intuition behind localized learning rules is to encourage correlations between neurons that have similar activation patterns. Equation (1) depicts the Hebbian weight update proposed in Hebb (1949), for a synapse with weight W, connecting a pair of input and output neurons whose activation values are represented by x and y, respectively, with η as the learning rate.

$$\triangle W = \eta \cdot x \cdot y \tag{1}$$

Considerable research has gone into evolving this equation over the years to improve the performance of localized learning (Oja, 1982; Zhong, 2005). However, many of the proposed rules are computationally complex, or are difficult to parallelize on modern hardware platforms such as GPUs. Since our primary goal is improving DNN training time, we adopt the computationally simple localized learning rule presented in Equation (1).

Note that the learning rule in Equation (1) assumes a distinct synapse between each input and output neuron pair. While its application to fully-connected (fc) layers is straightforward, we need to consider the sharing of weights between neuron pairs in convolutional (conv) layers. For updating a shared weight of a conv layer, we calculate the individual updates due to each pair of pre- and post-synaptic neurons sharing the weight and sum all such updates. This essentially reduces to a convolution operation between the input and output activations of the layer and can be expressed by Equation (3) in **Figure 1**. For further computational efficiency improvement, unlike Equation (1), we consider the pre-activation-function values of the outputs i.e., z_l instead of their post activation value a_l. Further, we normalize the localized update values as shown in Equation (4) of **Figure 1**, as it was observed to achieve better convergence in practice.

Overall, we utilize Equations (3) and (4) from **Figure 1** to perform the weight updates in all layers that are earlier than the *Localized→SGD* transition layer during a certain epoch. All other layers continue to be updated using SGD-based *BP*, expressed by Equations (5–7) in **Figure 1**. SGD updates are applied to batch-normalization layers present after the *Localized→SGD* transition layer, and are otherwise skipped. Clearly, Equation (3) has the same computational complexity as Equation (6) of SGD-based

$\underline{W_l}$: Filter Weight of Layer L
$\underline{z_l}$: Pre-Activation o/p of Layer L
$\underline{a_l}$: Post-Activation o/p of Layer L
$\underline{\delta_l}$: Error at Layer L
$\underline{\eta}$: Learning Rate

Stage	Localized Updates	SGD-Based Updates
FP	$z_l = GEMM(a_{l-1}, W_l)$ (2)	
Update Stage	$\Delta W_l = GEMM(a_{l-1}, z_l)$ (3) $W_l = W_l + \eta \dfrac{\Delta W_l}{\lVert \Delta W_l \rVert}$ (4)	$\delta_l = GEMM(\delta_{l+1}, W_l)$ (5) $\Delta W_l = GEMM(a_{l-1}, \delta_l)$ (6) $W_l = W_l + \eta \Delta W_l$ (7)

NOTE: GEMM includes operations in convolutional and fully-connected layers

FIGURE 1 | Comparing localized updates and SGD-based *BP*.

BP for conv and fc layers. Thus, from **Figure 1**, we can directly infer that our localized learning rule will be considerably faster than SGD-based *BP*. In practice, we measured this improvement to be more than 2× on a NVIDIA GTX 1080Ti GPU for the ImageNet-ResNet50 benchmark, across all conv and fc layers. In addition, localized learning also reduces the memory footprint of SGD-based *BP*. This is because DNN software frameworks commonly store all activation values computed during *FP* for use during SGD-based *BP* [a_{l-1} in Equation (6) of **Figure 1**]. In contrast, the localized update for a layer can be performed as soon as the *FP* through the layer is complete. The activation tensor a_l of layer L can be discarded or over-written as soon as *FP* proceeds to the next layer in the network, thereby freeing up a significant portion of on-device memory during training. In turn, this can allow larger minibatch sizes to be accommodated on a given hardware platform, when the localized updates are applied on a sufficient number of layers.

2.2. Learning Mode Selection Algorithm

The compute benefits of localized learning come at the cost of potential loss in accuracy with respect to SGD. To address this challenge, we propose a learning mode selection algorithm to judiciously choose when and where to apply localized learning. The algorithm identifies the learning mode of each layer in every epoch to create a favorable tradeoff between training time and accuracy.

Before describing the proposed learning mode selection algorithms, we first study the effects of different spatio-temporal patterns of localized learning on the computational efficiency and accuracy of a neural network. We specifically investigate whether localized learning is more suitable for specific layers in the network and specific phases in the training process.

Impact on runtime: We first analyze the impact of spatial patterns, i.e., whether applying localized learning to specific layers in the network results in better runtime. In a particular epoch, if a convolutional layer L, updated with SGD precedes a convolutional layer K, that is updated locally, calculating the SGD-based error gradients of Layer L, i.e., δ_L, requires error propagation through the locally updated layer K. From a compute efficiency perspective, the benefits of using localized-updates in layer K vanish. Thus, it makes sense to partition the network into two regions—a prefix (set of initial layers) that are updated using localized learning, followed by layers that are updated with SGD. In such a setting, SGD-based *BP* is simply stopped at the junction of the two regions. Naturally, the compute benefits increase when the number of locally updated layers are higher and thus the boundary, which we refer to as the *Localized→SGD* transition layer, is moved deeper into the network.

The impact of different temporal patterns on runtime efficiency is quite straightforward, with higher number of locally updated epochs leading to proportionally higher benefits. Further, as the compute complexity of localized updates is constant across different epochs, these benefits are agnostic of the specific epochs in which localized learning is utilized.

Impact on accuracy: To analyze the impact on accuracy, we first examine the nature of features learnt by different layers trained by SGD. It is commonly accepted that the initial layers of a network perform feature extraction (Agrawal et al., 2014), while later layers aid in the classification process. As localized learning demonstrates better performance for feature extraction, applying it more aggressively, i.e., for higher number of epochs, in the initial layers has a much smaller impact accuracy. For later layers in the network, the number of localized learning epochs should be progressively reduced to preserve accuracy.

Overall, based on the impact of localized learning on both runtime and accuracy, we find that a good learning mode selection algorithm should favor application of localized learning to a contiguous group of initial layers, while employing fewer localized learning epochs in later layers. We impose an additional constraint in order to ensure stability and convergence of training. We allow each layer to transition from one learning mode to another at most once during the entire training process. We empirically observe that utilizing SGD as the initial learning mode allows the network to achieve a higher accuracy than utilizing localized learning as the initial mode. In other words, SGD provides a better initialization point for the parameters of all layers, and the subsequent use of localized updates enables the training to converge with good accuracy. Taken together, the aforementioned constraints imply that if a layer L switches from the SGD learning to localized learning at epoch E, layer $L + 1$ may switch at an epoch $E^{'} >= E$. This is depicted graphically in **Figure 2**, where the *Localized→SGD* transition layer must move toward the right in successive epochs.

Static Learning Mode Selection Algorithm: In a static learning mode selection algorithm, the *Localized→SGD* transition layer is computed using a pre-determined schedule (**Figure 3**). Many functions can be used to impose the desired schedule, wherein the number of locally updated layers increases monotonically with the epoch index. These functions must be chosen such that the schedule is neither too conservative in the application of localized updates (which may lead to sub-optimal compute and memory benefits), nor too aggressive (which may lead to a large drop in accuracy). In our experiments, we observed that using a quadratic function provides a good tradeoff between efficiency and accuracy. We illustrate this in **Figure 4**, wherein we compare the performance of quadratic, exponential and linear schedules for the Cifar10-ResNet18 benchmark. The proposed linear, quadratic and exponential scheduling functions that specifies the index of the *Localized→SGD* transition layer $N_{l,E}$ at every epoch E are expressed as:

$$N = \lfloor max(c_1 \cdot E_{max} + c_2 \cdot E, 0) \rfloor \tag{2}$$

$$N = \lfloor max(c_1 - c_2 \cdot (E - E_{max})^2, 0) \rfloor \tag{3}$$

$$N = \lfloor max((e^{c2 \cdot E} - c_1 \cdot E_{max}), 0) \rfloor \tag{4}$$

where c_1 and c_2 are hyper-parameters, and E_{max} is the total number of training epochs. As shown in **Figure 3** for quadratic schedules, c_1 controls the maximum number of layers that are updated locally across the training process, while c_2 controls the epoch at which localized updates begin. The values of c_1 and c_2 are determined with the aim of maximizing the area

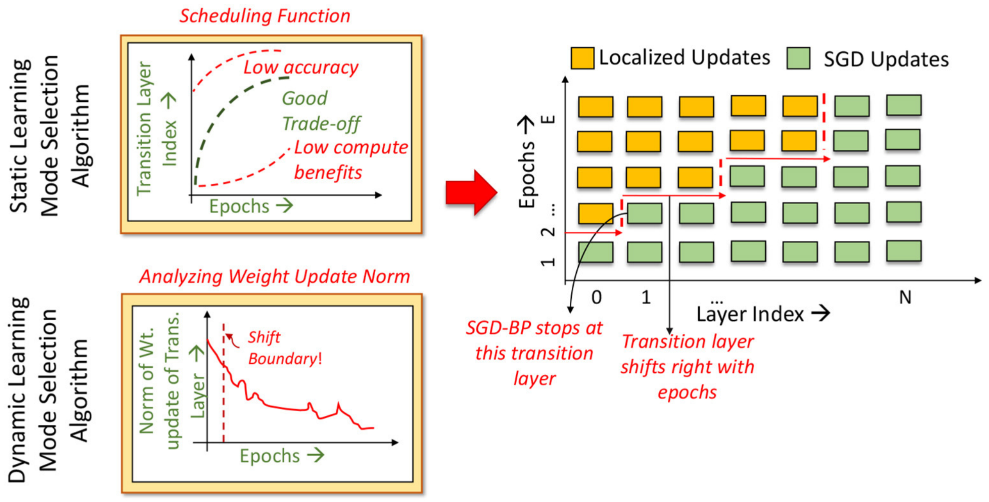

Static Learning Mode Selection Algorithm
Scheduling Function
Transition Layer Index →
Low accuracy
Good Trade-off
Low compute benefits
Epochs →
Dynamic Learning Mode Selection Algorithm
Analyzing Weight Update Norm
Norm of Wt. update of Trans. Layer →
Shift Boundary!
Epochs →
Localized Updates
SGD Updates
Epochs →
Layer Index →
SGD-BP stops at this transition layer
Transition layer shifts right with epochs

FIGURE 2 | Overview of the learning mode selection algorithm.

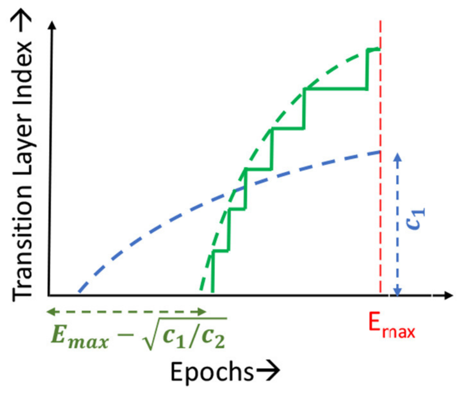


FIGURE 3 | Transition layer schedules.

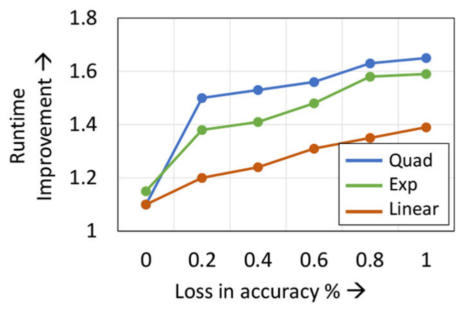


FIGURE 4 | Impact of different scheduling functions on Cifar10-ResNet18 training.

under the curve, i.e., employing localized updates as many layers and epochs as possible, while maintaining a competitive classification accuracy.

Algorithm 1 Learning Mode Selection Algorithm.

```
Input: $T_E$ (Index of the transition layer at epoch E), $||\triangle W_E||$ ($L_2$ norm of the weight update of the transition layer at epoch E), $L_{shift}$ (number of layers to shift boundary)
Output: $T_{E+1}$ (Index of the transition layer at epoch E+1)
1: if $||\triangle W_E|| <= \alpha \cdot W_{Avg}$
2:     $T_{E+1} = T_E + L_{shift}$
3: else
4:     $T_{E+1} = T_E$
```

Dynamic Learning Mode Selection Algorithm: As shown in **Figure 4**, the efficacy of the learning mode selection algorithm is dependent on the scheduling function chosen. Given the long training runtimes, identifying the optimal schedule for every network is a cumbersome process, and it is beneficial if the learning mode selection algorithm is free of hyper-parameters. To that end, we propose a dynamic learning mode selection algorithm that automatically identifies the position of the boundary every epoch.

The dynamic learning mode selection algorithm, described in **Algorithm 1**, analyzes the changes in the L_2 norm of the SGD weight update of the *Localized*→*SGD* transition layer, and determines whether the boundary can be shifted deeper into the network for the next epoch. The exponentially running average of the norm update, W_{avg}, is first evaluated (line 1). If the norm of the weight update in epoch E is significantly smaller than W_{avg}, i.e., less than some fraction α, the boundary is shifted right by L_{shift} layers (line 2). Else, the boundary remains stationary (line 4). The rationale for this criterion is that sustained high magnitudes of SGD weight updates in the transition layer indicate that they are potentially critical to accuracy, in which case the transition layer must continue being updated with SGD.

Naturally, α and L_{shift} provide trade-offs between accuracy and runtime savings—higher values of either quantity result in aggressive applications of localized updates and hence better runtimes, but at the cost of degradations in accuracy. Our experiments suggest that values of α between 0.1 and 0.5, and L_{shift} between 10 and 15%, provide good performance across all the benchmarks studied. In section 3, we explore this trade-off space in greater detail.

To summarize, we propose static and dynamic learning mode selection algorithms that help identify the position of the transition layer for every epoch. Each algorithm comes with its own benefits—static algorithms can be hand-tuned to provide superior performance, but at the cost of additional effort involved in tuning the hyperparameters.

2.3. Weak Supervision

To further bridge the accuracy gap between our hybrid and end-to-end SGD training, we introduce weak supervision in the locally updated layers. Unlike the SGD, the localized learning rules described thus far do not take advantage of the information provided by supervision, i.e., the classification error evaluated at the output. We incorporate this information through a low-cost weak supervision scheme that consists of a single signal sent to all layers updated locally in a particular epoch. This feedback is derived from the classification loss observed over past few epochs. The weak supervision scheme is described in **Algorithm 2**.

The key principle behind the weak supervision scheme is to control the learning rates of the locally updated layers based on the rate at which the overall classification loss changes. For example, if the overall classification loss has increased across consecutive epochs, we reverse the direction of the updates (line 3) in the next epoch. In contrast, the update direction is maintained if the overall loss is decreasing (line 5). We find that this weak supervision provides better accuracy results than other learning rate modulation techniques for the locally updated layers such as Adam or momentum-based updates.

We would like to highlight that traditional SGD provides fine-grained supervision and involves evaluating the error gradients for every neuron in the network. In contrast, the proposed weak supervision scheme provides coarse-grained supervision by forcing all weights to re-use the same loss information. Overall, our weak supervision scheme is not developed with the intent to compete with SGD updates, but is rather a simple, approximate and low-cost technique that brings the final accuracy of LoCal+SGD closer to end-to-end SGD training.

Algorithm 2 Weak Supervision Scheme.

```
Input: $L_i$ (Overall classification loss at epoch $i$), $lr_L$ (original learning rate of layer $L$)
Output: $W_L$ (Weight update of layer $L$)
1: $\triangle W_L = conv(a_{l-1}, z_l)$
2: if $L_{i-1} < L_i$
3:     $W_L = W_L - lr_L \cdot \frac{\triangle W_L}{||\triangle W_L||}$
4: else
5:     $W_L = W_L + lr_L \cdot \frac{\triangle W_L}{||\triangle W_L||}$
```

3. RESULTS AND DISCUSSION

In this section, we present the results of our experiments highlighting the compute benefits achieved by LoCal+SGD. We evaluate the benefits across a suite of 8 image-recognition DNNs across 3 datasets. We consider the ResNet18 (He et al., 2015) and VGG13 (Simonyan and Zisserman, 2015) networks for the Cifar10 (Krizhevsky et al., 2009) and Cifar100 (Krizhevsky et al., 2009) datasets; and the ResNet34, ResNet50 (He et al., 2015) and MobileNetV2 (Sandler et al., 2018) networks for the ImageNet dataset (Deng et al., 2009).

3.1. Experimental Setup

This subsection describes the experimental setup used for realizing the baseline and proposed LoCal+SGD training schemes. We conduct our experiments on the complete training and test datasets of each benchmark, using the PyTorch (Paszke et al., 2019) framework. All experiments are conducted on Nvidia GTX 1080Ti GPUs with the batch size set to 64 per GPU, unless otherwise mentioned.

Baseline: We consider end-to-end SGD training as the baseline in our experiments. The hyper-parameters used in SGD training of each of the benchmarks are described below.

ImageNet: For experiments in section 3.2 we utilize a batch-size of 64 per GPU, for all benchmarks. For the ResNet50 and ResNet34 benchmarks the initial learning rate set to 0.025. The learning rate is decreased by 0.1 every 30 epochs, for a total training duration of 90 epochs, and the weight decay is $4e-5$. The MobileNetV2 benchmark utilizes an initial learning rate of 0.0125. We use a cosine learning rate decay schedule, as in Li et al. (2019) for 150 epochs. The weight decay is set to $4e-5$. Both benchmarks use an input size of 224*224*3.

For the experiments in section 3.3, the total batch-size at epoch 1 is 256 (64*4), with the initial learning rate set to 0.1 for the ResNet benchmarks and 0.05 for the MobileNetV2 benchmark. All other parameters remain the same.

Cifar10 and Cifar100: All Cifar10 and Cifar100 experiments utilize a batch-size of 64. The Cifar10 benchmarks are trained with an initial learning rate of 0.05 that is decayed by 0.1 every 10 epochs, across 90 epochs. The initial learning rate of the Cifar100 benchmarks is 0.025 and decayed by 0.5 every 20 epochs, for 150 epochs in total. The weight decay is set to $5e-4$. Both benchmarks utilize an input size of 32*32*3.

LoCal+SGD: In the proposed LoCal+SGD training scheme, the layers updated with SGD are trained with the same hyper-parameters used in the baseline implementation. Further, LoCal+SGD training is conducted using the same number of epochs as baseline SGD training. When a layer is updated locally, the initial learning rate is 0.01 and is decayed by factors of 2 and 10 every 30 epochs for the Cifar and the ImageNet benchmarks, respectively. In all experiments, the α parameter is set to 0.8. We measure the accuracy and runtime of the proposed scheme for the same number of training epochs as the baseline implementations. Further, we utilize the same random seed to initialize the weights of the network when comparing the performance of LoCal+SGD against the baseline. Speed-up and accuracy results are averaged over 10 runs for each benchmark.

3.2. Single GPU Execution Time Benefits

ImageNet: Table 1 presents the performance of the baseline (end-to-end SGD training) and the proposed LoCal+SGD algorithm (both static and dynamic versions) on the ImageNet benchmarks in terms of the Top-1 classification error and runtime observed on a single GPU. For all benchmarks listed here, the static and dynamic versions of LoCal+SGD apply localized updates for nearly 50–60% of the layers. Further, the LoCal+SGD algorithms achieve upto ∼1.4× reduction in runtime compared to the baseline, while sacrificing <0.5% loss in Top-1 accuracy. The static LoCal+SGD algorithm exhibits slightly superior runtime performance for similar accuracies compared to the dynamic algorithm. However, as noted earlier, the dynamic algorithm eliminates the effort required to identify an optimal scheduling function.

Table 1 also compares the performance of LoCal+SGD against existing research efforts designed to improve training efficiency. We perform this analysis against two efforts, namely (i) Training with stochastic depth (Huang et al., 2016) and (ii) Structured Pruning during Training (Lym et al., 2019). Training with stochastic depth, as the name suggests, stochastically bypasses residual blocks by propagating input activations/error gradients via identity or downsampling transformations, resulting in improved training time. However, the approach is targeted toward extremely deep networks and as seen in **Table 1**, it incurs a noticeable accuracy loss on networks such as ResNet34, ResNet50 and MobileNetV2. Compared to training with stochastic depth, our proposal clearly achieves better accuracy as well as training runtime benefits. The key principle behind the pruning during training approach is to reduce the size of the weight and activation tensors in a structured manner during training, thereby providing speed-ups on GPU/TPU platforms. However, on complex benchmarks such as ResNet50, such techniques achieve speed-ups at the cost of significant drop in accuracy (∼ 1.5%). To further demonstrate the utility of localized updates in our approach, we consider a third technique, wherein layers selected to be updated locally for a given epoch are instead frozen, i.e., the parameters are held fixed during that epoch. While this achieves better runtime savings, it incurs considerably higher loss in accuracy, further underscoring the benefits of LoCal+SGD.

In **Figure 5**, we depict the validation accuracy curves for the ResNet50 and MobileNetV2 benchmarks trained with LoCal+SGD and SGD, normalized to SGD training runtime. For the sake of brevity, we have presented the curves when using the dynamic learning mode selection algorithm. As can be seen, after a few epochs have passed since localized updates began, LoCal+SGD achieves better accuracies for the same runtime.

CIFAR-10 and CIFAR-100: Table 2 presents the accuracy and corresponding compute benefits of the baseline and the proposed technique, as well as training with stochastic depth and layer freezing, for the CIFAR-10 and CIFAR-100 datasets. Stochastic depth is applicable only to residual blocks and is hence not considered for the VGG-13 network. Across benchmarks, we observe upto a 1.51× improvement in training runtime. Compared to the ImageNet benchmarks, LoCal+SGD applies localized updates more aggressively in the CIFAR-10 and CIFAR-100 benchmarks i.e., more layers are updated locally for a higher number of epochs. This leads to superior compute benefits on these benchmarks.

In **Table 3** we compare the final accuracy obtained by LoCal+SGD against the baseline for the same time budget across all our benchmarks. We note that the time budget considered is the time taken by LoCal+SGD to complete all epochs of training. Clearly, within the same time budget LoCal+SGD achieves better accuracy than baseline SGD.

TABLE 1 | ImageNet.

Network	Training strategy	Top-1 error (%)	Speed-up
ResNet34	Baseline SGD	26.6	1×
	LoCal+SGD (Static)	**27**	**1.34×**
	LoCal+SGD (Dynamic)	**27.04**	**1.26×**
	Training with Stochastic depth	27.89	1.13×
	Freezing layers during training	27.32	1.36×
ResNet50	Baseline SGD	24.02	1×
	LoCal+SGD (Static)	**24.51**	**1.42×**
	LoCal+SGD (Dynamic)	**24.45**	**1.37×**
	Training with Stochastic depth	26.76	1.08×
	Pruning during training	24.89	1.32×
	Freezing layers during training	24.84	1.49×
MobileNetV2	Baseline SGD	28.41	1×
	LoCal+SGD (Static)	**28.90**	**1.32×**
	LoCal+SGD (Dynamic)	**28.98**	**1.27×**
	Training with Stochastic depth	30.53	1.17×
	Freezing layers during training	29.31	1.54×

All experimental results pertaining to LoCal+SGD are highlighted in bold.

3.3. Execution Time Benefits for Multi-GPU Training

We analyze the memory footprint of the ResNet50 network when trained with LoCal+SGD on the ImageNet dataset (**Figure 6**). Training first commences with all layers updated with SGD, resulting in a high memory footprint. Due to the 10 GB capacity of the chosen GPU, the mini-batch size is limited to 64 per GPU. As the *Localized→SGD* transition layer progresses across the network, the memory footprint required also gradually reduces across epochs. We take advantage of this reduction in memory footprint in the context of distributed training using 4 GPUs with data parallelism. Specifically, we extract additional runtime benefits by increasing the batch size on each GPU, which reduces the frequency of gradient aggregation between devices and alleviates the communication overhead. At epoch 33, the memory footprint per GPU reduces to <5 GB, allowing training with an increased mini-batch size of 128 per GPU from epoch 33 onwards. As seen in **Table 4**, the doubling of the batch-size provides an additional 6% improvement in total training time. We note that other training techniques such as training

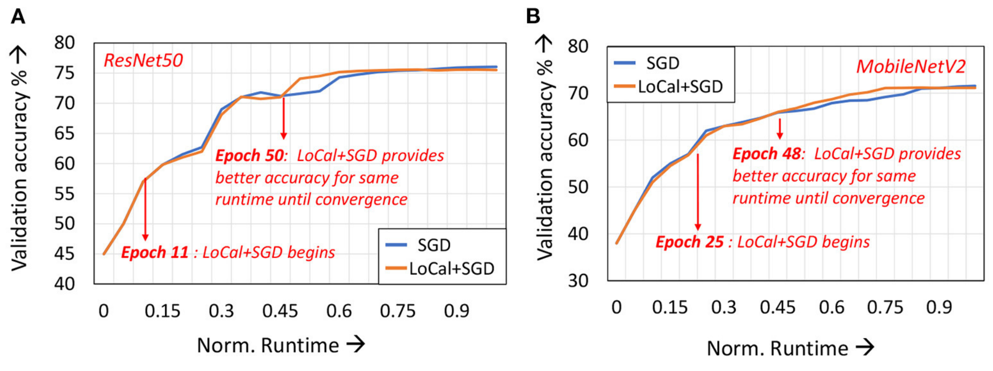


FIGURE 5 | Validation accuracies across training runtime for **(A)** ResNet50 and **(B)** MobileNetV2.

TABLE 2 | Cifar10 and Cifar100.

Network (Dataset)	Training strategy	Top-1 err. (%)	Speed-up
ResNet18	Baseline SGD	6.06	1×
(Cifar10)	LoCal+SGD (Static)	**6.17**	**1.53×**
	LoCal+SGD (Dynamic)	**6.23**	**1.43×**
	Training with Stochastic depth	6.79	1.35×
	Freezing layers during training	6.51	1.65×
VGG13 (Cifar10)	Baseline SGD	7.16	1×
	LoCal+SGD (Static)	**7.28**	**1.32×**
	LoCal+SGD (Dynamic)	**7.25**	**1.28×**
	Freezing layers during training	7.43	1.42×
ResNet18 (Cifar100)	Baseline SGD	23.39	1×
	LoCal+SGD (Static)	**23.61**	**1.47×**
	LoCal+SGD (Dynamic)	**23.63**	**1.44×**
	Training with Stochastic depth	23.97	1.35×
	Freezing layers during training	23.74	1.62×
VGG13 (Cifar100)	Baseline SGD	31.36	1×
	LoCal+SGD (Static)	**31.56**	**1.3×**
	LoCal+SGD (Dynamic)	**31.59**	**1.32×**
	Freezing layers during training	31.94	1.42×

All experimental results pertaining to LoCal+SGD are highlighted in bold.

TABLE 3 | Comparing accuracy at Iso-runtime.

Dataset	Network	Top-1 err. with LoCal+SGD (%)	Top-1 err. with baseline SGD (%)
ImageNet	ResNet34	27.04	27.36
	ResNet50	24.41	24.67
	MobileNetV2	28.94	29.18
Cifar10	VGG13	7.25	7.56
	ResNet18	6.23	6.47
Cifar100	VGG13	31.59	31.9
	ResNet18	23.63	23.97

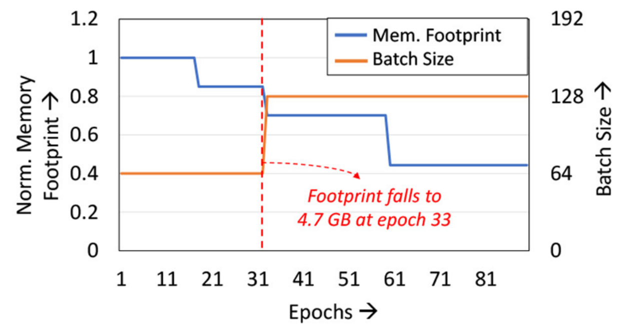


FIGURE 6 | Analyzing memory footprint and batch-size variation.

TABLE 4 | Analyzing impact of increasing batch-size on ImageNet.

Network	Training strategy	Top-1 err. (%)	Speed-up
ResNet50	Baseline SGD (fixed batch-size)	24.06	1×
	LoCal+SGD (fixed batch-size)	**24.48**	**1.27×**
	LoCal+SGD (variable batch-size)	**24.51**	**1.34×**

All experimental results pertaining to LoCal+SGD are highlighted in bold.

with stochastic depth cannot exploit this feature, since they do not impact memory footprint substantially.

3.4. Visualizing Activation Distributions

LoCal+SGD utilizes localized updates to adjust the weights of the initial feature-extraction layers. As discussed previously, these localized updates have been demonstrated to approximate popular unsupervised learning algorithms such as principal component analysis, k-means clustering, *etc.* We illustrate this in the context of LoCal+SGD. To this end, after training is complete, we extract the top 2 principal components of the activation outputs of the locally updated layers. For comparison, this procedure is repeated for the same layers when they are updated with SGD instead. In **Figure 7**, we have plotted the dominant components of activations of the second and fourth convolutional layers of the ResNet18-Cifar10 benchmark, when trained with SGD and LoCal+SGD. Interestingly, we find that LoCal+SGD provides comparable (**Figures 7B,C**), or in some cases even better separation (**Figure 7A**) between the classes compared to SGD. We illustrate this further in **Figure 8**, wherein we plot the L_2 difference between the top-2 principal components of either class across selected layers of the network. It is noteworthy that LoCal+SGD achieves these separations while requiring a substantially lower number of operations per convolutional layer.

3.5. Ablation Studies

As mentioned in section 2, the efficacy of the static and dynamic learning mode selection algorithms are controlled by different hyper-parameters. The performance of the static selection algorithm is dictated by c_1 and c_2, while α and L_{shift} impact the dynamic algorithm. Different values of these parameters can result in different learning mode configurations during training, resulting in different points in the computational efficiency vs. accuracy trade-off space. To understand this trade-off, we individually study the impact of each parameter. Further, we also discuss the impact of the weak supervision scheme on accuracy.

We begin by first analyzing the impact of the α and L_{shift} parameters used in the dynamic learning mode selection algorithm.

Impact of α: **Figures 9A,B** illustrate the runtime savings and accuracy achieved for different values of α, for the ResNet50 and MobileNetV2 benchmarks on ImageNet. For both benchmarks,

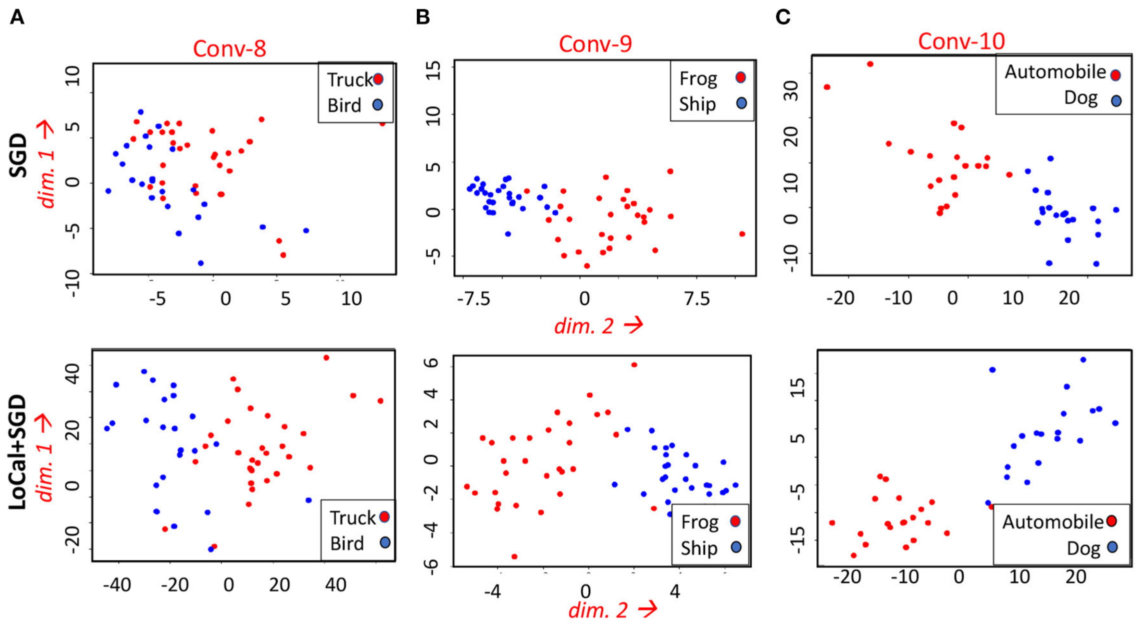


FIGURE 7 | Comparing the top 2 principal components at different layers of the ResNet18-Cifar10 benchmark for the following classes **(A)** Truck and Bird **(B)** Frog and Ship **(C)** Automobile and Dog.

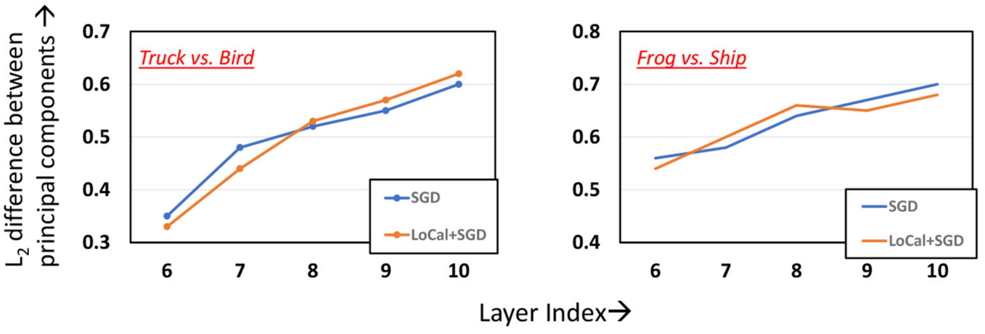


FIGURE 8 | Analyzing L_2 difference between the principal components across the layers.

increasing α from 0.1 until 0.5 improves the runtime benefits as the application of localized updates increases, while maintaining the loss in accuracy to within 0.2–0.3%. However, once α exceeds 0.6, the degradations in accuracy exceed 0.5% on both benchmarks. The speedups increase to 1.6× when around 1% loss in accuracy is tolerable.

Impact of L_{shift}: In **Figures 9C,D** we highlight the impact of different L_{shift} values (recall that L_{shift} denotes the amount by which we shift the transition layer). Note that we have normalized L_{shift} to the total network depth. The graphs indicate that for L_{shift} values between 10 and 15% of the total number of layers in the network, the loss in accuracy remains within 0.5%, and the runtime savings increase with increasing L_{shift}. However, when L_{shift} exceeds 15%, the accuracy begins to degrade. This can be attributed to the simultaneous transition in the learning mode of a large number of layers affecting convergence of training.

From **Figure 9**, we make an additional observation—across ResNet50 to MobileNetV2, similar values of α and L_{shift} provide a good trade-off. We find that this observation holds for other ImageNet benchmarks analyzed, such as ResNet34. We therefore utilize a common set of hyper-parameter values for all networks of a particular dataset. This eliminates the need to conduct a hyper-parameter search process to determine α and L_{shift} for every new network that is to be trained.

The static learning mode selection algorithm is controlled by two parameters c_1 and c_2. c_1 represents the maximum number of layers to which localized updates are applies, while c_2 controls the epoch at which localized updates begin. In **Figure 10**, we present the accuracy and runtime benefits obtained when varying these parameters for the ResNet18 network on the Cifar10 dataset. c_1 is represented as a fraction of the total number of layers in the network, while c_2 is expressed as a fraction of the total training

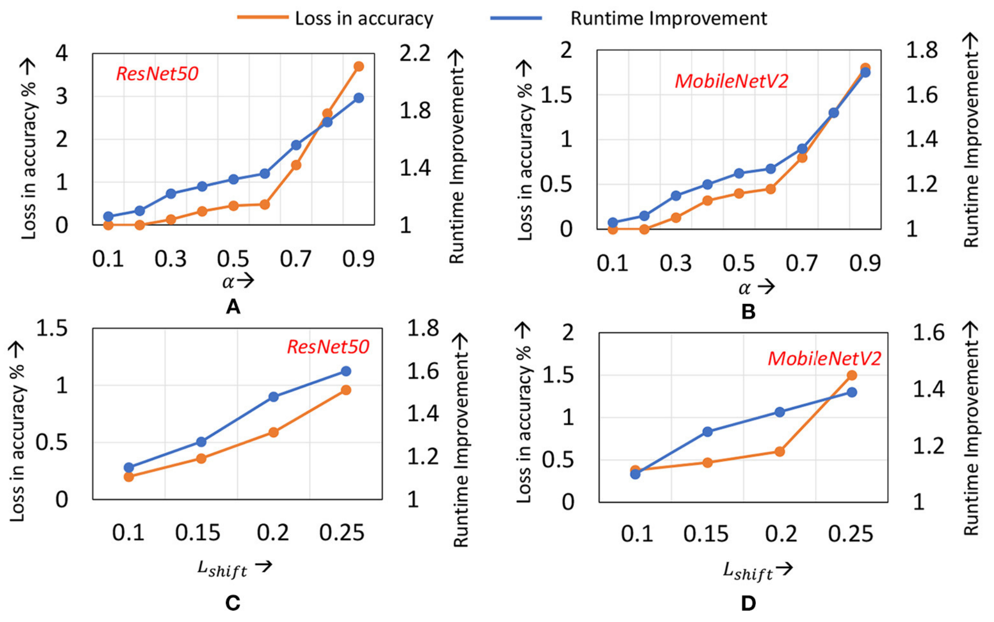


FIGURE 9 | Compute efficiency vs. accuracy trade-off on ImageNet when **(A)** α is varied for ResNet50 **(B)** α is varied for MobileNetV2 **(C)** L_{shift} is varied for ResNet50 **(D)** L_{shift} is varied for MobileNetV2.

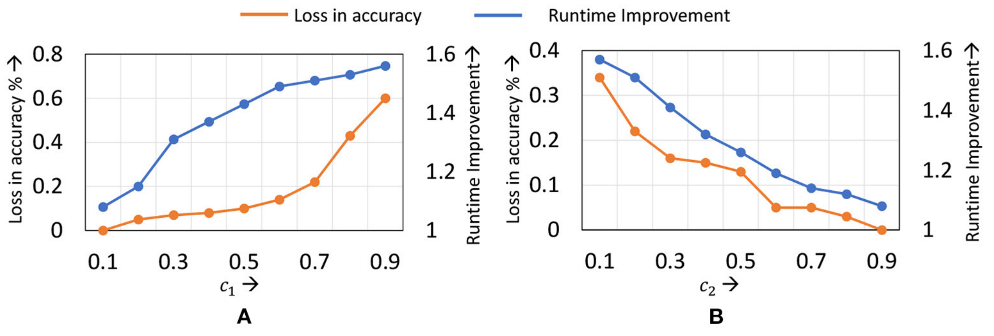


FIGURE 10 | Compute efficiency *vs.* accuracy trade-off obtained by varying **(A)** c_1 and **(B)** c_2 in the static learning mode selection algorithm for ResNet18 on the Cifar10 dataset.

TABLE 5 | Impact of weak supervision on accuracy.

Dataset	Network	Top-1 err. with weak supervision (%)	Top-1 err. without weak supervision (%)
ImageNet	ResNet34	27.04	27.1
	ResNet50	24.41	24.49
	MobileNetV2	28.94	29.03
Cifar10	VGG13	7.25	7.39
	ResNet18	6.23	6.41
Cifar100	VGG13	31.59	31.7
	ResNet18	23.63	23.75

TABLE 6 | Accuracy and runtime benefits of LoCal+Adam.

Training technique	Top-1 error (%).	Speed-up
Adam	7.7	1×
LoCal+Adam	7.96	1.48×

epochs. We observe that to achieve a good trade-off, setting c_1 between 0.5 and 0.7, and c_2 in the range of 0.2–0.3 provides best results. As with the dynamic learning mode selection algorithm, we find that common c_1 and c_2 values can be used for all networks of a particular dataset, with only marginal impact on performance.

Impact of weak supervision: In **Table 5**, we highlight the impact of the weak supervision technique on final classification accuracy. Across all our benchmarks, the weak supervision technique improves accuracy by 0.06–0.17%, bringing the final accuracy of $\text{LoCal} + \text{SGD}$ closer to baseline SGD. This improvement comes at no cost in runtime, since the overhead of modulating the learning rate of locally updated layers is negligible.

3.6. LoCal+Adam

We analyze the impact of combining localized learning with other gradient descent based learning algorithms such as Adam. In **Table 6**, we successfully demonstrate LoCal+Adam on the Cifar10-ResNet18 benchmark. We note that all other aspects of the design such as the learning mode selection algorithm etc., remain unchanged. These results thus speak to the widespread applicability of our technique, irrespective of the gradient descent learning algorithm used.

3.7. Applicability of LoCal+SGD to Other Networks

In this paper, $\text{LoCal}+\text{SGD}$ has been explored and demonstrated with a focus on convolutional neural networks. We demonstrate the applicability of $\text{LoCal}+\text{SGD}$ to segmentation networks such as U-Net (Ronneberger et al., 2015). The long-range connections in U-Net are handled similar to the shortcut connections in ResNets. Consider a Layer K, whose input and output activations are A_{K-1} and A_K. Further, let us assume Layer K receives activation input A_J from a preceding layer J. The weight update for Layer K is performed by convolving the summed activation $A_K + A_J$, with A_{K-1}. **Table 7** demonstrates the applicability

TABLE 7 | Accuracy and runtime benefits of $\text{LoCal}+\text{SGD}$ on U-Net.

Training technique	Dice coefficient.	Speed-up
Baseline SGD	0.948	1×
$\text{LoCal}+\text{SGD}$	0.943	1.26×

of $\text{LoCal}+\text{SGD}$ to U-Net training on the ISBI 2012 challenge dataset (Arganda-Carreras et al., 2015).

4. RELATED WORK

This section discusses research directions that are related to $\text{LoCal}+\text{SGD}$. These efforts can be broadly categorized into two classes. The first class of efforts focus on improving the computational efficiency of gradient-descent based DNN training. The second class of efforts involve the design of neuro-inspired learning rules such as feedback alignment, *etc.* (Nøkland, 2016). Our work is orthogonal to both classes of efforts, since our focus is on how to selectively combine localized learning rules with SGD for better computational efficiency. In section 3, we demonstrated how $\text{LoCal}+\text{SGD}$ achieves superior accuracy vs. computational efficiency trade-off than some of these efforts. We next elaborate upon the research efforts in both aforementioned directions.

Hyper-parameter tuning: Many efforts are directed toward achieving training efficiency by controlling the hyper-parameters involved in gradient-descent, notably the learning rate. For example (Akiba et al., 2017; Goyal et al., 2017; You et al., 2017a,b) propose learning rate tuning algorithms that accelerate training with no loss in accuracy, and scale to hundreds of CPU/GPU cores.

Model size reduction during training: Model size reduction via pruning and quantization is a popular technique to reduce compute costs during inference. In many of these efforts, a dense or full precision model is re-trained or fine-tuned to obtain a pruned or quantized model. However, recent efforts have also investigated dynamically pruning (Lym et al., 2019) or quantizing (Sun et al., 2019) a model during training itself, resulting in training speed-ups. Taking a slightly different approach (Huang et al., 2016) proposes stochastically dropping residual blocks on extremely deep networks such as ResNet-1202, not only for training runtime benefits but also better accuracies due to improved gradient strength.

Instance importance based training: Recent research efforts have discovered that not all training samples are required for improving loss minimization during SGD training (Jiang et al., 2019; Zhang et al., 2019). That is, a sizable fraction of the samples can be skipped during several epochs, depending on their impact on the classification loss evaluated during *FP*. This translates to a reduction in mini-batches per epoch, providing considerable runtime benefits.

Neuro-inspired learning rules: Back-propagation algorithms utilized in DNN training are not biologically plausible, i.e., they greatly differ from how learning happens in the brain. To this end, there have been several efforts that propose biologically plausible learning algorithms. These algorithms have demonstrated considerable success on complex networks and datasets. For example, feedback alignmnent algorithms (Nøkland, 2016) tackle the weight transport problem (Liao et al., 2015) by allowing for asymmetry in the weight values during forward and back propagation. Likewise, target propagation (Lee et al., 2015) encourages neural activity to reach desired target activations evaluated during forward propagation, instead of utilizing loss gradients. In equilibrium propagation (Scellier and Bengio, 2017), the gradients of locally defined objective functions are used to update the weights of a layer, thereby eliminating the propagation of gradients across the network.

LoCal+SGD represents a new direction wherein we combine localized learning and SGD in the context of an overall supervised learning framework, with the goal of reducing training time. Therefore, we surmise that advances in either SGD-based learning or localized learning can be incorporated within LoCal+SGD to further advance its benefits.

5. CONCLUSION

In this paper, we introduce a new approach to improve the training efficiency of state-of-the-art DNNs. Specifically, we take advantage of the computationally efficient nature of localized learning rules and selectively update some layers with these rules instead of SGD. We design a learning mode selection algorithm that determines the learning mode for the layers of the network in every epoch in order to achieve a favorable tradoff between training time and accuracy. Further, we also implement a low-cost weak supervision scheme that brings the accuracy of the proposed scheme closer to traditional SGD-based training. Across a benchmark suite of 8 DNNs, we achieve upto 1.5× reduction in training times on a modern GPU platform.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

SK devised and conducted experiments. All authors contributed to the formulation of the problem statement.

FUNDING

This work was supported in part by Semiconductor Research Corporation (SRC).

REFERENCES

Agrawal, P., Girshick, R. B., and Malik, J. (2014). Analyzing the performance of multilayer neural networks for object recognition. *arXiv.[Preprint].arXiv:1407.1610*. doi: 10.1007/978-3-319-10584-0_22

Akiba, T., Suzuki, S., and Fukuda, K. (2017). Extremely large minibatch SGD: training resnet-50 on imagenet in 15 minutes. *arXiv.[Preprint].arXiv:1711.04325*.

Arganda-Carreras, I., Turaga, S. C., Berger, D. R., Cireşan, D., Giusti, A., Gambardella, L. M., et al. (2015). Crowdsourcing the creation of image segmentation algorithms for connectomics. *Front. Neuroanat.* 9:142. doi: 10.3389/fnana.2015.00142

Bottou, L. (2010). "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, eds Y. Lechevallier and G. Saporta (Princeton, NJ: Physica-Verlag HD), 103–189.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning, Vol.119*, eds. H. D. III and A. Singh (Proceedings of Machine Learning Research PMLR), 1597–1607.

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., et al. (2012). "Large scale distributed deep networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems, Vol. 1, NIPS' 12* (Red Hook, NY: Curran Associates Inc.), 1223–1231.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "ImageNet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL: IEEE).

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* 29, 141–142.

Goldberg, Y., and Hirst, G. (2017). *Neural Network Methods in Natural Language Processing* (Bar-Ilan University, Israel: Morgan Claypool Publishers)

Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., et al. (2017). Accurate, large minibatch SGD: training imagenet in 1 hour. *arXiv[Preprint[.arXiv:1706.02677*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv[Prerpint].arXiv:1512.03385*. doi: 10.1109/CVPR.2016.90

Hebb, D. (1949). *The Organization of Behavior: A Neuropsychological Theory*. Hillsdale, NJ: Psychology Press, p. 378.

Hénaff,O. J., Srinivas, A., Fauw, J.D., Razavi, A.,Doersch, C., Eslami, S.M. A., et al. (2019). *Data-Efficient Image Recognition With Contrastive Predictive Coding*. arXiv:1905.09272.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. *arXiv[Preprint].arXiv:1603.09382*. doi: 10.1007/978-3-319-46493-0_39

Jiang, A. H., Wong, D. L. K., Zhou, G., Andersen, D. G., Dean, J., Ganger, G. R., et al. (2019). *Accelerating Deep Learning by Focusing on the Biggest Losers*. arXiv:1910.00762

Kingma, D. P., and Ba, J. (2015). "Adam: a method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015*, eds Y. Bengio and Y. LeCun, May 7–9, 2015, Conference Track Proceedings (San Diego, CA).

Krizhevsky, A., Nair, V., and Hinton, G. (2009). Cifar-10 (canadian institute for advanced research).

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems, Vol. 1, NIPS' 12* (Red Hook, NY: Curran Associates Inc), 1097–1105.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 84–90. doi: 10.1145/3065386

Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. (2015). "Difference target propagation," in *Proceedings of the 2015th European Conference on Machine Learning and Knowledge Discovery in Databases, Vol. Part I, ECMLPKDD'15, Gewerbestrasse 11 CH-6330* (Cham: CHE. Springer), 498–515.

Li, D., Zhou, A., and Yao, A. (2019). "Hbonet: Harmonious bottleneck on two orthogonal dimensions," in *The IEEE International Conference on Computer Vision (ICCV)* (Seoul: IEEE).

Liao, Q., Leibo, J. Z., and Poggio, T. A. (2015). How important is weight symmetry in backpropagation? *arXiv[Preprint].arXiv:1510.0506*.

Lym, S., Choukse, E., Zangeneh, S., Wen, W., Erez, M., and Shanghavi, S. (2019). Prunetrain: Gradual structured pruning from scratch for faster neural network training. *arXiv[Preprint].arXiv:1901.09290*. doi: 10.1145/3295500.3356156

Ng, J. Y., Hausknecht, M. J., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. (2015). Beyond short snippets: Deep networks for video classification. *arXiv[Preprint].arXiv:1503.08909*.

Nøkland, A. (2016). "Direct feedback alignment provides learning in deep neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16* (Red Hook, NY: Curran Associates Inc.), 1045–1053.

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *J. Math. Biol*. 15, 267–273. doi: 10.1007/BF00275687

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems* 32, eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Vancouver, BC: Curran Associates, Inc.), 8024–8035.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *arXiv[Preprint].arXiv:1505.04597*. doi: 10.1007/978-3-319-24574-4_28

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. (2018). Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv[Preprint].arXiv:1801.04381*. doi: 10.1109/CVPR.2018.00474

Scellier, B., and Bengio, Y. (2017). Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci*. 11:24. doi: 10.3389/fncom.2017.00024

Simonyan, K., and Zisserman, A. (2015). "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations* (New York, NY: Springer), 235–239.

Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V., et al. (2019). "Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks," in *NeurIPS*.

van den Oord, A., Li, Y., and Vinyals, O. (2018). *Representation learning with contrastive predictive coding*. arXiv:1807.03748.

You, Y., Gitman, I., and Ginsburg, B. (2017a). Scaling SGD batch size to 32k for imagenet training. *arXiv[Preprint].arXiv:1708.03888*.

You, Y., Zhang, Z., Hsieh, C., and Demmel, J. (2017b). 100-epoch imagenet training with alexnet in 24 minutes. *arXiv[Preprint].arXiv:1709.05011*.

Zhang, J., Yu, H., and Dhillon, I. S. (2019). Autoassist: a framework to accelerate training of deep neural networks. *arXiv[Preprunt].arXiv:1905.03381*.

Zhong, S. H. (2005). "Efficient online spherical k-means clustering," *Proceedings 2005 IEEE International Joint Conference on Neural Networks, 2005* (Montreal, QC: IEEE), 3180–3185.

Zhou, C., Sun, C., Liu, Z., and Lau, F. C. M. (2015). A C-LSTM neural network for text classification. *arXiv[Preprint].arXiv:1511.08630*.

Conflict of Interest: SS and SV are employed by IBM Research.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Krithivasan, Sen, Venkataramani and Raghunathan. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 31 March 2022
doi: 10.3389/fnins.2022.838832

ALSA: Associative Learning Based Supervised Learning Algorithm for SNN

Lingfei Mo, Gang Wang, Erhong Long and Mingsong Zhuo*

FutureX LAB, School of Instrument Science and Engineering, Southeast University, Nanjing, China

Spiking neural network (SNN) is considered to be the brain-like model that best conforms to the biological mechanism of the brain. Due to the non-differentiability of the spike, the training method of SNNs is still incomplete. This paper proposes a supervised learning method for SNNs based on associative learning: ALSA. The method is based on the associative learning mechanism, and its realization is similar to the animal conditioned reflex process, with strong physiological plausibility and rationality. This method uses improved spike-timing-dependent plasticity (STDP) rules, combined with a teacher layer to induct spikes of neurons, to strengthen synaptic connections between input spike patterns and specified output neurons, and weaken synaptic connections between unrelated patterns and unrelated output neurons. Based on ALSA, this paper also completed the supervised learning classification tasks of the IRIS dataset and the MNIST dataset, and achieved 95.7 and 91.58% recognition accuracy, respectively, which fully proves that ALSA is a feasible SNNs supervised learning method. The innovation of this paper is to establish a biological plausible supervised learning method for SNNs, which is based on the STDP learning rules and the associative learning mechanism that exists widely in animal training.

Keywords: spiking neural network, associative learning, supervised learning, STDP, long-term plasticity

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Gina Adam,
George Washington University,
United States
Dan Hammerstrom,
Portland State University,
United States

****Correspondence:***
Lingfei Mo
lfmo@seu.edu.cn

Specialty section:
This article was submitted to
Neuromorphic Engineering,
a section of the journal
Frontiers in Neuroscience

Received: *18 December 2021*
Accepted: *07 March 2022*
Published: *31 March 2022*

Citation:
Mo L, Wang G, Long E and
Zhuo M (2022) ALSA: Associative
Learning Based Supervised Learning
Algorithm for SNN.
Front. Neurosci. 16:838832.
doi: 10.3389/fnins.2022.838832

INTRODUCTION

In recent years, neural networks have made great progress in the field of information processing. Especially with the development of deep neural network (DNNs) (Rawat and Wang, 2017) and convolutional neural networks (CNNs) (LeCun et al., 1989; O'Shea and Nash, 2015; Rawat and Wang, 2017), the performance and application range of artificial neural networks (ANNs) has been greatly improved. However, there are still some problems for the ANNs. For example, most ANNs train the network according to the backpropagation of errors. Therefore, ANNs training requires a large number of labeled samples which is labor-intensive. In addition, although ANNs claim to be physiologically plausible, their training process is different from biological neural networks, which are mainly based on gradient descent and error backpropagation. Different from biological neural networks, which are mainly based on unsupervised learning like Hebbian learning (Caporale and Dan, 2008), the learning process of ANNs is mainly based on supervised learning. At the same time, the error backpropagation mechanism commonly used in ANNs lacks widespread evidence in biological neural networks.

Spiking neural networks (SNNs) (Maass, 1997) attracts more and more researchers because of their similarity to biological neural networks (Pan et al., 2020; Zirkle and Rubchinsky, 2020). Compared with ANNs, SNNs uses spike rate or spike timing to transmit information between neurons (Maass, 1997) instead of using numerical values to transmit information, and its unsupervised training process is also based on physiologically plausible spike-timing-dependent plasticity (STDP) (Caporale and Dan, 2008; Diehl and Cook, 2015; Masquelier and Kheradpisheh, 2018; Mozafari et al., 2019) instead of error backpropagation. Therefore, SNNs are closer to the biological neural network in principle. Thanks to the characteristics of SNN's event-driven computing, those neurons that are not activated will not participate in the actual computing, thus saving computing resources. It is very suitable for low-power consumption computing on dedicated chips, such as TrueNorth (Akopyan et al., 2015), Tianjic (Pei et al., 2019), Loihi (Davies et al., 2018), Darwin (Shen et al., 2016), etc. Using these chips, SNNs have an order of magnitude advantage over ANNs in terms of computational power consumption (Akopyan et al., 2015; Davies et al., 2018; Pei et al., 2019; Xu et al., 2020).

The main reason restricting the development of SNNs is the lack of training algorithms, especially the supervised learning algorithms of SNNs. Since the spike signal is not differentiable, the error backpropagation widely used in ANNs cannot be used to train SNNs. At the same time, backpropagation also rarely exists in biological neural networks (Bengio, 2015; Lillicrap et al., 2020). Therefore, it is difficult to find a physiologically plausible SNN supervised training algorithm.

Many scholars have also proposed some training methods for SNNs, which can be mainly divided into the following two categories. The first is the ANN-SNN conversion. This type of method uses specific rules to convert the ANN trained networks into a corresponding structure of the SNN networks, making full use of the low power consumption characteristics of the SNNs calculation (Pérez-Carrasco et al., 2013; Diehl et al., 2015; Rueckauer et al., 2017). Since the training process does not occur in the SNNs, it cannot fully reflect the characteristics of the strong physiological rationality of the SNNs. The second type is based on error backpropagation to obtain higher model accuracy, such as Tempotron (Gütig and Sompolinsky, 2006), PSD(Yu et al., 2013), ReSuMe (Ponulak, 2005), MST (Gütig, 2016), EMLC(Yu et al., 2020), MPD-AL(Zhang et al., 2019), SpikeProp (Bohte et al., 2000), STCA(Gu et al., 2019), etc. The supervised learning methods mainly calculate the difference between the voltage of the output neuron at target time points and the threshold value to change the weight (Xie et al., 2016). There is also some research using backpropagation and gradient descent to train deep neural networks for SNN models (Lee et al., 2020). Most of these rules make proper adjustments to neurons or spike signals to make BP feasible in SNNs, but they lack certain physiological plausibility.

Compared with supervised learning, SNNs unsupervised learning is much more unified. At present, the most widely used unsupervised learning method of SNNs is STDP and its variants, which can obtain significant unsupervised clustering and feature extraction results (Diehl and Cook, 2015; Masquelier and Kheradpisheh, 2018; Tavanaei et al., 2019; Xu et al., 2020). At the same time, the STDP rules have been supported by many related experiments in the field of neuroscience. They have been widely confirmed in biological neural networks and have strong physiological plausibility (Caporale and Dan, 2008).

Almost all supervised learning rules use error backpropagation and gradient descent methods to achieve good accuracy, though these methods lack biological interpretability. Lee et al. (2015) pointed out that biological neurons are linear and non-linear operations, while backpropagation is purely linear, and there is no corresponding mechanism to realize the precise timing of backpropagation signals and the alternating of feedforward and feedback propagation, as well as retrograde signal propagation along axons and synapses. Therefore, there is no biological justification for back transmission. Lillicrap et al. (2020) also showed that while feedback connections are ubiquitous in the cortex, it's hard to know how they transmit the error signals needed for backpropagation. The effect of feedback connection on neural activity still needs to be further explored. Given that, some researchers have begun to implement supervised learning combined with STDP rules. Legenstein et al. (2005) introduced a teacher signal by injecting current into the output neurons during training and combined it with STDP rules to achieve supervised learning. However, using this method does not guarantee STDP convergence for any input mode. The remote supervised method (ReSuMe) (Ponulak and Kasiński, 2010) uses STDP rules and makes output neurons spike at desired time points through a remote teacher signal. Wade (Wade et al., 2010) used Bienenstock, Cooper and Monroe (BCM) rules to adjust the learning window of STDP and proposed SWAT rules. In this method, the BCM model was used to slide the threshold and promote the synaptic weight to converge to a stable state. However, this method is only applicable to frequency coding. For ReSuMe and SWAT, although a liquid state machine or multi-layer feedforward network structure is used, only the synaptic weights of the output layer are learned, and the synapses of the hidden layer in the network are fixed after initialization. Hao et al. (2020) used symmetric STDP, combined with synaptic scaling and dynamic threshold, to achieve good results at both NMIST and fashion MNIST, but increasing the depth of the network has little effect on the performance of the network and is not conducive to the extension of this rule unless other methods such as convolution are introduced. Pfister et al. (2003), realized supervised learning by optimizing the possibility of observing postsynaptic impulse sequences at the expected time by starting from the criterion of probabilistic optimality and adding teacher signals to the model. But this model uses the relatively simple spike response model (SRM; Gerstner, 1995). Using other models will make this rule much more complicated. More details of probabilistic SNN can be found in Jang et al. (2019), which reviews probabilistic models and training methods based on a probabilistic signal processing framework. Also, there is some research that combines supervised and unsupervised STDP training. Using a simplified approximation of a conventional Bayesian neuron and an improved STPD rule, (Shrestha et al., 2017) combined unsupervised and supervised STDP learning to train a three-layer SNN on the MNIST dataset. Hao et al. (2020) achieved good performance on the MNIST dataset by combining their proposed symmetric spike-timing-dependent plasticity (sym-STDP) with synaptic scaling and dynamic threshold (Hao et al., 2020). There are also other plasticity-based unsupervised training with supervised modules (Querlioz et al., 2013; Kheradpisheh et al., 2018). However, most of these STDP based supervised learning methods do not have enough physiological plausibility.

To solve the above problems, an SNN supervised learning algorithm based on associative learning is proposed. The learning rules are based on the widely recognized STDP rules, and the classic STDP is simply adjusted while retaining physiological rationality. The major innovation of this paper is to establish a more biologically interpretable supervised learning method, which is based on the conditioned reflex associative learning mechanism that exists widely in animal training. To realize associative learning, an improved STDP model inspired by the heterosynaptic long-term plasticity is proposed.

The following contents of this article are mainly divided into methods, experiments and results, and discussion. The method part will introduce the neuron model, synaptic plasticity model, and the implementation methods of supervised learning. The experiments and results part includes the details of IRIS and MNIST classification networks, specific results, and process analysis of the classification tasks. In the discussion part, the advantages and current shortcomings of the supervised learning rules are analyzed.

METHODOLOGY

Neuron Model

In this paper, the leaky integrated and fire (LIF; Koch and Segev, 1998) model is adopted, which is a neuron model widely used in the field of SNN calculation and computational neuroscience simulation. This model is obtained by simplifying the Hodgkin Huxley (HH) (Hodgkin and Huxley, 1952) model but retains basic functionality. So that the computational results are close to those of the HH model, and the complexity and computational complexity of the model are greatly reduced. The model formula is shown as Equation 1.

$$C_m \frac{dV}{dt} = -g_L (V - V_L) + I_{syn} \tag{1}$$

Where C_m is the membrane capacitance, $V$is the membrane potential, g_L is the leakage conductance, V_L is the leakage potential, and I_{syn} is the input current from the presynaptic neurons. Assuming that the total conductance value is g_E, and the constant $\tau_m = \frac{C_m}{g_L}$ is defined, then (1) can be converted to (2).

$$\tau_m \frac{dV}{dt} = -(V - V_L) - \frac{g_E}{g_L}(V - V_E) \tag{2}$$

$$\tau_E \frac{dg_E}{dt} = -g_E + \sum_{j \in N_E} w_{j,i}\delta_t \tag{3}$$

g_E in (2) will dynamically change under the influence of the presynaptic spikes, and the specific changes are shown in (3). That is, once the presynaptic neuron generates a spike, g_E will increase non-linearly.V_E is the reversal potential of excitatory neurons, τ_E is the conductance decay time constant of excitatory neurons, N_E is the number of presynaptic neurons, δ_t is the specific moment when the presynaptic neuron generates spikes, and $w_{j,i}$ represents the connection weight of presynaptic neuron j to postsynaptic neuron i.

$$\text{if}\,(V > V_{thr}) \begin{cases} V = V_L \\ T_{ref} = T_0 \\ V_{thr} = V_{thr} + V_{thrDelta} \end{cases} \tag{4}$$

As shown in (4), when the membrane potential V increases to exceed the membrane potential threshold V_{thr}, the membrane potential will be reset, and the refractory period T_{ref} will be set to T_0. During the refractory period, the neuron will not respond to the presynaptic spikes as shown in **Figure 1**. At the same time, in order to ensure that the spike firing frequency of neurons is stable in a specific range, and avoid the situation where some neurons are firing too much and others not enough, the mechanism of neuron dynamic threshold is introduced referring to Diehl's approach (Diehl and Cook, 2015). Homeostasis, which is known in neuroscience, is also considered here (Watt and Desai, 2010). As shown in (4), every time a neuron generates a spike, the neuron threshold will be increased accordingly, thus raising the threshold for the next spike. $V_{thrDelta}$ is a hyperparameter used to control the difficulty of neuron spike generation.

$$\tau_{thr} \frac{dV_{thr}}{dt} = -(V_{thr} - V_{thrBase}) \tag{5}$$

Also, as shown in (5), V_{thr} will gradually decay to $V_{thrBase}$, lowering the membrane potential threshold of neurons that do not produce spikes for a while. Combined with (4), the difficulty of neuron spike firing is controlled at a reasonable range. τ_{thr}

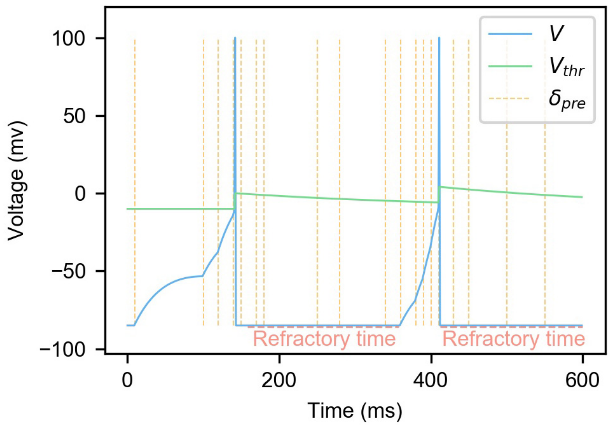


FIGURE 1 | The changes of membrane potential and membrane potential threshold of LIF neuron model with dynamic threshold under the influence of presynaptic neuron spikes. In the figure, the blue curve is the neuron membrane potential, the green curve is the membrane potential threshold, and the yellow vertical dashed lines are the moments when the presynaptic spikes are fired.

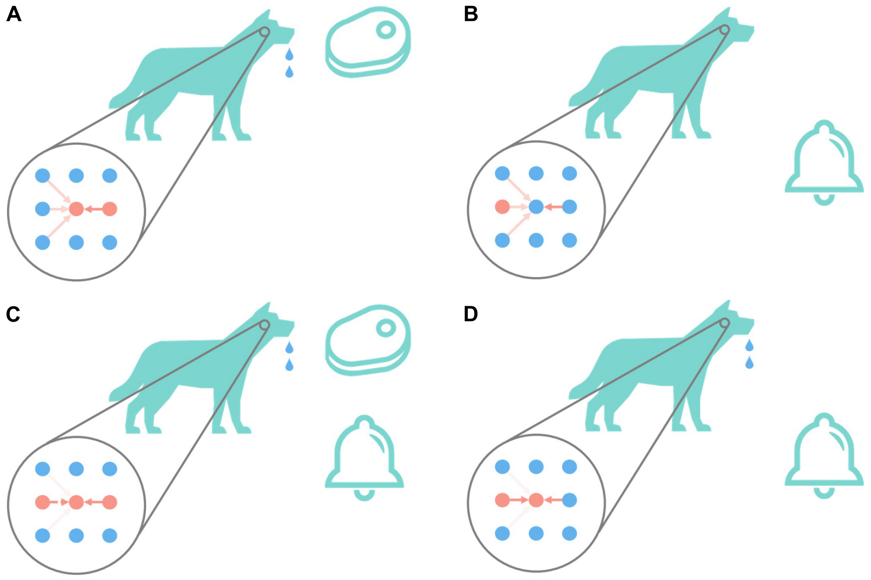


FIGURE 2 | Pavlov's dog experiment. The big circle in the figure represents the state diagram (imaginary) of the relevant neurons in the dog in the corresponding state. The first column represents auditory-related stimuli, the second column represents animal behaviors including drooling, and the third column represents olfactory-related stimuli. The depth of the red arrow indicates the strength of the excitatory connection between the corresponding neurons. The darker the color, the higher the strength of the connection between neurons, and vice versa, the lower the strength of the connection between neurons. The red dot indicates that the neuron is currently active (that is, it has fired a spike within a period), and the blue indicates that the neuron is resting (that is, it has not fired a spike for a while). The bells and meat in the picture will cause the second neuron in the first column and the second neuron in the third column to enter the active state, respectively. When the second neuron in the second column is active, the dog will drool. Pavlov's dog experiment is conducted in the order of **(A–D)**.

is the dynamic threshold decay time constant. **Figure 1** is an example of changes in neuron membrane potential and dynamic threshold. It can be seen from the figure that presynaptic neuron spikes will cause the neuron membrane potential to rise, and the membrane potential will slowly decrease over time. When the membrane potential exceeds the threshold, the membrane potential will rise and drop rapidly in a short time, completing the firing of a spike. Every time the neuron emits a spike, the membrane potential threshold will increase and gradually decay to its initial state. At the same time, when a neuron fires a spike, it will enter refractory time, during which the neuron does not respond to presynaptic spikes.

Synapse Model

The synapse model used in this paper is mainly based on the STDP rule, and the classic STDP is appropriately adjusted to make it more in line with the needs of this model.

$$\Delta w = \begin{cases} \eta\left(\alpha + \beta \cdot e^{-\frac{ISI}{\tau_p}}\right) * w * (1-w) \ \ if\ (ISI > 0) \\ 0 \qquad\qquad else \end{cases} \tag{6}$$

$$ISI = t_{post} - t_{pre} \tag{7}$$

Equation 6 is the synaptic plasticity model used in this paper, where Δw is the modified amplitude of the synapse weight after each spike, and ISI (inter-spike interval) is the time difference between the most recent spike time of the neuron before and after the synapse as in (7). η is the learning rate, α is a constant bias term that is usually less than 0 to simulate the heterosynaptic LTD (long-term depression)(Krug et al., 1985; Christie et al., 1994). β is used to adjust the intensity of weight change, usually greater than 0. τ_p is the time constant of the LTP (long-term potentiation) part to control the detail. Also, w(1-w) in equation 6 means limiting

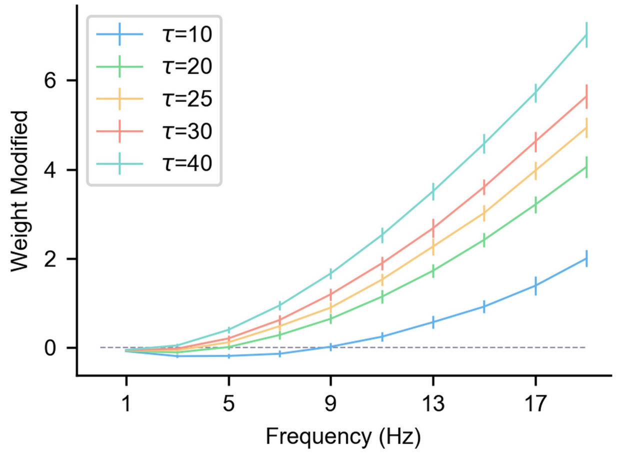


FIGURE 3 | The influence of the pre/postsynaptic spike frequency on the synapse weight under different τ_p. The gray dashed line in the figure is where the weight change is 0, and the length of the vertical lines on the curve represents the standard deviation of multiple trials (n = 50). The abscissa is the pre/postsynaptic spike frequency. Presynaptic and postsynaptic frequencies are equal and obey the Poisson distribution. Each simulation time is 100 s, α = –0.1, β = 1, and η = 0.01.

the weight to between 0 and 1. And when the current weight approaches 0 or 1, the change of weight is very small.

Heterosynaptic LTD is a long-term plasticity phenomenon that exists widely in biological neural networks (Krug et al., 1985; Christie et al., 1994). The main manifestation is that when a certain neuron generates a spike, the strength of the synapses which regards the current neuron as the postsynaptic neuron will be attenuated to a certain extent. This is mainly because VDCCs (voltage-dependent calcium channels) are activated after the neuron generates a spike signal. After the postsynaptic neuron pulses, the VDCC channel on the postsynaptic neuron opens, which activates inhibitory calmodulin such as PP1 in the cell, and produces a series of intermediate actions that ultimately lead to a decline in synaptic strength (Krug et al., 1985; Caporale and Dan, 2008). To realize heterosynaptic LTD, the classic STDP is modified in this paper. In the case of ISI $>$ 0, LTP will be generated when ISI is less than a certain value, and LTD will be generated when ISI is greater than a certain value. In Equation 6, the parameter α simulates the heterosynapses, producing the results that in the case of ISI $>$ 0, LTD is generated when ISI is greater than a certain threshold. When ISI $<$ 0, if the same LTD as the classic STDP is used, then, on the whole, the effect of LTD will be much greater than that of LTP, which will make all synaptic weights tend to 0 in the process of training. Therefore, the delta weight was set to 0 when ISI $<$ 0 to balance LTD and LTP. With this improved STDP and the heterosynaptic LTD, associative learning could be achieved.

Supervised Learning Algorithm

Associative learning is the basis of cognition and plays an important role in the process of animal learning and training (McSweeney and Murphy, 2014). **Figure 2** is the classic associative learning experiment of Pavlov's dog (Pavlov, 2010). As shown in **Figure 2A**, in the beginning, the dog secretes saliva under the stimulation of meat. This is an instinctive behavior, that is, there are naturally high-strength connections between the meat neuron and drooling neuron. And as shown in **Figure 2B**, the dog does not drool under the stimulation of the bell, and the connections between auditory-related neurons and the drooling neuron are relatively weak, which cannot cause the dog to drool. As shown in **Figure 2C**, give dog meat and bell stimulation at the same time, repeat this step for a while, the connection between the bell and drooling neuron is gradually potentiated. The connections between other auditory neurons and drooling are weakened. The result is shown in **Figure 2D**. Only under the stimulation of the bell, the dog drools too. This process realizes the associated learning of bells and drooling.

The conclusion can be made by observing the changes in the connections between neurons in the process: The essence of associative learning is that the connection strength between neurons that are simultaneously activated within a period increases, and the connection strength between unrelated stimuli and unresponsive behaviors decreases. This phenomenon is also consistent with the Hebb rule "neurons that fire together, wire together." The above steps are widely used in animal training to adjust the behavior of the training object by establishing the relationship between specific things (Pavlov, 2010). This process is similar to the effect achieved by supervised learning. So, is there a rule of synaptic long-term plasticity that can achieve similar effects, and then achieve associative learning and supervised learning?

Figure 3 shows the $\triangle w$ of the synaptic long-term plasticity rule introduced above under different frequency pre/postsynaptic spikes. As can be seen from the figure, τ_p affects the results significantly. However, when the pre/postsynaptic spike frequency is high enough, $\triangle w$ under any τ_p tends to increase. To be specific, in multiple (n = 50) simulations, the presynaptic spikes obeyed the Poisson distribution at a specific frequency. The relative positions of the pre/postsynaptic spikes are uncertain, but the standard deviation of $\triangle w$ or multiple trials is controlled within a relatively small range (**Figure 3**), indicating that when the pre/postsynaptic spike frequency is high enough, the synaptic weight shows a relatively stable LTP. This result is consistent with experimental results in neuroscience (Sjöström et al., 2001). Therefore, when a specific spike pattern (a combination of neurons' spike states, that is, some neurons generate spikes and others do not) is expressed in presynaptic neurons, synapses from presynaptic neurons relating a specific pattern to a specific postsynaptic neuron can be enhanced by inducing the specific postsynaptic neuron to fire. In the same way, the neurons that are not related to the current spike pattern do not produce spikes, and the strength of their connections to the current postsynaptic neurons is weakened under the effect of the heterosynaptic LTD. These characteristics can be used to enhance some synapses and weaken others, achieve a result similar to the above associative learning, and then realize supervised learning.

Take the network in **Figure 4** as an example. The input neurons in the network are equivalent to the bell signals in **Figure 2**, the output neurons in the network are equivalent to the drooling signals and the supervised neurons are equivalent to the meat signals. It simulates the associative learning process of Pavlov's dog experiment, which enables the output neuron to learn the input signal based on both the teacher signal and the input signal.

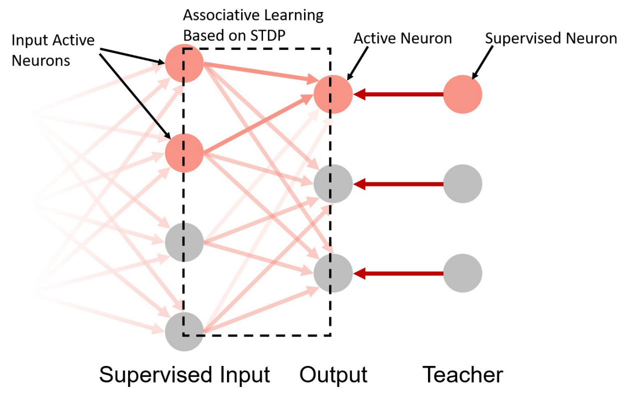


FIGURE 4 | A schematic diagram of the supervised learning network structure. The first, second, and third layers are the supervised input layer, output layer, and teacher layer, respectively. The gradient on the left indicates that the first layer can be used as the output layer of the previous network.

The implementation steps of supervised learning are as follows:

(1) Construct the network structure shown in **Figure 4**, where the first column is the supervised learning input layer (can be used as the output layer of unsupervised learning or other supervised learning output layers). The second column is the output layer, and the third column is the teacher layer. The number of neurons in the output layer and the teacher layer is equal to the number of sample categories, and a one-to-one mapping relationship between the output layer, the teacher layer, and the sample categories is constructed.
(2) Input spike signals to the supervised input layer, the spike signals are from the encoded spikes or the spikes of the previous neurons. Mark all neurons with spikes in the supervised input layer as I_s and others in the same layer as I_{non}.
(3) Simultaneously with (2), input spike signals to the teacher layer of the corresponding category, and induce the output neurons of the corresponding category to generate spikes. Mark the neuron with spikes in the output layer as O_s and others in the same layer as O_{non}.
(4) Since I_s and O_s both emit spikes for a while, under the mechanism described above, as long as the spike frequency of I_s and O_s is high enough, the strength of the synaptic connection from I_s to O_s will increase. The specific enhancement intensities are positively correlated with the spike frequencies of each neuron in I_s. At the same time, since I_{non} does not produce spikes, the strength of the synaptic connection from I_{non} to O_s will decrease under the effect of the heterosynaptic LTD. The strengths of all synaptic connections to O_{non} remain unchanged.
(5) Change the next sample and label, repeat steps (2) to (4) until the training of all samples is completed.

In the inducement of association supervised learning, it is necessary to use the long-term plasticity rules introduced in Equations 6, 7. In contrast, due to the existence of the negative semi-axis LTD in the classic STDP, under the effect of high-frequency pre/postsynaptic spikes, the change of synaptic weights will have a greater relationship with the specific spike moments, which makes it difficult for stable LTP to arise as in **Figure 3**. At the same time, due to the lack of heterosynaptic LTD, the synaptic connection of unrelated neurons cannot be effectively inhibited, so it cannot be used to realize associative learning and supervised learning.

Based on the above phenomenon, the potentiation of specific neuron connections can be achieved by inducing target neurons to emit spikes, that is, the spike induction of target neurons can achieve synaptic potentiation between specific spike patterns and target neurons and synaptic depression between unrelated neurons and target neurons. We call it ALSA (associative learning based supervised learning algorithm for SNNs). The supervised part only exists in the teacher layer, which is realized by stimulating the neurons in the teacher layer with a certain frequency, and no additional statistics on the number of output spikes and the precise time of output spikes are required. Synaptic strengthening and weakening are still achieved through unsupervised long-term plasticity rules. Therefore, ALSA can be said to be more in line with biological interpretability which is based on the universal associative learning behavior of animals, and it is relatively easy to realize which requires only certain teacher stimulation. In the following part, the feasibility of ALSA will be verified by two specific experiments.

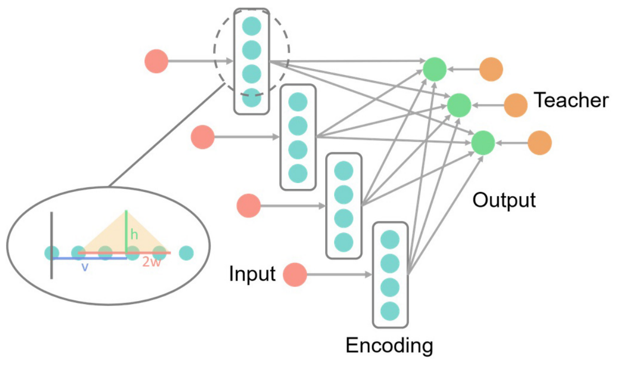


FIGURE 5 | IRIS classification network structure diagram. Each red input neuron in the picture receives an input vector of the IRIS dataset and encodes the numerical information into the neuron spikes signal of the encoding layer. The three neurons in the output layer correspond to the three categories of samples in the IRIS dataset, and the teacher layer is to generate supervised signals. The enlarged part of the dotted line in the figure is the details of the neurons in the coding layer, and the yellow translucent triangle is the encoding triangle.

EXPERIMENTS AND RESULTS

The IRIS and the MNIST classification experiments are used as examples. The details and results of the experiments as well as the feasibility of ALSA are introduced in detail. The simulator we used is an event-driven high accurate simulator (EDHA) for SNNs (Mo et al., 2021).

IRIS Classification

The IRIS (Dua and Graff, 2017) dataset contains three classes of irises, 50 of each class, and a total of 150 data. One of them is linearly separable from the other two, and the latter two are nonlinearly separable. The dataset contains four attributes: calyx length, calyx width, petal length, and petal width.

Figure 5 is the IRIS classification network structure diagram, including the input layer, encoding layer, output layer, and teacher layer. The input layer receives IRIS data and encodes it into neuron spikes of the encoding layer.

$$f_i = \max(0, \frac{-h\,|a_i - v|}{w} + h) \tag{8}$$

Since the input data of this dataset is all numerical information, it is difficult to directly use it in SNNs. Therefore, encoding is necessary. The encoding is realized by dispersing the data to multiple neurons. The encoding method is shown in (8),

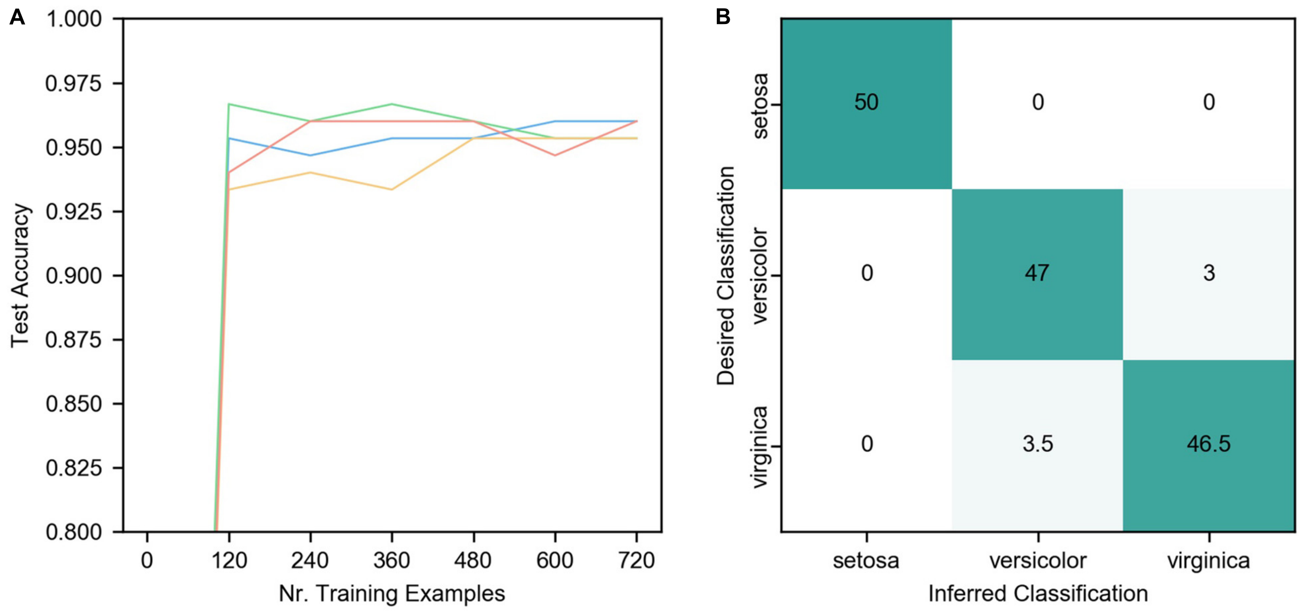


FIGURE 6 | Results of IRIS classification. **(A)** The network accuracy varies with the number of training samples. The four curves represent the data of 4 trials. The other parameters of the four trials are the same except for the random initial weights. **(B)** The final classification confusion matrix, the result is the average of the four trials in **(A)**.

f_i is the spike frequency of the corresponding subscript coding neuron, a_i is the distance from the corresponding subscript neuron to the starting point (the gray vertical line in **Figure 5**), and the remaining variables are as circled in **Figure 5** shown. v is the input value, w is one-half of the length of the bottom side of the encoding triangle, and h is the highest encoding spike frequency, that is, the height of the bottom side of the encoding triangle in **Figure 5**. Both w and h are hyperparameters. With the input value as the center, the closer the neuron is to the center of the input value, the higher the neuron spike frequency is.

Connections from the encoding layer to the output layer are fully-connected and all synapses are trainable. ALSA is implemented for training between the input layer and the output layer. The three neurons in the output layer correspond to three categories. During training, each sample is kept in the network for 200 milliseconds, during which the teacher layer induces the output layer neuron of the corresponding category to generate spikes. There is an interval of 50 milliseconds between the two samples, during which the coding layer neurons do not generate spikes, which resets the neuron state.

Due to the small number of samples in the IRIS dataset, the hold-out method is implemented to achieve cross-validation during training and validation. Divide each category of data elements into five groups, so that there are 30 data elements in each group, 10 data elements in each category. After dividing the data into five groups, four groups were used for training, and the remaining one group was used for validation. After the training is completed, the remaining group retained in advance is used as the test set to evaluate the network performance.

Figure 6 is the result of the IRIS classification network. The detailed network parameters are as follows: 4 groups of coding neurons, 12 in each group, 48 in total, h = 20hz, w = 2. The synapse weights from the coding layer to the output layer are evenly distributed from 0.2 to 0.3, $\eta = 0.015$, $\alpha = -0.1$, $\beta = 1$, $\tau_p = 50$, and the teacher spike frequency is 20 hz.

As can be seen from the figure, ALSA can effectively realize the classification of the IRIS dataset. Due to the single-layer structure, there is a certain deviation in the distinction between Versicolor and Virginia. The accuracy of network classification reaches about 95% after learning all training samples (cross-validation, the number of the training set is 120) once, but there are some differences in the four trials. As the number of iterations increases, the accuracy of the four trials gradually converges to a similar value, and the average accuracy of four trials is 95.7%.

MNIST Classification

The MNIST (LeCun et al., 1998) dataset is widely used in the performance test of various neural networks. The MNIST dataset contains ten classes of handwritten digits from 0 to 9, including a total of 60,000 samples in the training set and 10,000 samples in the test set.

Figure 7 is a structural diagram of the MNIST classification network, including four layers: input layer, features layer, output layer, and teacher layer. The input layer is fully connected with the features layer after the input data is encoded. The encoding method adopts time encoding, that is, the larger the corresponding pixel value, the earlier the neuron spike signal will be emitted, and the spike will not be emitted if the value is lower than the encoding threshold which is a hyper-parameter. Each picture is kept in the network for 200 milliseconds, and there is an interval of 50 milliseconds between two pictures, during which

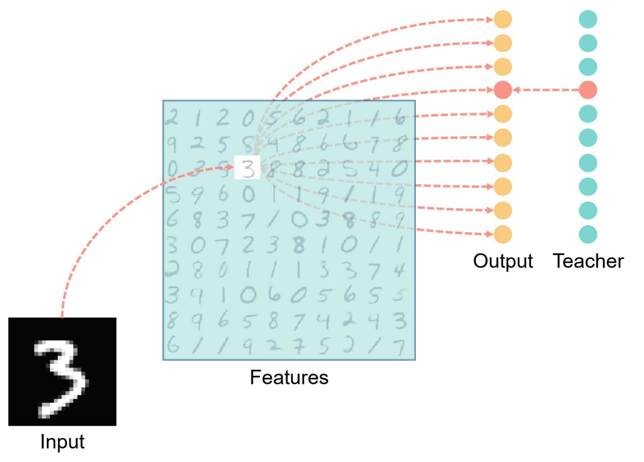


FIGURE 7 | MNIST handwritten digit classification network structure diagram. There are four layers including input, features, output, and teacher. The light blue color blocks in the features layer in the figure indicate inhibition, and the red neurons in the output layer and teacher layer indicate that the neurons are in an active state (that is, there is a spike signal for the time the input vector is presented to the network). The red dashed line indicates the excitatory connection between neurons.

no spike is generated in the input layer, which is for resetting the network state.

Each neuron in the features layer has inhibitory synaptic connections to all neurons in the same layer except itself, which is for achieving lateral inhibition to prevent repeated learning. Also, the features layer neurons are fully connected with the output layer neurons.

Teacher layer neurons are connected one-to-one with corresponding output layer neurons to induce output layer neurons to generate spikes.

The input layer to the features layer is mainly based on unsupervised learning, using the rules of synaptic plasticity described in 6, 7. The training method is similar to Diehl's work (Diehl and Cook, 2015). For every input image, one neuron in the features layer is activated first and the others are laterally inhibited. The weights between the input layer and the features layer change according to the modified STDP rules. From the features layer to the output layer, ALSA is implemented for supervised learning under the guidance of the teacher layer to realize the mapping of handwritten digitals from the features layer to the output layer.

The network is trained using a layer-by-layer training method, that is, the training between the input layer and the features layer is finished after certain samples, and then the training between the features layer and the output layer is performed.

Figure 8 is the final result of the MNIST classification network. The detailed network parameters are as follows: input layer 28×28, consistent with the sample resolution in the MNIST dataset, feature layer 20×20, output layer 1×10, and teacher layer 1×10. The encoding threshold is 0.3. The synapse weights from the input layer to the feature layer are uniformly distributed from 0.01 to 0.11, $\eta = 0.015$, $\alpha = -0.3$, $\beta = 1.3$, and τ_p 20. The synapse weights from the characteristic layer to the output layer are uniformly distributed from 0.1 to 0.2, $\eta = 0.001$, $\alpha = -0.003$, $\beta = 2$, and $\tau_p = 100$. The teacher spike frequency is 20 hz.

It can be seen from **Figure 8A** that when there are 400 neurons in the feature layer, as the number of training samples increases, the weights from the input layer to the features layer gradually show the sample pattern clearly. However, there are still some cases where the weight distribution from the input layer to the features layer is fuzzy, or multiple samples are superimposed which is mainly because of lack of learning or small among-class gaps. Samples with smaller intra-class gaps (such as 0, 2, 7) can show clearer contours with fewer training samples. The accuracy of the supervised learning part reaches a higher accuracy after training the complete training samples once, gradually converges in the subsequent training, and finally reaches 88.53%, which has a certain gap compared with the mainstream MNIST classification network. To verify the effectiveness of the supervised learning rules proposed in this paper, it will be compared with the widely cited experimental results of Diehl. It is worth noting that Diehl uses the unsupervised + statistical method in his paper (Diehl and Cook, 2015). In Diehl's results, when the size of the unsupervised learning output layer is 400 and 1600, the corresponding classification accuracy is 87 and 91.9%, respectively. For the convenience of comparison, this article chooses the network models with 400 and 1600 neurons in the features layer, respectively and the results are obtained in **Figures 8B,C**. After multiple training runs, the average results are obtained as follows: the classification accuracy of 400 neurons is 88.53%, and that of 1600 excitative neurons is 91.58% ($\eta = 0.065$, $\alpha = -0.2$, $\beta = 1.3$, and $\tau_p = 20$). This result indicates that the supervised learning using ALSA in this network can achieve performance similar to the statistical methods of Diehl. It can be seen from the above two classification experiments that ALSA can realize pattern recognition and classification, and it is proved to be working. The feasibility of the ALSA learning method is preliminarily verified here, and more different experiments are needed to improve it in the future.

DISCUSSION

Biologically Plausible

The ALSA supervised learning method proposed in this paper is based on associative learning. The synaptic long-term plasticity rule is also based on classic STDP after appropriate modifications. The main contents are supported by corresponding neuroscience-related experiments or phenomena (Krug et al., 1985; Christie et al., 1994; McSweeney and Murphy, 2014). By inducing the target neuron to emit spikes, the connection weights between the neuron corresponding to the current spike pattern and the target neuron are strengthened, and others are weakened, which is consistent with the Hebb rule "neurons that fire together, wire together." Moreover, the implementation method of supervised learning is similar to the process of animal training based on associative learning, and the latter has been proved to be an effective animal training method in a large number of experiments and practices.

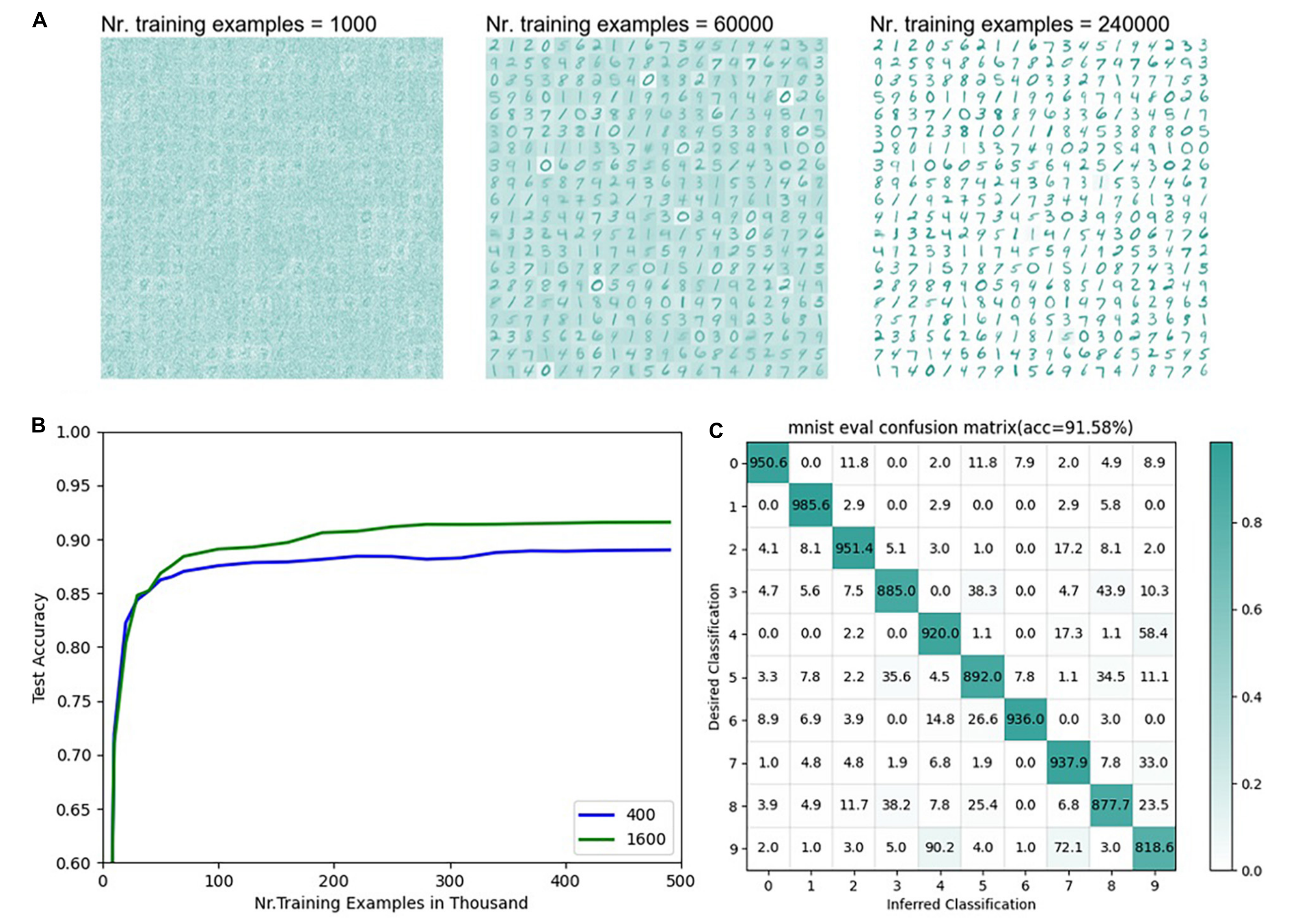


FIGURE 8 | Results of MNIST classification **(A)** The distribution of weights under different numbers of training samples from the input layer to the feature layer with 400 neurons in the feature layer (*n* = 400). **(B)** The accuracy of the supervised part of the network varies with the number of training samples. The blue line represents the accuracy of the model with different numbers of training samples when the number of neurons in the feature layer is 400 (*n* = 400). The green line represents the test accuracy changing with the number of training samples when the number of neurons in the feature layer is 1600 (*n* = 1600). **(C)** The final classification confusion matrix with 1600 neurons in the feature layer.

Therefore, ALSA is physiologically reasonable and has strong physiological plausibility.

In this paper, a supervised learning algorithm for spiking neural networks based on associative learning named ALSA was proposed. Compared to other supervised learning algorithms for SNN, ALSA is based on modified STDP, thus ALSA is more biologically plausible than most other training algorithms. In addition, the modified STDP used in ALSA shows more similarities to the Hebb rule and actual experiment results in neuroscience. Unsupervised learning is powerful in SNNs due to its great ability in spatial-temporal feature extraction called coincidence detection. Normally, coincidence detection is based on STDP or its modification. While none of the existing supervised learning algorithms excepting ALSA are based on STDP, which make it impossible to realize supervised and unsupervised learning algorithm in the same layer. ALSA shows more compatibility with unsupervised learning algorithms. The key difference of ALSA to unsupervised learning is the teacher signal, without the teacher signal, ALSA works as a normal unsupervised learning algorithm, with the teacher signal, ALSA works as a biologically plausible supervised learning algorithm. Thus, ALSA can make full use of the power of unsupervised learning and supervised learning.

Compatibility

At present, many SNN supervised learning algorithms have been able to achieve good training effects and performance. But most of the methods are incompatible with unsupervised learning methods. The current unsupervised learning method of SNNs is more reasonable in principle, with stronger physiological plausibility and rationality, and the unsupervised learning method of SNNs has also been proved to have strong feature extraction capabilities, especially the spatial-temporal features extraction ability (Dennis et al., 2015; Masquelier and Kheradpisheh, 2018; Wu et al., 2018). This is an ability that the traditional ANN networks do not have. It is also the place where SNNs have unique advantages. Therefore, it is important to give full play to the unsupervised learning ability of SNNs. ALSA is based on the improved STDP. As shown in the MNIST classification experiment above, this synaptic plasticity rule can realize SNNs unsupervised learning well, that is, the same rule can realize unsupervised learning and supervised learning at the same time. And through appropriate adjustments, in theory, unsupervised learning and supervised learning can be realized in the same layer. Unsupervised learning and supervised learning can be performed at different phases for better learning. Therefore, ALSA has strong compatibility with SNN unsupervised learning, which greatly expands the application scope of ALSA.

Trainable Layers

Due to the characteristics of ALSA, it can only be used for single-layer network training. However, as mentioned above, ALSA has strong compatibility with SNN unsupervised learning. Therefore, we can make full use of the powerful unsupervised learning ability of SNNs to build multi-layer unsupervised + single-layer supervised SNNs to make up for the shortcomings of only a single-layer training. Also, the supervised method can be used to some key layers in the network by inducing neurons in these layers, to realize multi-layer unsupervised + multi-layer supervised SNNs as a whole.

Performance

In the experimental part, two experiments, training with IRIS dataset and MNIST dataset are conducted, respectively, and both achieved satisfactory results. The average accuracy of the four training trials of the IRIS dataset was 95.7%. When the number of neurons in the feature layer was 1600, the classification accuracy of the MNIST dataset achieved 91.58%, when training with the proposed ALSA rule. Although the performance achieved by the SNN network has a certain gap compared with the current mainstream ANN networks based on error backpropagation or other classifiers. However, the results of these two experiments prove the feasibility of ALSA to a large extent. In the future, combined with the above-mentioned multi-layer unsupervised and multi-layer supervised methods, with a large network scale, the performance of ALSA has a lot of room for development. Right now, it is still a challenge for us to increase the network scale and improve the recognition accuracy. For the MNIST dataset, there are over 60,000 pictures, and it takes several days to several weeks to train once after further increasing the network scale. In the future, we plan to improve the speed by optimizing the computing framework, such as using multithread or GPU acceleration.

Robustness

The existence of dynamic membrane potential can prevent some neurons from over-emitting spikes while other neurons do not emit spikes, which will lead to the problem of "winner takes all," making all neurons have a relatively fair environment to learn spike patterns and improve the efficiency of feature learning. In addition, because of the dynamic threshold, the spiking frequency of teacher neurons has little influence on associative learning and supervised learning. The learning performance of the network is not sensitive to the teacher spiking frequency. According to the results of the two classification experiments, the performance of the final network tends to be stable, indicating that ALSA can control the state of the neural network in a relatively stable state and has high robustness.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

LM proposed the idea and the detailed implementation methods of ALSA. GW designed and implemented the two confirmatory experiments. EL and MZ participated in the revision and supplementary experiment of the manuscript. All authors took part in the writing of the manuscript and discussion of the whole process.

FUNDING

This work was financed by the National Key R&D Program of China (2020YFD1100201).

ACKNOWLEDGMENTS

We would like to thank all the members of FutureX LAB of Southeast University for their help and support, especially for the enlightening discussion.

REFERENCES

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE. Trans. Comput. Des. Integr. Circuits Syst.* 34, 1537–1557. doi: 10.1109/tcad.2015.2474396

Bengio, Y. (2015). "Difference target propagation," in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (bilbao), 1–17.

Bohte, S. M., Kok, J. N., and La Poutré, J. A. (2000). "SpikeProp: backpropagation for networks of spiking neurons," in *Proceedings of the European Symposium on Artificial Neural Networks, ESANN* (Bruges, Belgium), 17–37. doi: 10.1016/s0925-2312(01)00658-0

Caporale, N., and Dan, Y. (2008). Spike timing-dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.* 31, 25–46. doi: 10.1146/annurev.neuro.31.060407.125639

Christie, B. R., Kerr, D. S., and Abraham, W. C. (1994). Flip side of synaptic plasticity: Long-term depression mechanisms in the hippocampus. *Hippocampus* 4, 127–135. doi: 10.1002/hipo.45004 0203

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/mm.2018.112130359

Dennis, J., Tran, H. D., and Li, H. (2015). "Combining robust spike coding with spiking neural networks for sound event classification," in *Proceedings of the ICASSP, IEEE International Conference on Acoustics, Speech Signal Process* (Singapore), 176–180. doi: 10.1109/ICASSP.2015.7177955

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN)* (New York, NY), 1–8.

Dua, D., and Graff, C. (2017). *{UCI} Machine Learning Repository*. Available online at: http://archive.ics.uci.edu/ml (accessed July 30, 2018).

Ponulak, F., and Kasiński, A. (2010). Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. *Neural Computat.* 22, 467–510. doi: 10.1162/neco.2009.11-08-901

Gerstner, W. (1995). Time structure of the activity in neural network models. *Phys. Rev. E, Stat. phys., Plasmas Fluids Relat. Interdiscip. Topics* 51, 738–758. doi: 10.1103/physreve.51.738

Gu, P., Xiao, R., Pan, G., and Tang, H. (2019). "STCA: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks," in *Proceedings of the IJCAI Int. Jt. Conf. Artif. Intell*, (New York, NY), 1366–1372. doi: 10.24963/ijcai.2019/189

Gütig, R. (2016). Spiking neurons can discover predictive features by aggregate-label learning. *Science* 351:aab4113. doi: 10.1126/science.aab4113

Gütig, R., and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nat. Neurosci.* 9, 420–428. doi: 10.1038/nn1643

Hao, Y., Huang, X., Dong, M., and Xu, B. (2020). A biologically plausible supervised learning method for spiking neural networks using the symmetric STDP rule. *Neural Netw.* 121, 387–395. doi: 10.1016/j.neunet.2019.09.007

Hodgkin, A. L., and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* 117, 500–544. doi: 10.1113/jphysiol.1952.sp004764

Jang, H., Simeone, O., Gardner, B., and Gruning, A. (2019). An introduction to probabilistic spiking neural networks: probabilistic models, learning rules, and applications. *IEEE. Signal Process. Mag.* 36, 64–77. doi: 10.1109/msp.2019.2935234

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP- based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Koch, C., and Segev, I. (1998). *Methods in Neuronal Modeling: from Ions to Networks*. Cambridge, CA: MIT press.

Krug, M., Müller-Welde, P., Wagner, M., Ott, T., and Matthies, H. (1985). Functional plasticity in two afferent systems of the granule cells in the rat dentate area: frequency-related changes, long-term potentiation and heterosynaptic depression. *Brain Res.* 360, 264–272. doi: 10.1016/0006-8993(85)91242-9

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1, 541–551. doi: 10.1162/neco.1989.1.4.541

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE.* 86, 2278–2324. doi: 10.1109/5.726791

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* 14:119. doi: 10.3389/fnins.2020.00119

Lee, D. H., Zhang, S., Fischer, A., and Bengio, Y. (2015). Difference target propagation. *Lect. Notes Comput. Sci. (Lect. Notes Artif. Intell. Lect. Notes Bioinform.)* 9284, 498–515.

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. (2020). Backpropagation and the brain. *Nat. Rev. Neurosci.* 21, 335–346. doi: 10.1038/s41583-020-0277-3

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/s0893-6080(97)00011-7

Masquelier, T., and Kheradpisheh, S. R. (2018). Optimal localist and distributed coding of spatiotemporal spike patterns through STDP and coincidence detection. *Front. Comput. Neurosci.* 12:74. doi: 10.3389/fncom.2018.00074

McSweeney, F. K., and Murphy, E. S. (2014). *The Wiley Blackwell Handbook of Operant and Classical Conditioning*. Hoboken, NJ: John Wiley & Sons.

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., and Masquelier, T. (2019). Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern Recognit.* 94, 87–95. doi: 10.1016/j.patcog.2019.05.015

Mo, L., Chen, X., and Wang, G. (2021). EDHA: Event-driven high accurate simulator for spike neural networks. *Electronics* 10:2281. doi: 10.3390/electronics10182281

O'Shea, K., and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv* [Preprint] arXiv: 1511.08458,

Pan, Z., Chua, Y., Wu, J., Zhang, M., Li, H., and Ambikairajah, E. (2020). An efficient and perceptually motivated auditory neural encoding and decoding algorithm for spiking neural networks. *Front. Neurosci.* 13:1420. doi: 10.3389/fnins.2019.01420

Pavlov, P. I. (2010). Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex. *Ann. Neurosci.* 17:136.

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Pérez-Carrasco, J. A., Zhao, B., Serrano, C., Acha, B., Serrano-Gotarredona, T., Chen, S., et al. (2013). Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets. *IEEE. Trans. Pattern Anal. Mach. Intell.* 35, 2706–2719. doi: 10.1109/TPAMI.2013.71

Pfister, J. P., Barber, D., and Gerstner, W. (2003). Optimal hebbian learning: a probabilistic point of view. *Lect. Notes Comput. Sci. (Lect. Notes Artif. Intell. Lect. Notes Bioinform.)* 2714, 92–98. doi: 10.1007/3-540-44989-2_12

Legenstein, R., Naeger, C., and Maass, W. (2005). What Can a Neuron Learn with Spike-Timing-Dependent Plasticity? *Neural Computat.* 17, 2337–2382. doi: 10.1162/0899766054796888

Ponulak, F. (2005). *ReSuMe-New Supervised Learning Method for Spiking Neural Networks*. Poznoń University of Technology: Institute of Control and Information Engineering.

Querlioz, D., Bichler, O., Dollfus, P., and Gamrat, C. (2013). Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE. Trans. Nanotechnol.* 12, 288–295. doi: 10.1109/tnano.2013.2250995

Rawat, W., and Wang, Z. (2017). Deep convolutional neural networks for image classification: a comprehensive review. *Neural Comput.* 29, 2352–2449. doi: 10.1162/NECO_a_00990

Rueckauer, B., Lungu, I. A., Hu, Y., Pfeiffer, M., and Liu, S. C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11:682. doi: 10.3389/fnins.2017.00682

Shen, J., Ma, D., Gu, Z., Zhang, M., Zhu, X., Xu, X., et al. (2016). Darwin: a neuromorphic hardware co-processor based on Spiking Neural Networks. *Sci. China Inf. Sci.* 59, 1–5. doi: 10.1007/s11432-015-5511-7

Shrestha, A., Ahmed, K., and Wang, Y. (2017). "Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning," in *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK: IEEE, Institute of Electrical and Electronics Engineers).

Sjöström, P. J., Turrigiano, G. G., and Nelson, S. B. (2001). Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron* 32, 1149–1164. doi: 10.1016/s0896-6273(01)00542-6

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2019). Deep learning in spiking neural networks. *Neural Netw.* 111, 47–63. doi: 10.1016/j.neunet.2018.12.002

Wade, J. J., McDaid, L. J., Santos, J. A., and Sayers, H. M. (2010). SWAT: a spiking neural network training algorithm for classification problems. *IEEE. Trans. Neural netw.* 21, 1817–1830. doi: 10.1109/TNN.2010.2074212

Watt, A. J., and Desai, N. S. (2010). Homeostatic plasticity and STDP: Keeping a neuron's cool in a fluctuating world. *Front. Synaptic Neurosci.* 2:5. doi: 10.3389/fnsyn.2010.00005

Wu, J., Chua, Y., Zhang, M., Li, H., and Tan, K. C. (2018). A spiking neural network framework for robust sound classification. *Front. Neurosci.* 12:836. doi: 10.3389/fnins.2018.00836

Xu, Q., Peng, J., Shen, J., Tang, H., and Pan, G. (2020). Deep CovDenseSNN: A hierarchical event-driven dynamic framework with spiking neurons in noisy environment. *Neural Netw.* 121, 512–519. doi: 10.1016/j.neunet.2019.08.034

Xie, X., Qu, H., Yi, Z., and Kurths, J. (2016). Efficient training of supervised spiking neural network via accurate synaptic-efficiency adjustment method. *IEEE. Trans. Neural Netw. Learn. Syst.* 28, 1411–1424.

Yu, Q., Li, S., Tang, H., Wang, L., Dang, J., and Tan, K. C. (2020). Toward efficient processing and learning with spikes: new approaches for multispike learning. *IEEE. Trans. Cybern.* 1–13. doi: 10.1109/TCYB.2020.2984888

Yu, Q., Tang, H., Tan, K. C., and Li, H. (2013). Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns. *PLoS One* 8:e78318. doi: 10.1371/journal.pone.0078318

Zhang, M., Wu, J., Chua, Y., Luo, X., Pan, Z., Liu, D., et al. (2019). Mpd-al: an efficient membrane potential driven aggregate-label learning algorithm for spiking neurons. *Proc. AAAI. Conf. Artif. Intell.* 33, 1327–1334. doi: 10.1609/aaai.v33i01.33011327

Zirkle, J., and Rubchinsky, L. L. (2020). Spike-Timing Dependent Plasticity Effect on the Temporal Patterning of Neural Synchronization. *Front. Comput. Neurosci.* 14:52. doi: 10.3389/fncom.2020.00052

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Mo, Wang, Long and Zhuo. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

ORIGINAL RESEARCH
published: 07 April 2022
doi: 10.3389/fnins.2022.815258

ACE-SNN: Algorithm-Hardware Co-design of Energy-Efficient & Low-Latency Deep Spiking Neural Networks for 3D Image Recognition

Gourav Datta, Souvik Kundu, Akhilesh R. Jaiswal and Peter A. Beerel*

Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA, United States

High-quality 3D image recognition is an important component of many vision and robotics systems. However, the accurate processing of these images requires the use of compute-expensive 3D Convolutional Neural Networks (CNNs). To address this challenge, we propose the use of Spiking Neural Networks (SNNs) that are generated from iso-architecture CNNs and trained with quantization-aware gradient descent to optimize their weights, membrane leak, and firing thresholds. During both training and inference, the analog pixel values of a 3D image are directly applied to the input layer of the SNN without the need to convert to a spike-train. This significantly reduces the training and inference latency and results in high degree of activation sparsity, which yields significant improvements in computational efficiency. However, this introduces energy-hungry digital multiplications in the first layer of our models, which we propose to mitigate using a processing-in-memory (PIM) architecture. To evaluate our proposal, we propose a 3D and a 3D/2D hybrid SNN-compatible convolutional architecture and choose hyperspectral imaging (HSI) as an application for 3D image recognition. We achieve overall test accuracy of 98.68, 99.50, and 97.95% with 5 time steps (inference latency) and 6-bit weight quantization on the Indian Pines, Pavia University, and Salinas Scene datasets, respectively. In particular, our models implemented using standard digital hardware achieved accuracies similar to state-of-the-art (SOTA) with ∼560.6× and ∼44.8× less average energy than an iso-architecture full-precision and 6-bit quantized CNN, respectively. Adopting the PIM architecture in the first layer, further improves the average energy, delay, and energy-delay-product (EDP) by 30, 7, and 38%, respectively.

Keywords: hyperspectral images, spiking neural networks, quantization-aware, gradient descent, processing-in-memory

OPEN ACCESS

Edited by:
Bipin Rajendran,
King's College London,
United Kingdom

Reviewed by:
Chankyu Lee,
Intel, United States
Brajesh Kumar Kaushik,
Indian Institute of Technology
Roorkee, India

****Correspondence:***
Gourav Datta
gdatta@usc.edu

Specialty section:
This article was submitted to
Neuromorphic Engineering,
a section of the journal
Frontiers in Neuroscience

Received: *15 November 2021*
Accepted: *03 March 2022*
Published: *07 April 2022*

Citation:
Datta G, Kundu S, Jaiswal AR and Beerel PA (2022) ACE-SNN: Algorithm-Hardware Co-design of Energy-Efficient & Low-Latency Deep Spiking Neural Networks for 3D Image Recognition.
Front. Neurosci. 16:815258.
doi: 10.3389/fnins.2022.815258

1. INTRODUCTION

3D image classification is an important problem, with applications ranging from autonomous drones to augmented reality. 3D content creation has been gaining momentum in the recent past and the amount of information in the form of 3D input data becoming publicly available is steadily increasing. In particular, hyperspectral imaging (HSI), which extracts rich spatial-spectral information about the ground surface, has shown immense promise in remote sensing

(Chen et al., 2014), and thus, has become an important application for 3D image recognition. HSI is currently used in several workloads ranging from geological surveys (Wan et al., 2021), to the detection of camouflaged vehicles (Papp et al., 2020). In hyperspectral images (HSIs), each pixel can be modeled as a high-dimensional vector where each entry corresponds to the spectral reflectivity of a particular wavelength (Chen et al., 2014), and constitutes the 3^{rd} dimension of the image. The goal of the classification task is to assign a unique semantic label to each pixel (Zheng et al., 2020). For HSI classification, several spectral feature-based methods have been proposed, including support vector machine (Melgani and Bruzzone, 2004), random forest (Pal, 2003), canonical correlation forest (Xia et al., 2017), and multinomial logistic regression (Krishnapuram et al., 2005). However, these spectral-spatial feature extraction methods rely on hand-designed descriptions, prior information, and empirical hyperparameters (Chen et al., 2014).

Lately, convolutional neural networks (CNNs), consisting of a series of hierarchical filtering layers for global optimization have yielded higher accuracy than the hand-designed features (Krizhevsky, 2012), and have shown promise in multiple applications including image classification (He et al., 2016), object detection (Ren et al., 2017), semantic segmentation (He et al., 2018), and depth estimation (Repala and Dubey, 2019). The 2D CNN stacked autoencoder (Chen et al., 2014) was the first attempt to extract deep features from its compressed latent space to classify HSIs. To extract the spatial-spectral features jointly from the raw HSI, researchers proposed a 3D CNN architecture (Ben Hamida et al., 2018), which achieved SOTA classification results. In Lee and Kwon (2017), Roy et al. (2020), and Luo et al. (2018) successfully created multiscale spatiospectral relationships using 3D CNNs and fused the features using a 2D CNN to extract more robust representation of spectral–spatial information. However, compared to 2D CNNs used to classify traditional RGB images, multi-layer 3D CNNs require significantly higher power and energy costs (Li et al., 2016). A typical hyperspectral image cube consists of several hundred spectral frequency bands that, for target tracking and identification, require real time on-device processing (Hien Van Nguyen et al., 2010). This desire for HSI sensors operating on energy-limited devices motivates exploring alternative lightweight classification models.

In particular, low-latency spiking neural networks (SNNs) (Pfeiffer and Pfeil, 2018), illustrated in **Figure 1**, have gained attention because they are more computational efficient than CNNs for a variety of applications, including image analysis. To achieve this goal, analog inputs are first encoded into a sequence of spikes using one of a variety of proposed encoding methods, including rate coding (Diehl et al., 2016; Sengupta et al., 2019), direct coding (Rathi and Roy, 2020), temporal coding (Comsa et al., 2020), rank-order coding (Kheradpisheh and Masquelier, 2020), phase coding (Kim et al., 2018), and other exotic coding schemes (Almomani et al., 2019; Datta et al., 2021). Among these, direct coding have shown competitive performance on complex tasks (Diehl et al., 2016; Sengupta et al., 2019) while others are either limited to simpler tasks such as learning the XOR function and classifying MNIST images or require a large number of spikes for inference.

In addition to accommodating various forms of encoding inputs, supervised learning algorithms for SNNs have overcome various roadblocks associated with the discontinuous derivative of the spike activation function (Lee et al., 2016; Wu et al., 2019). In particular, recent works have shown that SNNs can be efficiently converted from artifical neural networks (ANNs) by approximating the activation value of ReLU neurons with the firing rate of spiking neurons (Sengupta et al., 2019). Low-latency SNNs trained using ANN-SNN conversion, coupled with supervised training, have been able to perform at par with ANNs in terms of classification accuracy in traditional image classification tasks (Rathi and Roy, 2020; Datta and Beerel, 2021; Kundu et al., 2021c). Consequently, SNNs have lower compute cost than their non-spiking CNN counterparts. This is particularly useful in 3D CNNs which have higher arithmetic intensity (the ratio of floating point operations to accessed bytes) than 2D CNNs. This motivates this work which explores the effectiveness of SNNs converted from 3D CNNs for HSI classification.

To improve energy efficiency, model compression techniques, such as pruning (Han et al., 2015a), can be adapted to CNN/SNN models for HSI classification. Unstructured pruning can lead to significant parameter reduction ($>10\times$ for both 2D CNN and SNN models for traditional vision tasks (Kundu et al., 2021a,b). However, unstructured pruning typically requires specialized ASIC/FPGA hardware to reap energy-savings benefits and does not lead to savings when implemented on standard GPUs. On the other hand, structured pruning is compatible with general-purpose CPU/GPU hardware, but is unable to remove a large number of weights while maintaining accuracy, particularly for our proposed compact CNN architectures.

The energy efficiency of SNN inference can also be improved by using integer or fixed-point computational units implemented either as CMOS-based digital accumulators or memory array based processing-in-memory (PIM) accelerators. Previous research (Rathi et al., 2017; Sulaiman et al., 2020) have proposed post-training SNN quantization tailored toward unsupervised learning, which has not been shown to scale to complex vision tasks without requiring high precision (≥ 8 bits). This work addresses this gap by proposing a quantization-aware SNN training algorithm that requires only 5 time steps with 6-bit weights, yielding a $2\times$ reduction in bit width compared to a post-training quantization baseline that yields similar accuracy.

The first layer in direct coded SNNs still requires multiply-and-accumulates (MAC), which are significantly more expensive than the accumulates required in a spiking layer. To mitigate this issue, we propose an SRAM-based processing-in-memory (PIM) architecture to process the first layer, which cannot only reduce the CMOS-based digital MAC cost, but also address the Von-Neumann bottleneck by eliminating data movement between the memory and the convolutional processing elements. Moreover, the relatively lower parameter count of the first 3D CNN layer ensures that we can perform the whole convolution in a single memory array, thereby improving area efficiency. The remaining layers, which involves cheap accumulates and threshold comparisons, are implemented with highly parallel programmable digital architectures, as

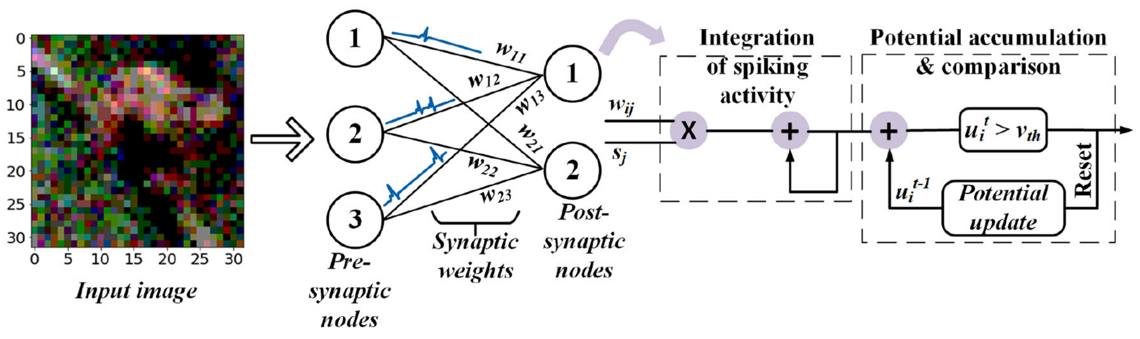


FIGURE 1 | Feedforward fully-connected SNN architecture with integrate and fire (IF) spiking dynamics.

used in Chen et al. (2022) and Park et al. (2019). Our proposed hardware-software co-design results in 700.7× and 1.79× improvements in energy-delay product (EDP) for HSI classification compared to digital implementatons with standard ANNs and SNNs.

In summary, this paper provides the following contributions:

- We analyse the arithmetic intensities of 3D and 2D CNNs, and motivate the use SNNs to address the compute energy bottleneck faced by 3D CNN layers used for HSI classification.
- We propose a hybrid training algorithm that first converts an ANN for HSI classification to an iso-architecture SNN, and then trains the latter using a novel quantization-aware spike timing dependent backpropagation (Q-STDB) algorithm that yields low latency (5 time steps) and low bit width (6-bits).
- We propose two compact convolutional architectures for HSI classification that can yield classification accuracies similar to state-of-the-art (SOTA) and are compatible with our ANN-SNN conversion framework.
- We propose a novel circuit framework and its associated energy models for energy-efficient hardware implementation of the SNNs obtained by our training framework, and benchmark the EDP gains compared to standard ANNs. Our experimental results reveal that the SNNs trained for HSI classification offer four and two orders of magnitude improvement in energy consumption compared to full-precision and iso-precision ANNs.

The remainder of this paper is structured as follows. In Section 2 we present necessary background and related work. Section 3 describes our analysis of arithmetic intensities of 3D and 2D CNNs, and highlights the motivation of using SNNs for 3D imaging. Sections 4, 5 discusses our proposed quantization-aware SNN training method and a PIM architecture to improve the energy efficiency of our proposed SNN models during inference. Section 6 focuses on our proposed network architectures, benchmark datasets, and our training details. We present detailed experimental results and analysis in Section 7. Finally, the paper concludes in Section 8.

2. BACKGROUND

2.1. SNN Modeling

The spiking dynamics of a neuron are generally modeled using either the Integrate-and-Fire (IF) (Burkitt, 2006) or Leaky-Integrate-and-Fire (LIF) model (Lee et al., 2020), where the activity of pre-synaptic neurons modulates the membrane potential of postsynaptic neurons. The membrane potential of a IF neuron does not change during the time period between successive input spikes while in the LIF model, the membrane potential leaks at a constant rate. In this work, we adopt the LIF model in our proposed training technique, as the leak term improves the bio-plausibility and robustness to noisy spike-inputs (Chowdhury et al., 2020).

The LIF is probably one of the earliest and simplest spiking neuron models, but it is still very popular due to the ease with which it can be analyzed and simulated. In its simplest form, a neuron is modeled as a "leaky integrator" of its input $I(t)$:

$$\tau_m \frac{\partial v}{\partial t} = -v(t) + R \cdot I(t) \tag{1}$$

where $v(t)$ represents the membrane potential of the neuron at time t, τ_m is the membrane time constant and R is the membrane resistance. When $v(t)$ reaches a certain firing threshold, it is instantaneously reset to a lower value v_r (reset potential), the neuron generates a spike, and the leaky integration process described by Equation 1 starts afresh with the initial value v_r. However, due to its continuous representation, Equation (1) is not suitable for implementations in popular Machine Learning (ML) frameworks (e.g., Pytorch). Hence, we convert Equation (1) into an iterative discrete-time version, as shown in Eqs. 2 and 3, within which spikes in a particular layer l, denoted as o_l^t, are characterized as binary values (1 represents the presence of a spike) (Rathi et al., 2020). The pre-spikes in the $(l-1)^{th}$ layer, o_{l-1}^t are modulated by the synaptic weights $\hat{w}_l$ to be integrated as the current influx in the membrane potential u_l^t that decays with a leak factor λ_l.

$$\boldsymbol{u}_l^t = \lambda_l \boldsymbol{u}_l^{t-1} + \hat{\boldsymbol{w}}_l \boldsymbol{o}_{l-1}^t - v_l \boldsymbol{o}_l^t \tag{2}$$

$$\boldsymbol{z}_l^t = \frac{\boldsymbol{u}_l^t}{v_l} - 1, \quad \boldsymbol{o}_l^t = \begin{cases} 1, & \text{if } \boldsymbol{z}_l^t > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The third term in Equation (2) exhibits soft reset by setting the reset potential to the threshold v_l (instead of 0) i.e., reducing the membrane potential u_l by v_l at time step t, if an output spike is generated at the t^{th} time step. As shown in Rathi et al. (2020), soft reset enables each spiking neuron to carry forward the surplus potential above the firing threshold to the subsequent time step (Rathi et al., 2020), thereby minimizing the information loss.

2.2. SNN Training Techniques

Recent research on training supervised deep SNNs can be primarily divided into three categories: 1) ANN-SNN conversion-based training, 2) Spike timing dependent backpropagation (STDB), and 3) Hybrid training.

2.2.1. ANN-SNN Conversion

ANN-SNN conversion involves copying the SNN weights from a pretrained ANN model and estimating the threshold values in each layer by approximating the activation value of ReLU neurons with the firing rate of spiking neurons (Cao et al., 2015; Diehl et al., 2015; Rueckauer et al., 2017; Hu et al., 2018; Sengupta et al., 2019). The ANN model is trained using standard gradient descent based methods and helps an iso-architecture SNN achieve impressive accuracy in image classification tasks (Rueckauer et al., 2017; Sengupta et al., 2019). However, the SNNs resulting from these conversion algorithms require an order of magnitude more time steps compared to other training techniques (Sengupta et al., 2019). In this work, we use ANN-SNN conversion as an initial step in Q-STDB because it is of relatively low complexity and yields high classification accuracy on deep networks as shown in Section 7.5.3.

2.2.2. STDB

The thresholding-based activation function in the IF/LIF model is discontinuous and non-differentiable, which poses difficulty in training SNNs with gradient-descent based learning methods. Consequently, several approximate training methodologies have been proposed (Lee et al., 2016; Panda and Roy, 2016; Bellec et al., 2018; Neftci et al., 2019), where the spiking neuron functionality is either replaced with a differentiable model or the real gradients are approximated as surrogate gradients. However, the backpropagation step requires these gradients to be integrated over all the time steps required to train the SNN, which significantly increases the memory requirements.

2.2.3. Hybrid Training

A recent paper (Rathi et al., 2020) proposed a hybrid training technique where the ANN-SNN conversion is performed as an initialization step and is followed by an approximate gradient descent algorithm. The authors observed that combining the two training techniques helps SNNs converge within a few epochs while requiring fewer time steps. In Rathi and Roy (2020) extended the above hybrid learning approach by training the membrane leak and the firing threshold along with other network parameters (weights) via gradient descent. Moreover, Rathi and Roy (2020) applied direct-input encoding where the pixel intensities of an image are applied into the SNN input layer as fixed multi-bit values each time step to reduce the number of time steps needed to achieve SOTA accuracy by an order of magnitude. Though the first layer now requires MACs, as opposed to cheaper accumulates in the remaining layers, the overhead is negligible for deep convolutional architectures (Rathi and Roy, 2020). This work extends these hybrid learning techniques by using a novel representation of weights for energy efficiency and performing quantization-aware training in the SNN domain.

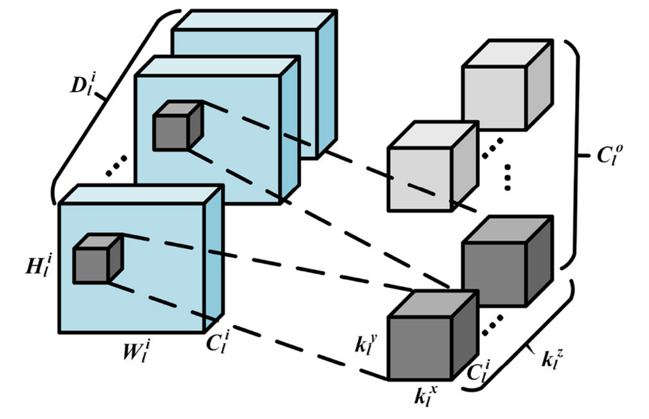


FIGURE 2 | Illustration of the 3D convolution operation.

3. 3D VS 2D CNNS: ARITHMETIC INTENSITY

In this section, we motivate using SNNs to classify 3D images. As discussed earlier, 3D images require 3D convolutions to extract both coarse and fine-grained features from all three dimensions. Essentially, it's the same as 2D convolutions, but the kernel sliding is now 3-dimensional, enabling a better capture of dependencies within the 3 dimensions and creating a difference in output dimensions post convolution. The kernel of the 3D convolution will move in 3-dimensions if the kernel's depth is less than the feature map's depth. Please see the illustration in **Figure 2**, where width, height, and depth of a convolutional kernel are given by k_l^x, k_l^y, and k_l^z, respectively. H_l^i, W_l^i, and D_l^i represents the height, weight, and depth for the input feature map. C_l^i and C_l^o denote the channel numbers of input and output feature map, respectively. Note that 3D convolutions are compute dominated, because the filters are strided in three directions for all the input channels to obtain a single output activation.

Let us evaluate the compute and memory access cost of a 3D CNN layer l with $X_l \in \mathbb{R}^{H_l^i \times W_l^i \times C_l^i \times D_l^i}$ as the input activation tensor, and $W_l \in \mathbb{R}^{k_l^x \times k_l^y \times k_l^z \times C_l^i \times C_l^o}$ as the weight tensor. Assuming no spatial reduction, the total number of floating point operations (FLOP) and memory accesses (Mem), which involves fetching the input activation (IA) tensor, weight (W) tensor, and writing to the output activation (OA) tensor, in layer l are given as

$$FLOP_{3D}^l = k_l^x \times k_l^y \times k_l^z \times C_l^i \times C_l^o \times H_l^i \times W_l^i \times D_l^i \tag{4}$$

$$Mem^l_{3D} = H^i_l \times W^i_l \times C^i_l \times D^i_l + k^x_l \times k^y_l \times k^z_l \times C^i_l \times C^o_l + H^i_l \times W^i_l \times C^o_l \times D^i_l \tag{5}$$

where the first, second and third term in Mem^l_{3D} correspond to IA, W, and OA, respectively. Note that we assume the whole operation can be performed in a single compute substrate (e.g., systolic array), without having to incur any additional data movement, and that the number of operations is independent of activation and weight bit-widths. Similarly, for a 2D CNN layer l, the total number of MACs and memory accesses is

$$FLOP^l_{2D} = k^x_l \times k^y_l \times C^i_l \times C^o_l \times H^i_l \times W^i_l \tag{6}$$

$$Mem^l_{2D} = H^i_l \times W^i_l \times C^i_l + k^x_l \times k^y_l \times C^i_l \times C^o_l + H^i_l \times W^i_l \times C^o_l \tag{7}$$

where we do not have the third dimension D. From Equations (3–6),

$$\frac{FLOP^l_{3D}}{FLOP^l_{2D}} = k^z_l \times D^i_l \tag{8}$$

$$\frac{Mem^l_{3D}}{Mem^l_{2D}} = \frac{(H^i_l \times W^i_l \times C^i_l \times D^i_l) + (k^x_l \times k^y_l \times k^z_l \times C^i_l \times C^o_l) + (H^i_l \times W^i_l \times C^o_l \times D^i_l)}{(H^i_l \times W^i_l \times C^i_l) + (k^x_l \times k^y_l \times C^i_l \times C^o_l) + (H^i_l \times W^i_l \times C^o_l)} \tag{9}$$

$$\leq \frac{H^i_l \times W^i_l \times C^i_l \times D^i_l}{H^i_l \times W^i_l \times C^i_l} + \frac{(k^x_l \times k^y_l \times k^z_l \times C^i_l \times C^o_l)}{(k^x_l \times k^y_l \times C^i_l \times C^o_l)} + \frac{(H^i_l \times W^i_l \times C^o_l \times D^i_l)}{(H^i_l \times W^i_l \times C^o_l)} \tag{10}$$

$$\leq 2D^i_l + k^z_l \tag{11}$$

Assuming $k^z_l = 3$ (all SOTA CNN architectures have filter size 3 in each dimension),

$$\frac{FLOP^l_{3D}}{FLOP^l_{2D}} \geq \frac{Mem^l_{3D}}{Mem^l_{2D}} \text{ if } D^i_l \geq 3 \tag{12}$$

Hence, 3D CNNs have higher arithmetic intensity, compared to 2D CNNs, when the spatial dimension D is higher than 3. This holds true in all but the last layer of a deep CNN network. For a 100×100 input activation tensor with 64 and 128 input and output channels, respectively, adding a third dimension of size 100 (typical hyperspectral images has 100s of spectral bands), and necessitating the use of 3D CNNs, increases the FLOP count by 300×, whereas the memory access cost increases by 96.5×. Note that these improvement factors are obtained by setting the input and output activation dimensions above in Eqs. 8 and 9 and assuming $k^x_l = k^y_l = k^z_l = 3$.

Moreover, as shown in Section 7, the energy consumption of a 3D CNN is compute bound on both general-purpose and neuromorphic hardware, and the large increment in FLOPs translates to significant SNN savings in total energy, as an AC operation is significantly cheaper than a MAC operation. Note that SNNs cannot reduce the memory access cost involving the weights.

4. PROPOSED QUANTIZED SNN TRAINING METHOD

In this section, we evaluate and compare the different choices for SNN quantization in terms of compute efficiency and model accuracy. We then incorporate the chosen quantization technique into STDB, which we refer to as Q-STDB.

4.1. Study of Quantization Choice

Uniform quantization transforms a weight element $w \in [w_{min}, w_{max}]$ to a range $[-2^{b-1}, 2^{b-1}-1]$ where b is the bit-width of the quantized integer representation. There are primarily two choices for the above transformation, known as *affine* and *scale* quantization. In affine quantization, the quantized value can be written as $w_a = s_a \cdot w + z_a$, where s_a and z_a denote the scale and zero point (the quantized value to which the real value zero is mapped), respectively. However, scale quantization performs range mapping with only a scale transformation, does not have a zero correction term, and has a symmetric representable range $[-\alpha, +\alpha]$. Hence, affine quantization leads to more accurate representations compared to the scale counterpart. Detailed descriptions of these two types of quantization can be found in Jain et al. (2020) and Wu et al. (2020).

To evaluate the compute cost of our quantization framework, let us consider a 3D convolutional layer l, the dominant layer in HSI classification models, that performs a tensor operation $O_l = X_l \circledast W_l$ where $X_l \in \mathbb{R}^{H^i_l \times W^i_l \times C^i_l \times D^i_l}$ is the IA tensor, $W_l \in \mathbb{R}^{k^x_l \times k^y_l \times k^z_l \times C^i_l \times C^o_l}$ is the W tensor and $O^l \in \mathbb{R}^{H^o_l \times W^o_l \times C^o_l \times D^o_l}$ is the OA tensor, with the same notations as used in Section 3. The result of the real-valued operation $O_l = X_l \circledast W_l$ can be approximated with quantized tensors X^Q_l and W^Q_l, by first dequantizing them producing $\hat{X}_l$ and $\hat{W}_l$, respectively, and then performing the convolution. Note that the same quantization parameters are shared by all elements in the weight tensor, because this reduces the computational cost compared to other granularity choices with no impact on model accuracy. Activations are similarly quantized, but only in the input layer, since they are binary spikes in the remaining layers. Also, note that both X^Q_l and W^Q_l have similar dimensions as X_l and W_l, respectively. Assuming the tensors are scale-quantized per layer,

$$O_l = X_l \circledast W_l \approx \hat{X}_l \circledast \hat{W}_l = X^Q_l \circledast W^Q_l \cdot \left(\frac{1}{s^X_s \cdot s^W_s}\right) \tag{13}$$

where s^X_s and s^W_s are scalar values for scale quantization representing the levels of the input and weight tensor, respectively. Hence, scale quantization results in an integer convolution, followed by a point-wise floating-point multiplication for each output element. Given that a typical 3D convolution operation involves a few thousands of MAC operations (accumulate for binary spike inputs) to compute an output element, a single floating-point operation for the scaling shown in Equation (13) is a negligible computational cost. This is because computing $X^l \circledast W^l$ involves element-wise multiplications of the weight kernels across multiple channels (for example, for a 3D convolution with 3 × 3 × 3 kernel and 100 channels, we need to perform 2700 MACs) and the corresponding overlapping input activation maps. The accumulated output then needs to be divided by $s_s^X \cdot s_s^W$, which adds negligible compute cost.

Although both affine and scale quantization enable the use of low-precision arithmetic, affine quantization results in more computationally expensive inference as shown below.

$$O_l \approx \frac{X_l^Q - z_a^X}{s_a^X} \circledast \frac{W_l^Q - z_a^W}{s_a^W}$$
$$= \frac{(X_l^Q \circledast W_l^Q - z_a^X \circledast (W_l^Q - z_a^W) - X_l^Q \circledast z_a^W)}{s_a^X \cdot s_a^W} \quad (14)$$

Note that z_a^X and z_a^W are tensors of sizes equal to that of X_l^Q and W_l^Q, respectively, that consist of repeated elements of the scalar zero-values of the input activation and weight tensor, respectively. On the other hand, s_a^X and s_a^W are the corresponding scale values. The first term in the numerator of Equation (14) is the integer convolution operation similar to the one performed in scale quantization shown in Equation (13). The second term contains integer weights and zero-points, which can be computed offline, and adds an element-wise addition during inference. The third term, however, involves point-wise multiplication with the quantized activation X_l^Q, which cannot be computed beforehand. As we show in Section 7.5.1, this extra computation can increase the energy consumption of our SNN models by over an order of magnitude.

However, our experiments detailed in Section 7 show that ignoring the affine shift during SNN training degrades the test accuracy significantly. Hence, the forward path computations during SNN training follows affine quantization as per (Equation 14), while the other steps involved in SNN training (detailed in Section 4.2), namely gradient computation, and parameter update, use the full-precision weights and membrane potentials, similar to binary ANN training to aid convergence (Courbariaux et al., 2016).

After training, the full-precision weights are rescaled for inference using scale quantization, as per Equation (13), which our results show yields negligible accuracy drop compared to using affine-scaled weights. The membrane potentials obtained as results of the accumulate operations only need to be compared with the threshold voltage once for each time step, which consumes negligible energy, and can be performed using fixed-point comparators (in the periphery of the memory array for PIM accelerators).

Notice that the affine quantization acts as an intermediate representation that lies between full-precision and scale quantization during training; using full-precision causes a large mismatch between weight representations during training and inference, while scale quantization during training results in a similar mismatch during its forward and backward computations. Thus, in principle, this approach is similar to incremental quantization approaches (Zhou et al., 2017) in which we incrementally adjust the type of quantization from the more accurate affine form to more energy-efficient scale form. Lastly, we note that our approach to quantization is also applicable to standard 3D CNNs but the relative savings is significantly higher in SNNs due to the fact that inference is implemented without multiply accumulates.

4.2. Q-STDB Based Training

Our proposed training algorithm, illustrated in **Figure 3**, incorporates the above quantization methodology into the STDB technique (Rathi and Roy, 2020), where the spatial and temporal credit assignment is performed by unrolling the SNN network in time and employing BPTT.

Output Layer: The neuron model in the output layer L only accumulates the incoming inputs without any leakage, does not generate an output spike, and is described by

$$\boldsymbol{u}_L^t = \boldsymbol{u}_L^{t-1} + \hat{\boldsymbol{w}}_L \boldsymbol{o}_{L-1}^t \quad (15)$$

where N is the number of output labels, $\boldsymbol{u}_L$ is a vector containing the membrane potential of N output neurons, $\hat{\boldsymbol{w}}_L$ is the affine quantized weight matrix connecting the last two layers (L and $L-1$), and $\boldsymbol{o}_{L-1}$ is a vector containing the spike signals from layer ($L-1$). The loss function is defined on $\boldsymbol{u}_L$ at the last time step T ($\boldsymbol{u}_L^T$). Since u_L^T is a vector consisting of continuous values, we compute the SNN's predicted distribution (p) as the softmax of u_L^T, similar to the output fully-connected layer of a CNN. Since our SNN is used only for *classification* tasks, we employ the popular cross-entropy loss. The loss function $\mathcal{L}$ is thus defined as the cross-entropy between the true one-hot encoded output (y) and the distribution p.

$$\mathcal{L} = -\sum_{i=1}^{N} y_i log(p_i), \quad p_i = \frac{e^{u_i^T}}{\sum_{j=1}^{N} e^{u_j^T}}, \quad (16)$$

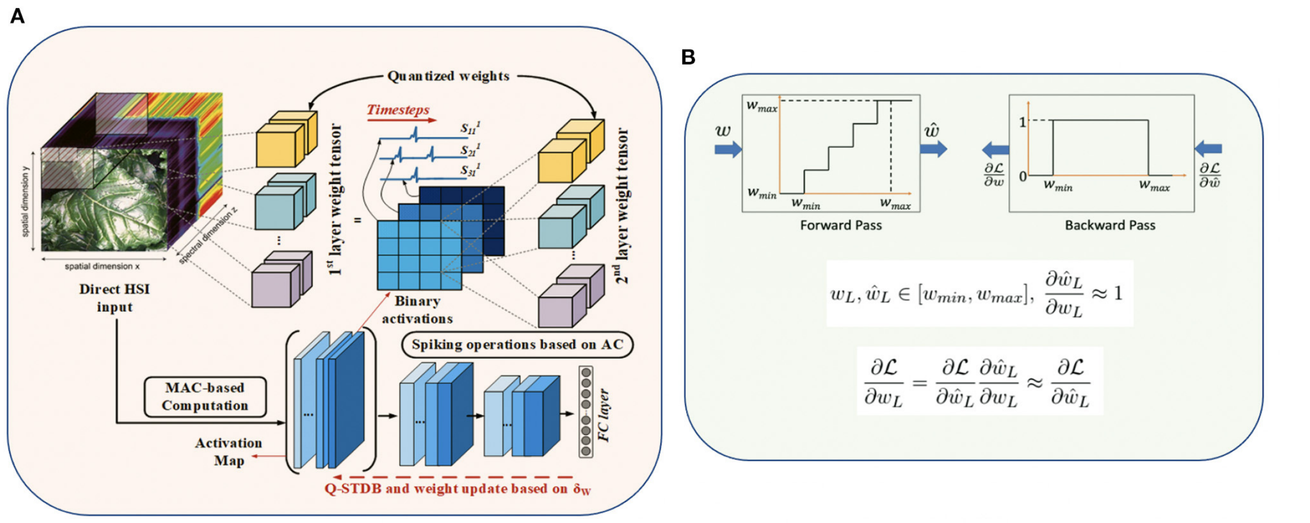
The derivative of the loss function with respect to the membrane potential of the neurons in the final layer is described by $\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_L^T} = (\boldsymbol{p}-\boldsymbol{y})$, where $\boldsymbol{p}$ and $\boldsymbol{y}$ are vectors containing the softmax and one-hot encoded values of the true label, respectively. To compute the gradient at the current time step, the membrane potential at the previous step is considered as an input quantity (Rathi and Roy, 2020). With the affine-quantized weights in the forward path, gradient descent updates the network parameters $\boldsymbol{w}_L$ of the output layer as

$$\boldsymbol{w}_L = \boldsymbol{w}_L - \eta \Delta \boldsymbol{w}_L \quad (17)$$

$$\Delta \boldsymbol{w}_L = \sum_t \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_L} = \sum_t \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_L^t} \frac{\partial \boldsymbol{u}_L^t}{\partial \hat{\boldsymbol{w}}_L} \frac{\partial \hat{\boldsymbol{w}}_L}{\partial \boldsymbol{w}_L}$$
$$= \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_L^T} \sum_t \frac{\partial \boldsymbol{u}_L^t}{\partial \hat{\boldsymbol{w}}_L} \frac{\partial \hat{\boldsymbol{w}}_L}{\partial \boldsymbol{w}_L} \approx (\boldsymbol{p}-\boldsymbol{y}) \sum_t \boldsymbol{o}_{L-1}^t \quad (18)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{o}_{L-1}^t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_L^t} \frac{\partial \boldsymbol{u}_L^t}{\partial \boldsymbol{o}_{L-1}^t} = (\boldsymbol{p}-\boldsymbol{y}) \hat{\boldsymbol{w}}_L \quad (19)$$

where η is the learning rate (LR). Note that the derivative of the affine quantization function of the weights ($\frac{\partial \hat{\boldsymbol{w}}_L}{\partial \boldsymbol{w}_L}$) is undefined at the step boundaries and zero everywhere, as shown in **Figure 3A**. Our training framework addresses this challenge by using the Straight-through Estimator (STE) (Courbariaux et al., 2016),



$$w_L, \hat{w}_L \in [w_{min}, w_{max}], \quad \frac{\partial \hat{w}_L}{\partial w_L} \approx 1$$

$$\frac{\partial \mathcal{L}}{\partial w_L} = \frac{\partial \mathcal{L}}{\partial \hat{w}_L}\frac{\partial \hat{w}_L}{\partial w_L} \approx \frac{\partial \mathcal{L}}{\partial \hat{w}_L}$$

FIGURE 3 | (A) Proposed SNN training framework details with 3D convolutions, and **(B)** Fake quantization forward and backward pass with straight through estimator (STE) approximation.

which approximates the derivative to be equal to 1 for inputs in the range $[w_{min}, w_{max}]$ as shown in **Figure 3B**, where w_{min} and w_{max} are the minimum and maximum values of the weights in a particular layer. Note that w_{min} and w_{max} are updated at the end of every mini-batch to ensure all the weights lie between w_{min} and w_{max} during the forward and backward computations in each training iteration. Hence, we use $\frac{\partial \hat{\boldsymbol{w}}_L}{\partial \boldsymbol{w}_L} \approx 1$ to compute the loss gradients in Equation (18).

Hidden layers: The neurons in all the hidden layers follow the quantized LIF model shown in Equation (2). All neurons in a layer possess the identical leak and threshold value. This reduces the number of trainable parameters and we did not observe any noticeable accuracy change by assigning different threshold/leak value to each neuron, similar to Datta et al. (2021). With a single threshold for each layer, it may seem redundant to train both the weights and threshold together. However, we observe, similar to Rathi and Roy (2020) and Datta et al. (2021) that the latency required to obtain the SOTA classification accuracy decreases with the joint optimization, which further drops by training the leak term. This may be because the loss optimizer can reach an improved local minimum when all the parameters are tunable. The weight update in Q-STDB is calculated as

$$\Delta w_l = \sum_t \frac{\partial \mathcal{L}}{\partial w_l} = \sum_t \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_l^t}\frac{\partial \boldsymbol{z}_l^t}{\partial \boldsymbol{o}_l^t}\frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{u}_l^t}\frac{\partial \boldsymbol{u}_l^t}{\partial \hat{\boldsymbol{w}}_l}\frac{\partial \hat{\boldsymbol{w}}_l}{\partial \boldsymbol{w}_l}$$
$$\approx \sum_t \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_l^t}\frac{\partial \boldsymbol{z}_l^t}{\partial \boldsymbol{o}_l^t}\frac{1}{v_l}\boldsymbol{o}_{l-1}^t \cdot 1 \qquad (20)$$

where $\frac{\partial \hat{\boldsymbol{w}}_l}{\partial \boldsymbol{w}_l}$ and $\frac{\partial \boldsymbol{z}_l^t}{\partial \boldsymbol{o}_l^t}$ are the two discontinuous gradients. We calculate the former using STE described above, while the latter is approximated using surrogate gradient (Bellec et al., 2018) shown below.

$$\frac{\partial \boldsymbol{z}_l^t}{\partial \boldsymbol{o}_l^t} = \gamma \cdot max(0, 1 - |\boldsymbol{z}_l^t|) \qquad (21)$$

Note that γ is a hyperparameter denoting the maximum value of the gradient. The threshold and leak update is computed similarly using BPTT (Rathi and Roy, 2020).

5. SRAM-BASED PIM ACCELERATION

Efficient hardware implementations of neural network algorithms are being widely explored by the research community in an effort to enable intelligent computations on resource constrained edge devices (Chen et al., 2020). Existing computing systems based on the well-known von-Neumann architecture (characterized by physically separated memory and computing units) suffer from energy and throughput bottleneck, referred as the *memory wall bottleneck* (Agrawal et al., 2018; Dong et al., 2018). Novel memory-centric paradigms like PIM are being extensively investigated by the research community to mitigate the energy-throughput constraints arising from the memory wall bottleneck. As discussed in Section 1, the first layer of a direct coded SNN is not as computationally efficient as the other layers, as it processes continuous valued inputs as opposed to spiking inputs, and dominates the total energy consumption. Further, for 3D images such as HSI, the number of real valued computations in the first layer of an SNN is orders of magnitude more than 2D images.

In order to enable energy-efficient hardware for SNNs catering to 3D images, we propose to exploit the high-parallelism, high-throughput and low-energy benefits of analog PIM in SRAM, for the first layer of the SNN. As mentioned earlier, the first layer of SNN requires real valued MAC operations which are well-suited to be accelerated using analog PIM approaches (Kang et al., 2018; Ali et al., 2020). Moreover, the number of weights in

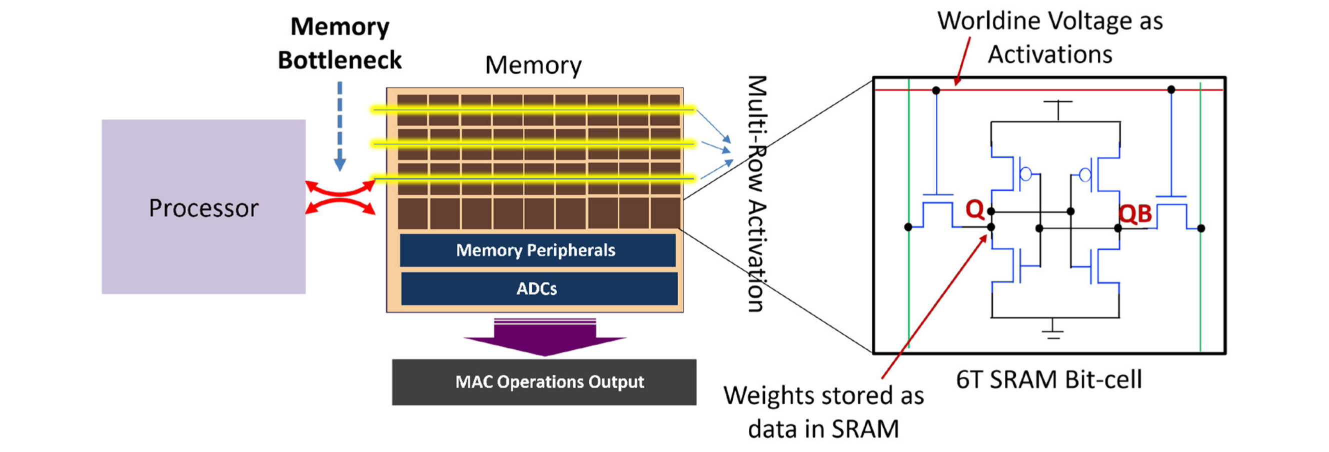


FIGURE 4 | PIM architecture in the first layer to process MAC operations for the first layer of direct coded SNNs. Other layers of the SNN are processed with highly parallel programmable architecture using simpler accumulate operations.

the first layer of a typical 3D CNN architecture is substantially less compared to the other layers, which ensures that we can perform PIM using a single memory array, thereby reducing the complexity of the peripheral circuits, such as adder trees for partial sum reduction. Several proposals achieving multiple degrees of compute parallelism within on-chip memory based on SRAM arrays have been proposed (Agrawal et al., 2018, 2019; Dong et al., 2018; Biswas and Chandrakasan, 2019; Jaiswal et al., 2019). Interestingly, both digital (Agrawal et al., 2018; Dong et al., 2018) as well as analog- mixed-signal approaches (Agrawal et al., 2019; Biswas and Chandrakasan, 2019) have been explored extensively. Analog approaches are of particular importance due to higher levels of data parallelism and compute throughput compared to digital counterparts in performing MAC computations. Our adopted PIM architecture for the first layer of our proposed SNNs is illustrated in **Figure 4**. The PIM architecture leverages analog computing for parallel MAC operations by mapping activations as voltages on the wordlines and weights as data stored in the SRAM bit-cells (represented as Q and QB). As shown in Ali et al. (2020), multi-bit MAC operations can be enabled in SRAM arrays by activating multiple rows, simultaneously, allowing appropriately weighted voltages to develop on each column of the SRAM array representing the resulting MAC operations computed in analog domain. Peripheral ADC circuits are used to convert the analog MAC operation into corresponding digital data for further computations.

To summarize, we propose use of analog PIM to accelerate the MAC intensive compute requirements for the first layer of the SNN. The remaining layers of the SNN leverage traditional digital hardware implementing simpler accumulate operations. Advantageously, our proposed quantized SNN with small number of weights in the first layer is well-suited for low-overhead PIM circuits, as reduction in bit-precision and peripheral complexity drastically improves the energy and throughput efficiency of analog PIM architectures (Kang et al., 2018).

6. PROPOSED CNN ARCHITECTURES, DATASETS, AND TRAINING DETAILS

6.1. Model Architectures

We developed two models, a 3D and a hybrid fusion of 3D and 2D convolutional architectures, that are inspired by the recently proposed CNN models (Ben Hamida et al., 2018; Luo et al., 2018; Roy et al., 2020) used for HSI classification and compatible with our ANN-SNN conversion framework. We refer to the two models CNN-3D and CNN-32H.

There are several constraints in the training of the baseline ANN models needed to obtain near lossless ANN-SNN conversion (Diehl et al., 2016; Sengupta et al., 2019). In particular, we omit the bias term from the ANN models because the integration of the bias term over multiple SNN timesteps tends to shift the activation values away from zero which causes problems in the ANN-SNN conversion process (Sengupta et al., 2019). In addition, similar to Sengupta et al. (2019), Rathi and Roy (2020), Rathi et al. (2020), and Kim and Panda (2021), we do not use batch normalization (BN) layers because using identical BN parameters (e.g., global mean μ, global standard deviation σ, and trainable parameter γ) for the statistics of all timesteps do not capture the temporal dynamics of the spike train in an SNN. Instead, we use dropout (Srivastava et al., 2014) as the regularizer for both ANN and SNN training. Recent research (Rathi and Roy, 2020; Rathi et al., 2020) indicates that there is no problem in yielding state-of-the-art accuracy in complex image recognition tasks, such as CIFAR-100, with models without batch normalization and bias. We observe the same for HSI models in this work as well. Moreover, our initial ANN models employ ReLU nonlinearity after each convolutional and linear layer (except the classifier layer), due to the similarity between ReLU and LIF neurons. Our pooling operations use average pooling because for binary spike based activation layers, max pooling incurs significant information loss. Our SNN-specific architectural modifications are illustrated in **Figure 5**.

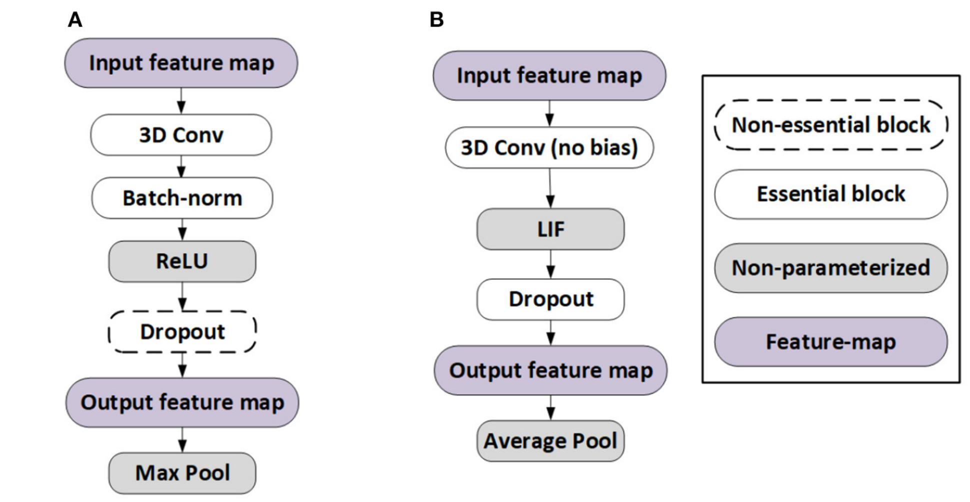


FIGURE 5 | Architectural differences between **(A)** ANN and **(B)** SNN for near-lossless ANN-SNN conversion.

TABLE 1 | Model architectures employed for CNN-3D and CNN-32H in classifying the IP dataset.

Layer type	Size of input feature map	Number of filters	Size of each filter	Stride value	Padding value	Dropout value	Size of output feature map
Architecture : CNN-3D							
3D Convolution	(5,5,200,1)	20	(3,3,3)	(1,1,1)	(0,0,0)	-	(3,3,198,20)
3D Convolution	(3,3,198,20)	40	(1,1,3)	(1,1,2)	(1,0,0)	-	(3,3,99,40)
3D Convolution	(3,3,99,40)	84	(3,3,3)	(1,1,1)	(1,0,0)	-	(1,1,99,84)
3D Convolution	(1,1,99,84)	84	(1,1,3)	(1,1,2)	(1,0,0)	-	(1,1,50,84)
3D Convolution	(1,1,50,84)	84	(1,1,3)	(1,1,1)	(1,0,0)	-	(1,1,50,84)
3D Convolution	(1,1,50,84)	84	(1,1,2)	(1,1,2)	(1,0,0)	-	(1,1,26,84)
Architecture : CNN-32H							
3D Convolution	(3,3,200,1)	90	(3,3,18)	(1,1,7)	(0,0,0)	-	(1,1,27,90)
2D Convolution	(27,90,1)	64	(3,3)	(1,1)	(0,0)	-	(25,88,64)
2D Convolution	(25,88,64)	128	(3,3)	(1,1)	(0,0)	-	(23,86,128)
Avg. Pooling	(23,86,128)	-	(4,4)	(4,4)	(0,0)	-	(59,21,128)
Dropout	(5,21,128)	-	-	-	-	0.2	(5,21,128)
Linear	13,440	6,881,280	-	-	-	-	512

Every convolutional and linear layer is followed by a ReLU non-linearity. The last classifier layer is not shown. The size of the activation map of a 3D CNN is written as (H,W,D,C) where H, W, D, and C represent the height, width, depth of the input feature map and the number of channels. Since the 2D CNN layer does not have the depth dimension, its feature map size is represented as (H,W,C).

We also modified the number of channels and convolutional layers to obtain compact yet accurate models. 2D patches of sizes 5×5 and 3×3 were extracted for CNN-3D and CNN-32H, respectively, without any reduction in dimensionality from each dataset. Higher sized patches increase the computational complexity without any significant improvement in test accuracy. Note that magnitude based structured weight pruning (Han et al., 2015b), which has been shown to be an effective technique for model compression, can only remove $<$ 15% of the weights averaging across the two architectures, with <1% degradation in test accuracy for all the three datasets used in our experiments, which also indicates the compactness of our models. The details of both models are given in **Table 1**.

6.2. Datasets

We used four publicly available datasets, namely Indian Pines, Pavia University, Salinas scene, and HyRANK. A brief description follows for each one, and few sample images found in some of these datasets are shown in **Figure 6**.

Indian Pines: The Indian Pines (IP) dataset consists of 145×145 spatial pixels and 220 spectral bands in a range of 400–2,500 nm. It was captured using the AVIRIS sensor over North-Western Indiana, USA, with a ground sample distance (GSD) of 20 m and has 16 vegetation classes.

Pavia University: The Pavia University (PU) dataset consists of hyperspectral images with 610×340 pixels in the spatial dimension, and 103 spectral bands, ranging from 430 to 860 nm

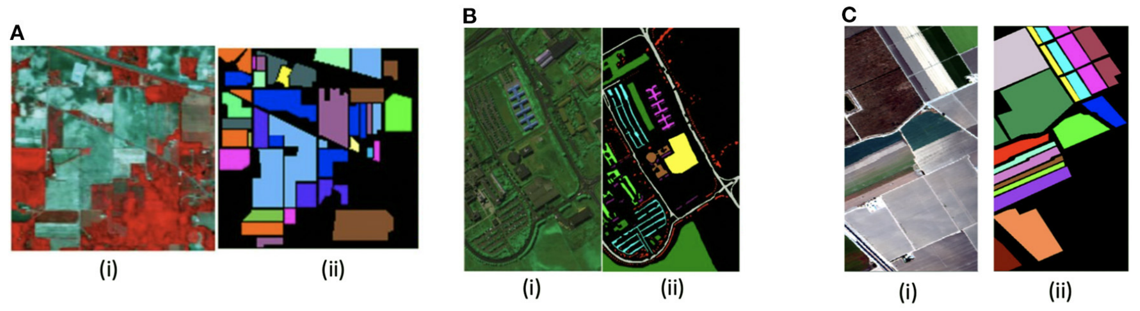


FIGURE 6 | (i) False color-map and (ii) ground truth images of different HSI datasets used in our work, namely **(A)** Indian Pines, **(B)** Pavia University, and **(C)** Salinas Scene.

in wavelength. It was captured with the ROSIS sensor with GSD of 1.3 m over the University of Pavia, Italy. It has a total of 9 urban land-cover classes.

Salinas Scene: The Salinas Scene (SA) dataset contains images with 512×217 spatial dimension and 224 spectral bands in the wavelength range of 360–2, 500 nm. The 20 water absorbing spectral bands have been discarded. It was captured with the AVIRIS sensor over Salinas Valley, California with a GSD of 3.7 m. In total 16 classes are present in this dataset.

HyRANK: The ISPRS HyRANK dataset is a recently released hyperspectral benchmark dataset. Different from the above HSI datasets that contain a singlr hyperspectral scene, the HyRANK dataset consists of two hyperspectral scenes, namely Dioni and Loukia. Similar to Meng et al. (2021), we use the available labeled samples in the Dioni scene for training, while those in the Loukia scene for testing. The Dioni and Loukia scenes comprise 250 × 1, 376 and 249 × 945 spectral samples, respectively, and each has 176 spectral reflectance bands.

For preprocessing, images in all the data sets are normalized to have a zero mean and unit variance. For our experiments, all the samples (except that of the HyRANK dataset) are randomly divided into two disjoint training and test sets. The limited 40% samples are used for training and the remaining 60% for performance evaluation.

6.3. ANN Training and SNN Conversion Procedures

We start by performing full-precision 32-bit ANN training for 100 epochs using the standard SGD optimizer with an initial learning rate (LR) of 0.01 that decayed by a factor of 0.1 after 60, 80, and 90 epochs.

The ANN-SNN conversion entails the estimation of the values of the weights and per-layer thresholds of the SNN model architecture. The weights are simply copied from a trained DNN model to the iso-architecture target SNN model. The threshold for each layer is computed sequentially as the 99.7 percentile of the pre-activation distribution (weighted sum of inputs received by each neuron in a layer) over the total number of timesteps (Rathi and Roy, 2020) for a small batch of HSI images (of size 50 in our case). Note that we use 100 time steps to evaluate the thresholds, while the SNN training and inference are performed with only 5 time steps. In our experiments we scale the initial layer thresholds by 0.8. We keep the leak of each layer set to unity while evaluating these thresholds. Note that employing direct coding as used in our work and others (Rathi and Roy, 2020) can help avoid any approximation error arising from the input spike generation (conversion from raw images to spike trains) process and aid ANN-SNN conversion. Lower bit-precision of weights will most likely not exacerbate the conversion process, assuming the ANN models can be trained accurately with the same bit-precision.

We then perform quantization-aware SNN training as described in Section 4 for another 100 epochs. We set $\gamma = 0.3$ (Bellec et al., 2018) and used the ADAM optimizer with a starting LR of 10^{-4} which decays by a factor of 0.5 after 60, 80, and 90 epochs. All experiments are performed on a Nvidia 2080Ti GPU with 11 GB memory.

7. EXPERIMENTAL RESULTS AND ANALYSIS

This section first describes our inference accuracy results, then analyzes the associated spiking and energy consumption. It then describes several ablation studies and a comparison of the training time and memory requirements.

7.1. ANN and SNN Inference Results

We report the best Overall Accuracy (OA), Average Accuracy (AA), and Kappa Coefficient measures to evaluate the HSI classification performance for our proposed architectures, similar to Ben Hamida et al. (2018). Here, OA represents the number of correctly classified samples out of the total test samples. AA represents the average of class-wise classification accuracies, and Kappa is a statistical metric used to assess the mutual agreement between the ground truth and classification maps. Column-2 in **Table 2** shows the ANN accuracies, column-3 shows the accuracy after ANN-SNN conversion with 50 time steps[1]. Column-4 shows the accuracy when we perform our proposed training without quantization, while columns 5 to 7 shows the SNN

[1]We empirically observe that at least 50 time steps are required for lossless ANN-SNN conversion.

TABLE 2 | Model performances with Q-STDB based training on IP, PU, SS, and HyRANK datasets for CNN-3D and CNN-32H after (A) ANN training, (B) ANN-to-SNN conversion, (C) 32-bit SNN training, (D) 4-bit SNN training, (E) 5-bit SNN training, and (F) 6-bit SNN training, with only 5 time steps.

Dataset	A. ANN accuracy (%)			B. Accuracy after ANN-to-SNN conv. (%)			C. Accuracy after FP SNN training (%)			D. Accuracy after 4-bit SNN training (%)			E. Accuracy after 5-bit SNN training (%)			F. Accuracy after 6-bit SNN training (%)		
	OA	AA	Kappa	OA	AA	Kappa	OA	AA	Kappa	OA	AA	Kappa	OA	AA	Kappa	OA	AA	Kappa
Architecture : CNN-3D																		
IP	98.86	98.42	98.55	57.68	50.88	52.88	98.92	98.76	98.80	97.08	95.64	95.56	98.38	97.78	98.03	98.68	98.34	98.20
PU	99.69	99.42	99.58	91.16	88.84	89.03	99.47	99.06	99.30	98.21	97.54	97.75	99.26	98.48	98.77	99.50	99.18	99.33
SS	98.89	98.47	98.70	81.44	76.72	80.07	98.49	97.84	98.06	96.47	93.16	94.58	97.25	95.03	95.58	97.95	97.09	97.43
HyRANK	64.21	63.27	47.34	34.80	58.97	20.64	63.18	61.25	45.25	59.76	56.40	42.28	61.70	60.48	46.06	62.96	61.27	46.82
Architecture : CNN-32H																		
IP	97.60	97.08	97.44	70.88	66.56	67.89	97.27	96.29	96.35	96.63	95.81	95.89	97.23	96.08	96.56	97.45	96.73	96.89
PU	99.50	99.09	99.30	94.96	90.12	93.82	99.38	98.83	99.13	99.17	98.41	98.68	99.25	98.84	98.86	99.35	98.88	98.95
SS	98.88	98.39	98.67	88.16	84.19	85.28	97.92	97.20	97.34	97.34	96.32	96.77	97.65	96.81	96.97	97.99	97.26	97.38
HyRANK	64.43	70.68	52.82	24.26	26.90	19.37	63.72	67.89	49.59	62.27	62.50	46.58	63.27	65.32	47.98	63.34	66.66	48.21

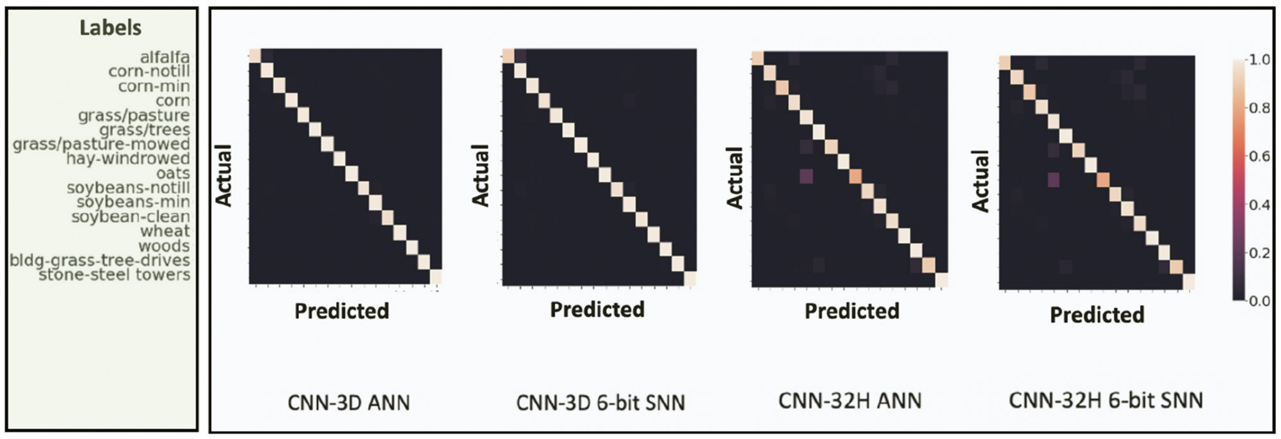


FIGURE 7 | Confusion Matrix for HSI test performance of ANN and proposed 6-bit SNN over IP dataset for both CNN-3D and CNN-32H. The ANN and SNN confusion matrices look similar for both the network architectures. CNN-32H incurs a little drop in accuracy compared to CNN-3D due to shallow architecture.

test accuracies obtained with Q-STDB for different weight bit precisions (4 to 6 bits). SNNs trained with 6-bit weights result in 5.33× reduction in bit-precision compared to full-precision (32-bit) models and, for all three tested data sets, perform similar to the full precision ANNs for both the CNN-3D and CNN-32H architectures. Although the membrane potentials do not need to be quantized as described in Section 4, we observed that the model accuracy does not drop significantly even if we quantize them, and hence, the SNN results shown in **Table 2** correspond to 6-bit membrane potentials. Four-bit weights and potentials provide even lower complexity, but at the cost of a small accuracy drop. **Figure 7** shows the confusion matrix for the HSI classification performance of the ANN and proposed SNN over the IP dataset for both the architectures.

The inference accuracy (OA, AA, and Kappa) of our ANNs and SNNs trained via Q-STDB are compared with the current state-of-the-art ANNs used for HSI classification in **Table 3**. As we can see, simply porting the ANN architectures used in Ben Hamida et al. (2018) and Luo et al. (2018) to SNNs, and performing 6-bit Q-STDB results in significant drops in accuracy, particularly for the India Pines data set. In contrast, our CNN-3D-based SNN models suffer negligible OA drop (<1% for all datasets) compared to the best performing ANN models for HSI classification.

7.2. Spiking Activity

Each SNN spike involves a constant number of AC operations, and hence, consumes a fixed amount of energy. Consequently, the average spike count of an SNN layer l, denoted ζ_l, can be treated as a measure of compute-energy of the model (Sengupta et al., 2019; Rathi et al., 2020). We calculate ζ_l as the ratio of the total spike count in T steps over all the neurons of layer l to the number of neurons in the layer. Hence, the energy efficiency of an SNN model can be improved by decreasing the spike count.

TABLE 3 | Inference accuracy (OA, AA, and Kappa) comparison of our proposed SNN models obtained from CNN-3D and CNN-32H with state-of-the-art deep ANNs on IP, PU, SS, and HyRANK datasets.

References	ANN/SNN	Architecture	OA (%)	AA (%)	Kappa (%)
Dataset : Indian Pines					
Alipour-Fard et al. (2020)	ANN	MSKNet	81.73	71.4	79.2
Song et al. (2018)	ANN	DFFN	98.52	97.69	98.32
Zhong et al. (2018)	ANN	SSRN	99.19	98.93	99.07
Roy et al. (2020)	ANN	HybridSN	**99.75**	**99.63**	**99.71**
Ben Hamida et al. (2018)	ANN	6-layer 3D CNN	98.29	97.52	97.72
	SNN		95.88	94.26	95.34
Luo et al. (2018)	ANN	Hybrid CNN	96.15	94.96	95.73
	SNN		94.90	94.08	94.78
This work	ANN	CNN-3D	98.86	98.42	98.55
	SNN		98.79	98.34	98.60
This work	ANN	CNN-32H	97.60	97.08	97.44
	SNN		97.45	96.73	96.89
Dataset : Pavia University					
Alipour-Fard et al. (2020)	ANN	MSKNet	90.66	88.09	87.64
Song et al. (2018)	ANN	DFFN	98.73	97.24	98.31
Zhong et al. (2018)	ANN	SSRN	99.61	**99.56**	99.33
Meng et al. (2021)	ANN	DRIN	96.4	95.8	95.2
Ben Hamida et al. (2018)	ANN	6-layer 3D CNN	99.32	99.02	99.09
	SNN		98.55	98.02	98.28
Luo et al. (2018)	ANN	Hybrid CNN	99.05	98.35	98.80
	SNN		98.40	97.66	98.21
This work	ANN	CNN-3D	**99.69**	99.42	**99.58**
	SNN		99.50	99.18	99.33
This work	ANN	CNN-32H	99.50	99.09	99.30
	SNN		99.35	98.88	98.95
Dataset : Salinas Scene					
Song et al. (2018)	ANN	DFFN	98.87	**98.75**	98.63
Meng et al. (2021)	ANN	DRIN	96.7	98.6	96.3
Luo et al. (2018)	ANN	Hybrid CNN	98.85	98.35	98.22
	SNN		97.05	97.41	97.18
This work	ANN	CNN-3D	**98.89**	98.47	**98.70**
	SNN		97.95	97.09	97.43
This work	ANN	CNN-32H	98.88	98.39	98.67
	SNN		97.99	97.26	97.38
Dataset : HyRANK					
Meng et al. (2021)	ANN	DRIN	54.4	56.0	43.3
This work	ANN	CNN-3D	64.21	63.27	47.34
	SNN		62.96	61.27	46.82
This work	ANN	CNN-32H	**64.43**	**69.68**	**52.82**
	SNN		63.34	66.66	48.21

The bold values indicate maximum values.

Figure 8 shows the average spike count for each layer with Q-STDB when evaluated for 200 samples from each of the three datasets (IP, PU, SS) for the CNN-3D and CNN-32H architecture. For example, the average spike count of the 3^{rd} convolutional layer of the CNN-3D-based SNN for IP dataset is 0.568, which means each neuron in that layer spikes 0.568 times on average over all input samples over a 5 time step period. Note that the average spike count is less than 1.4 for all the datasets across both the architectures which leads to significant energy savings as described below.

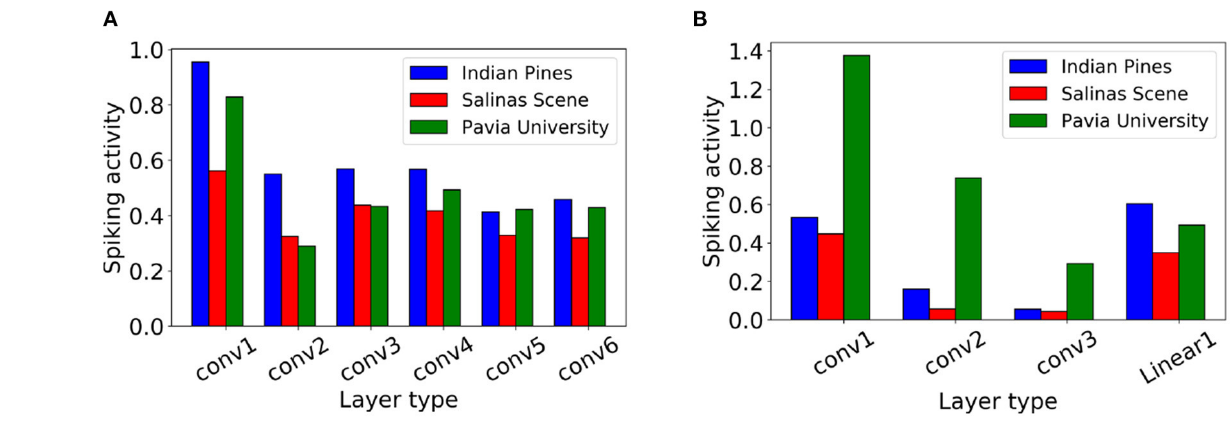


FIGURE 8 | Layerwise spiking activity plots for **(A)** CNN-3D and **(B)** CNN-32H on Indian Pines, Salinas Scene and Pavia University datasets.

7.3. Energy Consumption and Delay

In this section, we analyze the improvements in energy, delay, and EDP of our proposed SNN models compared to the baseline SOTA ANN models running on digital hardware for all the three datasets. We show that further energy savings can be obtained by using the PIM architecture discussed in Section 5 to process the first layer of our SNN models.

7.3.1. Digital Hardware

Let us assume a 3D convolutional layer l having weight tensor $\mathbf{W}^l \in \mathbb{R}^{k\times k\times k\times C_l^i\times C_l^o}$ that operates on an input activation tensor $\mathbf{I}^l \in \mathbb{R}^{H_l^i\times W_l^i\times C_l^i\times D_l^i}$, where the notations are similar to the one used in Section 4. We now quantify the energy consumed to produce the corresponding output activation tensor $\mathbf{O}^l \in \mathbb{R}^{H_l^o\times W_l^o\times C_l^o\times D_l^o}$ for an ANN and SNN, respectively. Our model can be extended to fully-connected layers with f_l^i and f_l^o as the number of input and output features, respectively, and to 2D convolutional layers, by shrinking a dimension of the feature maps.

In particular, for any layer l, we extend the energy model of Ali et al. (2020) and Kang et al. (2018) to 3D CNNs by adding the third dimension of weights (k) and output feature maps (D_l^o), as follows

$$E_l^{CNN} = C_l^i C_l^o k^3 E_{read} + C_l^i C_l^o k^3 H_l^o W_l^o D_l^o E_{mac} + P_{leak} T_l^{CNN} \tag{22}$$

where the first term denotes the memory access energy, the second term denotes the compute energy, while the third term highlights the static leakage energy. Note that T_l is the latency incurred to process the layer l, and can be written as

$$T_l^{CNN} = \left(\frac{C_l^i C_l^o k^3}{\frac{B_{IO}}{B_W} N_{bank}}\right) T_{read} + \left(\frac{C_l^i C_l^o k^3}{N_{mac}}\right) H_l^o W_l^o D_l^o T_{mac} \tag{23}$$

The notations for Equations (22) and (23), along with their values, obtained from Kang et al. (2018) and Ali et al. (2020) are illustrated in **Table 4**. The total energy is compute bound since the compute energy alone consumes ∼98% of the total energy averaged across all the layers for the CNN-3D architecture on all the datasets. The memory cost only dominates the few fully connected layers, accounting for $>$ 85% of their total energy.

TABLE 4 | Notations and their values used in energy, delay, and EDP equations for ANN and 6-bit SNNs.

Notation	Description	Value
B_{IO}	Number of bits fetched from SRAM to processor per bank	64
B_W	Bit width of the weight stored in SRAM	6
N_{col}	Number of columns in SRAM array	256
N_{bank}	Number of SRAM banks	4
$N_{mac}(N_{ac})$	Number of MACs (ACs) in processing element (PE) array	175 (175)
T_{read}	Time required to transfer 1-bit data between SRAM and PE	4 ns
T_{BLP}	Time required for one analog in-memory accumulation	4 ns
$E_{mac}(E_{ac})$	Energy consumed in a single MAC (AC)	3.1 pJ (0.1 pJ) for 32-bit
	Operation for a particular bit-precision	full-precision inputs (Horowitz, 2014)
$T_{mac}(T_{ac})$	Time required to perform a single MAC (AC) in PE	4 ns (0.4 ns)
T_{adc}	Time required for a single ADC operation	6 ns
E_{read}	Energy to transfer each weight element between SRAM and PE	5.2 pJ
E_{BLP}	Energy required for a single in-memory analog accumulation	0.08 pJ
E_{adc}	Energy required for an ADC operation	0.268 pJ

Similarly, we can extend the energy and delay model of Kang et al. (2018) and Ali et al. (2020) to our proposed SNNs, as follows

$$E_l^{SNN} = C_l^i C_l^o k^3 E_{read} + C_l^i C_l^o k^3 H_l^o W_l^o D_l^o \zeta_l E_{ac} + P_{leak} T_l^{SNN} \tag{24}$$

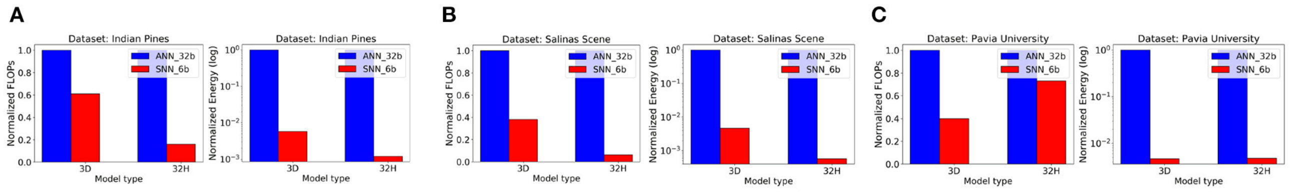


FIGURE 9 | Comparison of FLOPs and compute energy of CNN-3D and CNN-32H between ANN and SNN models while classifying on **(A)** Indian Pines, **(B)** Salinas Scene, and **(C)** Pavia University datasets, respectively.

$$T_l^{SNN} = \left(\frac{C_l^i C_l^o k^3}{\frac{B_{IO}}{B_W} N_{bank}}\right) T_{read} + \left(\frac{C_l^i C_l^o k^3}{N_{ac}}\right) H_l^o W_l^o D_l^o T_{ac} \quad (25)$$

for any layer l except the input layer that is based on direct encoding, whose energy and delay can be obtained from Equations (22, 23), respectively. The notations used in Equations (23, 24), along with their values are also shown in **Table 4**. Notice that the spiking energy in Equation (22) assume the use of zero-gating logic that activates the compute unit only when an input spike is received and thus is a function of spiking activity ζ_l. However, to extend the benefits of a low ζ^l to latency, we require either custom hardware or compiler support (Liu et al., 2018). For this reason, unlike energy, this paper assumes no delay benefit from ζ_l as is evident in Equation (25).

To compute E_{MAC} for full-precision weights (full-precision and 6-bits) and E_{AC} (6-bits) at 65 nm technology, we use the data from Horowitz (2014) obtained by silicon measurements (see **Table 4**). For 6-bit inputs, we scale the energy according to $E_{mac} \propto Q^{1.25}$ as shown in Moons et al. (2017), where Q is the bit-precision. On the other hand, E_{ac} (6-bits) is computed by scaling the full-precision data from Horowitz (2014), according to Simon et al. (2019), which shows E_{AC} is directly proportional to the data bit-width. Our calculations imply that E_{AC} is ∼13× smaller than E_{MAC} for 6-bit precision. Note that this number may vary for different technologies, but, in most technologies, an AC operation is significantly less expensive than a MAC operation. As required in the direct input encoding layer, we obtain E_{mac} for 8-bit inputs and 6-bit weights from Kang et al. (2018), applying voltage scaling for iso-V_{dd} conditions with the other E_{mac} and E_{ac} estimations from Horowitz (2014). We use $T_{ac} = 0.1T_{mac}$ for 6-bit inputs from Ganesan (2015) and the fact that the latency of a MAC unit varies logarithmically with bit precision (assuming a carry-save adder) to calculate the delay, and the resulting EDP of the baseline SOTA ANN and our proposed SNN models. Note that the architectural modifications applied to the existing SOTA models to create our baseline ANNs (Ben Hamida et al., 2018; Roy et al., 2020) only enhance ANN-SNN conversion, and do not lead to significant changes in energy consumption. Since the total energy is compute bound, we also calculate the total number of floating point operations (FLOPs), which is a standard metric to evaluate the energy cost of ML models.

Figure 9 illustrates the total energy consumption and FLOPs for full precision ANN and 6-bit quantized SNN models of the two proposed architectures, where the energy is normalized to that of the baseline ANN. We also consider 6-bit ANN models to compare the energy efficiency of low-precision ANNs and SNNs. We observe that 6-bit ANN models are 12.5× energy efficient compared to 32-bit ANN models due to significant improvements in MAC energy with quantization, as shown in Moons et al. (2017). Note that we can achieve similar HSI test accuracies shown in **Table 2** with quantized ANNs as well. We compare the layer-wise and total energy, delay, and EDP of our proposed SNNs with those of equivalent-precision ANNs in **Figure 10**.

The FLOPs for SNNs obtained by our proposed training framework is smaller than that for the baseline ANN due to low spiking activity. Moreover, because the ACs consume significantly less energy than MACs for all bit precisions, SNNs are significantly more compute efficient. In particular, for CNN-3D on IP, our proposed SNN consumes ∼199.3× and ∼33.8× less energy than an iso-architecture full-precision and 6-bit ANN with similar parameters, respectively. The improvements become ∼560.6× (∼9976× in EDP) and ∼44.8× (∼412.2× in EDP), respectively, averaging across the two network architectures and three datasets.

7.3.2. PIM Hardware

Though SNNs improve the total energy significantly as shown above, the first layer needs the expensive MACs due to direct encoding, and accounts for ∼27% and ∼22% of the total energy on average across the three datasets for CNN-3D and CNN-32H, respectively. To address this issue, we propose to adopt an SRAM-based memory array to process the computations incurred in the first layer, in the memory array itself, as discussed in Section 5.

We similarly extended the energy and delay models of Ali et al. (2020) and Kang et al. (2018) to the PIM implementation of the first layer of our proposed SNN architectures. The resulting energy and delay can be written as

$$E_1^{SNN} = C_1^i C_1^o k^3 \left(E_{BLP} + \frac{E_{ADC}}{R}\right) + P_{leak} T_1^{SNN} \quad (26)$$

$$T_1^{SNN} = \left(\frac{C_1^i C_1^o k^3}{\frac{N_{col}}{B_W} N_{bank}}\right) H_1^o W_1^o D_1^o \left(T_{read} + \frac{T_{adc}}{R}\right) \quad (27)$$

where the new notations along with their values are in **Table 4**. Following 65 nm CMOS technology limitations, we keep the array parameters similar to Kang et al. (2018), and T_{adc} and

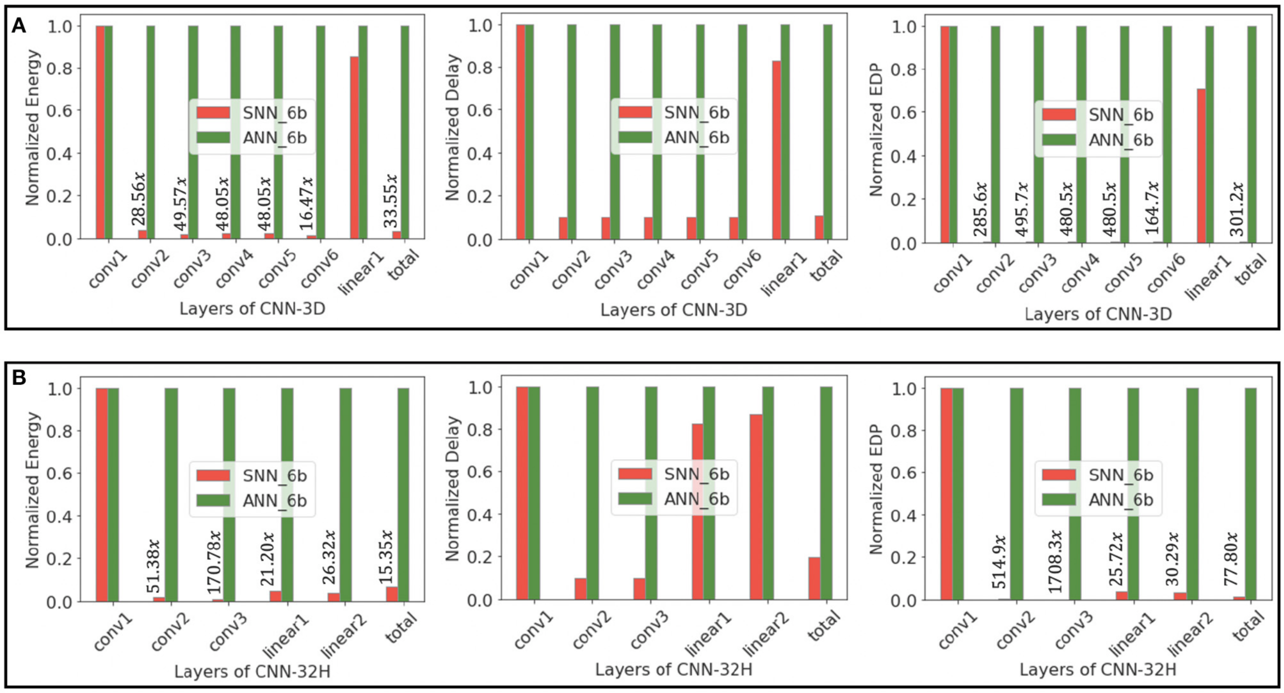


FIGURE 10 | Energy, delay, and EDP of layers of **(A)** CNN-3D and **(B)** CNN-32H architectures, comparing 6-bit ANNs and SNN (obtained via Q-STDB) models while classifying IP.

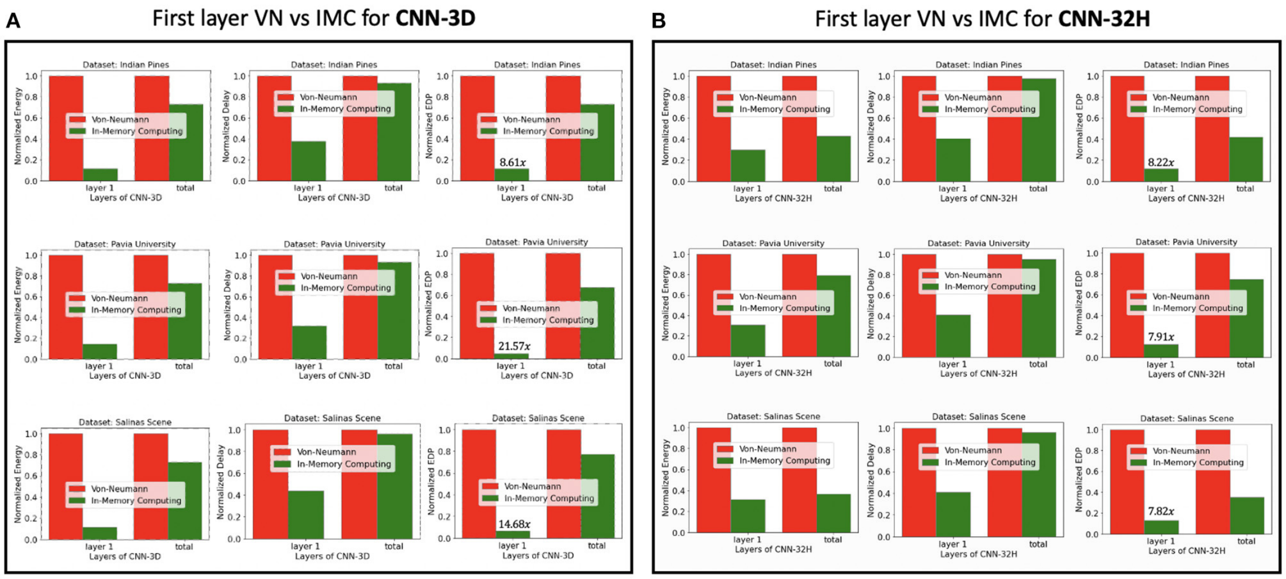


FIGURE 11 | Energy, delay, and EDP comparison of traditional digital and in-memory computing (only 1st layer) hardware for the SNN models obtained with **(A)** CNN-3D, and **(B)** CNN-32H architectures classifying Indian Pines, Pavia University, and Salinas Scene datasets.

E_{adc} for our 6-bit SNN are obtained by extending the circuit simulation results of Ali et al. (2020) with the ADC energy and delay models proposed in Gonugondla et al. (2020).

Figure 11 compares the energy, delay and the EDP of the first-layer-PIM implementation of the spiking version of CNN-3D and CNN-32H against the corresponding digital implementations for the IP, PU, and SS datasets. The improvements in the total energy, delay and EDP for CNN-3D on IP dataset, are seen to be 1.28×, 1.08× and 1.38×, respectively, over an iso-architecture-and-precision SNN implemented with digital

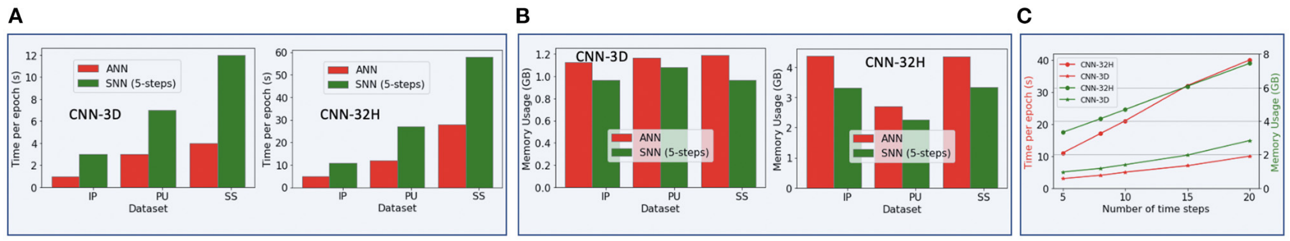


FIGURE 12 | Comparison between our baseline SOTA ANNs and proposed SNNs with 5 time steps based on **(A)** training time per epoch, and **(B)** memory usage during training. Variation of **(A,B)** with the number of time steps for the IP dataset and CNN-32H architecture are shown in **(C)**.

hardware. The improvements become 1.30×, 1.07× and 1.38×, respectively, averaging across the three datasets. However, since CNN-32H is shallower than CNN-3D, and has relatively cheaper 2D CNNs following the input 3D CNN layer, the PIM implementation in the first layer can decrease the total energy consumption significantly. The energy, delay, and EDP improvements compared to the digital implementations are estimated to be 2.12×, 1.04×, and 2.20× for CNN-32H, and 1.71×, 1.06×, and 1.79× on average across the two architectures and three datasets. Hence, the total improvements for our proposed hybrid hardware implementation (PIM in first layer and digital computing in others), coupled with our energy-aware quantization and training technique, become 953×, 17.76×, 16921× compared to iso-architecture full-precision ANNs and 76.16×, 9.2×, 700.7× compared to iso-architecture iso-precision ANNs.

Note that analog-PIM based SNNs are more cheaper in terms of energy consumption than their CNN counterparts. This is because of the reasons summarized below.

- Since CNN requires both multi-bit activations and multi-bit weights, the precision of ADCs and DACs required in analog-PIM based CNN accelerator is higher than for analog-SNN based accelerators. As is well known, ADCs are the most energy-expensive components in analog PIM accelerators, thus, this higher precision requirement leads to higher energy consumption. For example, an 8 bit ADC consumes 2× more energy compared to a 4 bit ADC (Ali et al., 2021).
- The limited precision of ADCs also necessitates 'bit-streaming' (Ankit et al., 2020), wherein multi-bit activations of CNN are serially streamed to analog-PIM crossbars and accumulated over time. Such serial streaming increases both delay and power consumption for computing.
- Finally, the higher algorithmic sparsity associated with SNN leads to reduction in energy consumption while performing analog-PIM operations. Note that this sparsity can also be leveraged by custom digital hardware.

However, the energy-delay benefit associated with analog-PIM based SNNs with respect to digital SNN implementation is lower as compared to analog-PIM based CNN in comparison digital CNN implementation. This is because CNNs require extensive energy-hungry multiplication operations, while SNNs rely on cheaper accumulate operations. Moreover, analog PIM implementation leads to increased non-idealities and can decrease the resulting test accuracy of our HSI models. As the number of weights increases after the first layer (4.5× in the 2^{nd} layer to 352.8× in the 6th layer for CNN-3D), a single layer has to be mapped over multiple memory sub-arrays. This, in turn, requires partial sums generated from individual sub-arrays to be transferred via Network-on-chip (NoC) for accumulation and generation of output activation. The NoC and associated data transfer incurs increase in energy-delay and design complexity. Hence, we choose to avoid PIM in the subsequent layers.

7.4. Training Time and Memory Requirements

We also compared the simulation time and memory requirements during the training of the baseline SOTA ANN and our proposed SNN models. Because SNNs require iterating over multiple time steps and storing the membrane potentials for each neuron, their simulation time and memory requirements can be substantially higher than their ANN counterparts. However, training with ultra low-latency, as done in this work, can bridge this gap significantly as shown in **Figure 12**. We compare the simulation time and memory usage during training of the baseline ANNs and our proposed SNN models in **Figures 12A,B**, respectively. As we can see, the training time per epoch is less than a minute for all the architectures and datasets. Moreover, the peak memory usage during training is also lower for our SNN models compared to their ANN counterparts. Hence, we conclude that our approach does not incur any significant training overhead. Note that both the training time and memory usage are higher for CNN-32H than for CNN-3D because the output feature map of its last convolutional layer is very large.

7.5. Ablation Studies

We conducted several ablation studies on combinations of affine and scale quantization during training and inference, quantized training approaches, and the efficacy of ANN-based pre-training.

7.5.1. Affine vs. Scale Quantization

Figure 13A compares inference accuracies for three different quantization techniques during the forward path of training and test on the CNN-3D architecture with the IP dataset using 6-bit quantization. Performing scale quantization during training significantly degrades performance, which further justifies our

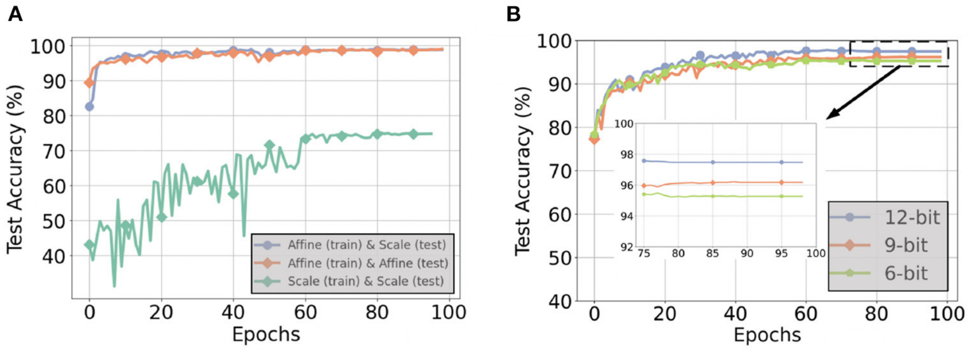


FIGURE 13 | (A) Test accuracies for different quantization techniques during the forward path of training and inference with a 6-bit CNN-3D model on the IP dataset with 5 timesteps, **(B)** Test accuracies with 6, 9, and 12-bit weight precisions for post-training quantization with a CNN-32H model on the IP dataset with 5 timesteps.

TABLE 5 | Loss in accuracy associated with use of scale quantization during inference.

Bit-precision	A. Affine (training) and Affine (inference)			B. Affine (training) and Scale (inference), Δ from Column A.		
	OA (%)	AA (%)	Kappa (%)	Δ OA (%)	Δ AA (%)	Δ Kappa (%)
6	98.89	98.39	98.21	0.21	0.05	0.01
5	98.79	98.36	98.24	0.41	0.13	0.21
4	98.50	98.01	98.07	1.42	2.37	2.53

Evaluated using the CNN-3D model on the IP dataset.

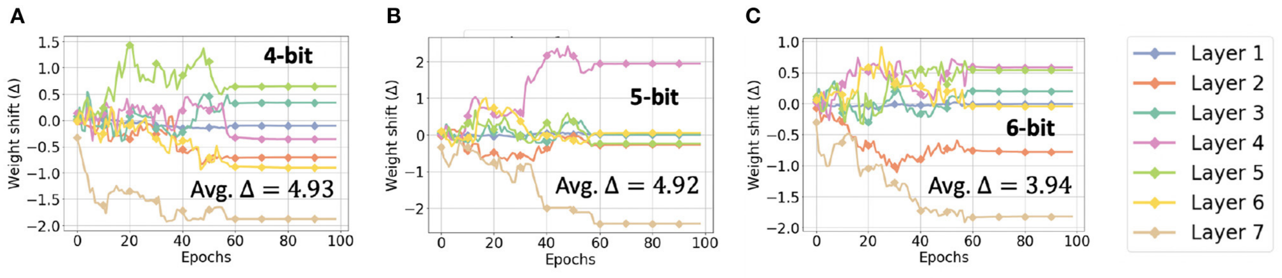


FIGURE 14 | Weight shift (Δ) in each layer of CNN-3D for **(A)** 4, **(B)** 5, and **(C)** 6-bit quantization, while classifying the IP dataset.

use of affine quantization during training. However, using scale quantization during inference results in similar accuracy as affine quantization. We further explored the gap in accuracy for 4-bit and 5-bit quantization, as summarized in **Table 5**. We observed that the accuracy gap associated with using scale quantization instead of affine quantization during inference modestly grows to 1.42% for 4-bit weights.

This small drop in relative accuracy for low bit-precisions may be attributed to the benefit of the zero factor in affine quantization on quantization error. Quantization error is typically measured by half of the width of the quantization bins, where the number of bins N_B used is independent of the type of quantization and, due to the 2's complement representation, centered around zero. However, the range of values these bins must span is smaller for affine quantization because the zero factor ensures the distribution of values is also centered at zero. This difference in range can be calculated as $\Delta = r^{scale} - r^{affine} = 2 \cdot max(w_{max}, |w_{min}|) - (w_{max} - w_{min})$. Assuming $w_{min} = -x \cdot w_{max}$,

$$\Delta = \begin{cases} (1-x)w_{max}, & \text{if } w_{max} > -w_{min} \\ (x-1)w_{max}, & \text{otherwise.} \end{cases} \tag{28}$$

As empirically shown in **Figure 14**, the average Δ across all the layers increases modestly as we decrease the bit-precision from 6 to 4. In contrast, the increase in quantization error associated with scale quantization is equal to $\frac{\Delta}{2N_B}$ and thus grows exponentially as the number of bits decrease.

7.5.2. Q-STDB vs. Post-training Quantization

PTQ with scale representation cannot always yield ultra low-precision SNNs with SOTA test accuracy. For example, as

TABLE 6 | Comparison between model performances for Q-STDB from scratch, proposed hybrid training, and ANN-SNN conversion alone.

Architecture	Dataset	A. Q-STDB *from* scratch			B. Diff. between proposed hybrid training and Q-STDB from scratch			C. Diff. between ANN-SNN conversion alone and Q-STDB from scratch		
		OA (%)	**AA (%)**	**Kappa (%)**	**Δ OA (%)**	**Δ AA (%)**	**Δ Kappa (%)**	**Δ OA (%)**	**Δ AA (%)**	**Δ Kappa (%)**
CNN-3D	IP	96.83	96.25	96.23	1.85	2.11	1.97	-39.15	-45.37	-43.35
	PU	99.38	99.04	99.17	0.14	0.13	0.16	-8.22	-10.2	-10.14
	SS	96.05	95.79	95.60	1.90	1.30	1.83	-14.61	-19.07	-15.53
CNN-32H	IP	95.93	95.36	95.40	1.53	1.37	1.49	-25.05	-28.8	-27.51
	PU	99.12	98.49	98.55	0.23	0.39	0.40	-4.16	-8.37	-4.73
	SS	96.04	95.90	95.33	1.95	1.36	1.95	-7.88	-11.71	-10.05

All cases are for 5 time steps and 6-bits.

illustrated in **Figure 13B**, for the IP dataset and CNN-32H architecture with 5 time steps, the lowest bit precision of the weights that the SNNs can be trained with PTQ for no more than 1% reduction in SOTA test accuracy is 12, two times larger bit-width than required by Q-STDB. Interestingly, the weights can be further quantized to 8-bits with less than 1% accuracy reduction if we increase the time steps to 10, but this costs latency.

7.5.3. Comparison Between Q-STDB With and Without ANN-SNN Conversion

To quantify the extent that the ANN-based pre-training helps, we performed Q-STDB from scratch (using 5 time steps), where the weights are initialized from the standard Kaiming normal distribution. The results are reported in **Table 6**, where the results in the columns labeled B and C are obtained by comparing those from the columns labeled F and B, respectively in **Table 2** with Q-STDB without ANN-SNN conversion. The results show that while Q-STDB from scratch beats conversion-only approaches, the inference accuracy can often be further improved using our proposed hybrid training combining Q-STDB and ANN-SNN conversion.

8. CONCLUSIONS AND BROADER IMPACT

In this paper, we extensively analyse the arithmetic intensities of 3D and 2D CNNs, and motivate the use of energy-efficient, low-latency, LIF-based SNNs for applications involving 3D image recognition, that requires 3D CNNs for accurate processing. We then present a quantization-aware training technique, that yields highly accurate low-precision SNNs. We propose to represent weights during the forward path of training using affine quantization and during the inference forward path using scale quantization. This provides a good trade-off between the SNN accuracy and inference complexity. We propose a 3D and hybrid combination of 3D and 2D convolutional architectures that are compatible with ANN-SNN conversion for HSI classification; the hybrid architecture incurs a small accuracy drop compared to the 3D counterpart, which shows the efficacy of 3D CNNs for HSI. Our quantized SNN models offer significant improvements in energy consumption compared to both full and low-precision ANNs for HSI classification. We also propose a PIM architecture to process the energy-expensive first layer of our direct encoded SNN to further reduce the energy, delay and EDP of the SNN models.

Our proposal results in energy-efficient SNN models that can be more easily deployed in HSI or 3D image sensors and thereby mitigates the bandwidth and privacy concerns associated with off-loading inference to the cloud. This improvement in energy-efficiency is particularly important as the applications of HSI analysis expand and the depth of the SOTA models increases (Boldrini et al., 2012).

To the best of our knowledge, this work is the first to address energy efficiency of HSI models, and can hopefully inspire more research in algorithm-hardware co-design of neural networks for size, weight, and power (SWAP) constrained HSI applications.

DATA AVAILABILITY STATEMENT

The datasets used for this study can be found in http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes. More analysis on the effect of quantization on our proposed models are included in the article/**Supplementary Material**.

AUTHOR CONTRIBUTIONS

GD conceived the idea and performed the simulations required to evaluate the efficacy of SNNs for HSI and prepared the first draft. AJ helped in the analysis of energy, delay, and EDP for the PIM implementation. All authors helped in writing the article. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported in part by the NSF CCF-1763747 award, and in part by the DARPA HR00112190120 award.

ACKNOWLEDGMENTS

We would like to thank Dr. Whitney Mason from DARPA for her insightful suggestions that helped to improve this work.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2022.815258/full#supplementary-material

REFERENCES

Agrawal, A., Jaiswal, A., Lee, C., and Roy, K. (2018). X-SRAM: Enabling in-memory boolean computations in CMOS static random access memories. *IEEE Trans. Circ. Syst. I* 65, 4219–4232. doi: 10.1109/TCSI.2018.2848999

Agrawal, A., Jaiswal, A., Roy, D., Han, B., Srinivasan, G., Ankit, A., et al. (2019). Xcel-RAM: accelerating binary neural networks in high-throughput SRAM compute arrays. *IEEE Trans. Circ. Syst. I* 66, 3064–3076. doi: 10.1109/TCSI.2019.2907488

Ali, M., Chakraborty, I., Saxena, U., Agrawal, A., Ankit, A., and Roy, K. (2021). A 35.5-127.2 tops/w dynamic sparsity-aware reconfigurable-precision compute-in-memory sram macro for machine learning. *IEEE Solid State Circ. Lett*. 4, 129–132. doi: 10.1109/LSSC.2021.3093354

Ali, M., Jaiswal, A., Kodge, S., Agrawal, A., Chakraborty, I., and Roy, K. (2020). IMAC: in-memory multi-bit multiplication and accumulation in 6t sram array. *IEEE Trans. Circ. Syst. I*. 67, 2521–2531. doi: 10.1109/TCSI.2020.2981901

Alipour-Fard, T., Paoletti, M. E., Haut, J. M., Arefi, H., Plaza, J., and Plaza, A. (2020). Multibranch selective kernel networks for hyperspectral image classification. *IEEE Geosci. Remote Sens. Lett*. 1, 1–5. doi: 10.1109/LGRS.2020.2990971

Almomani, D. A., Alauthman, M., Alweshah, M., Dorgham, O., and Albalas, F. (2019). A comparative study on spiking neural network encoding schema: implemented with cloud computing. *Cluster Comput*. 22, 419–433. doi: 10.1007/s10586-018-02891-0

Ankit, A., Hajj, I. E., Chalamalasetti, S. R., Agarwal, S., Marinella, M., Foltin, M., et al. (2020). Panther: a programmable architecture for neural network training harnessing energy-efficient reram. *IEEE Trans. Comput*. 69, 1128–1142. doi: 10.1109/TC.2020.2998456

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv preprint* arXiv:1803.09574.

Ben Hamida, A., Benoit, A., Lambert, P., and Ben Amar, C. (2018). 3-D deep learning approach for remote sensing image classification. *IEEE Trans. Geosci. Remote Sens*. 56, 4420–4434. doi: 10.1109/TGRS.2018.2818945

Biswas, A., and Chandrakasan, A. P. (2019). CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks. *IEEE J. Solid State Circ*. 54, 217–230. doi: 10.1109/JSSC.2018.2880918

Boldrini, B., Kessler, W., Rebner, K., and Kessler, R. (2012). Hyperspectral imaging: a review of best practice, performance and pitfalls for in-line and on-line applications. *J. Near Infrared Spectrosc*. 20, 483–508. doi: 10.1255/jnirs.1003

Burkitt, A. (2006). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biol. Cybern*. 95, 1–19. doi: 10.1007/s00422-006-0068-6

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis*. 113, 54–66. doi: 10.1007/s11263-014-0788-3

Chen, Q., He, G., Wang, X., Xu, J., Shen, S., Chen, H., et al. (2022). "A 67.5μJ/prediction accelerator for spiking neural networks in image segmentation," in *IEEE Transactions on Circuits and Systems II: Express Briefs* (IEEE), 574–578. doi: 10.1109/TCSII.2021.3098633

Chen, Y., Lin, Z., Zhao, X., Wang, G., and Gu, Y. (2014). Deep learning-based classification of hyperspectral data. *IEEE J. Select. Top. Appl. Earth Observat. Remote Sens*. 7, 2094–2107. doi: 10.1109/JSTARS.2014.2329330

Chen, Y., Xie, Y., Song, L., Chen, F., and Tang, T. (2020). A survey of accelerator architectures for deep neural networks. *Engineering* 6, 264–274. doi: 10.1016/j.eng.2020.01.007

Chowdhury, S. S., Lee, C., and Roy, K. (2020). Towards understanding the effect of leak in spiking neural networks. *arXiv preprint* arXiv:2006.08761. doi: 10.1016/j.neucom.2021.07.091

Comsa, I. M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., and Alakuijala, J. (2020). "Temporal coding in spiking neural networks with alpha synaptic function," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, *Vol. 1*, (Barcelona: IEEE), 8529–8533.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint* arXiv:1602.02830.

Datta, G., and Beerel, P. A. (2021). Can deep neural networks be converted to ultra low-latency spiking neural networks?. *arXiv[Preprint]*. arXiv: 2112.12133. Available online at: https://arxiv.org/pdf/2112.12133.pdf

Datta, G., Kundu, S., and Beerel, P. A. (2021). Training energy-efficient deep spiking neural networks with single-spike hybrid input encoding. *arXiv preprint* arXiv:2107.12374. doi: 10.1109/IJCNN52387.2021.9534306

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, *Vol*. 1 (Killarney: IEEE), 1–8.

Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., and Neftci, E. (2016). "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *2016 IEEE International Conference on Rebooting Computing (ICRC)* (San Diego, CA: IEEE), 1–8.

Dong, Q., Jeloka, S., Saligane, M., Kim, Y., Kawaminami, M., Harada, A., et al. (2018). A 4+2T SRAM for searching and in-memory computing with 0.3-V $v_d dmin$. *IEEE J. Solid State Circ*. 53, 1006–1015. doi: 10.1109/JSSC.2017.2776309

Ganesan, S. (2015). *Area, delay and power comparison of adder topologies* (Maseters' Thesis). University of Texas at Austin.

Gonugondla, S. K., Sakr, C., Dbouk, H., and Shanbhag, N. R. (2020). Fundamental limits on energy-delay-accuracy of in-memory architectures in inference applications. *ArXiv*, abs/2012.13645. doi: 10.1145/3400302.3416344

Han, S., Mao, H., and Dally, W. J. (2015a). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint* arXiv:1510.00149.

Han, S., Pool, J., Tran, J., and Dally, W. (2015b). "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 1135–1143.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2018). Mask R-CNN. *arXiv preprint* arXiv:1703.06870. doi: 10.1109/ICCV.2017.322

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV: IEEE), 770–778.

Hien Van, N.guyen, Banerjee, A., and Chellappa, R. (2010). "Tracking via object reflectance using a hyperspectral video camera," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops, Vol*. 1 (San Francisco, CA: IEEE), 44–51.

Horowitz, M. (2014). "1.1 Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (San Francisco, CA: IEEE), 10–14.

Hu, Y., Tang, H., and Pan, G. (2018). Spiking deep residual network. *arXiv preprint* arXiv:1805.01352.

Jain, S. R., Gural, A., Wu, M., and Dick, C. H. (2020). Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. *arXiv preprint* arXiv:1903.08066.

Jaiswal, A., Chakraborty, I., Agrawal, A., and Roy, K. (2019). 8T SRAM cell as a multibit dot-product engine for beyond von neumann computing. *IEEE Trans. Very Large Scale Integr. Syst*. 27, 2556–2567. doi: 10.1109/TVLSI.2019.2929245

Kang, M., Lim, S., Gonugondla, S., and Shanbhag, N. R. (2018). An in-memory VLSI architecture for convolutional neural networks. *IEEE J. Emerg. Select. Top. Circ. Syst*. 8, 494–505. doi: 10.1109/JETCAS.2018.2829522

Kheradpisheh, S. R., and Masquelier, T. (2020). Temporal backpropagation for spiking neural networks with one spike per neuron. *Int. J. Neural Syst*. 30:2050027. doi: 10.1142/S0129065720500276

Kim, J., Kim, H., Huh, S., Lee, J., and Choi, K. (2018). Deep neural networks with weighted spikes. *Neurocomputing* 311, 373–386. doi: 10.1016/j.neucom.2018.05.087

Kim, Y., and Panda, P. (2021). Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *arXiv preprint* arXiv:2010.01729. doi: 10.3389/fnins.2021.773954

Krishnapuram, B., Carin, L., Figueiredo, M. A. T., and Hartemink, A. J. (2005). Sparse multinomial logistic regression: fast algorithms and generalization bounds. *IEEE Trans. Pattern Anal. Mach. Intell*. 27, 957–968. doi: 10.1109/TPAMI.2005.127

Krizhevsky, A. (2012). "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 1097–1105.

Kundu, S., Datta, G., Pedram, M., and Beerel, P. A. (2021a). "Spike-thrift: towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 3953–3962.

Kundu, S., Datta, G., Pedram, M., and Beerel, P. A. (2021b). Towards low-latency energy-efficient deep snns via attention-guided compression. *arXiv preprint* arXiv:2107.12445.

Kundu, S., Pedram, M., and Beerel, P. A. (2021c). "HIRE-SNN: harnessing the inherent robustness of energy-efficient deep spiking neural networks by training with crafted input noise," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5209–5218

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci*. 14,119. doi: 10.3389/fnins.2020.00119

Lee, H., and Kwon, H. (2017). Going deeper with contextual cnn for hyperspectral image classification. *IEEE Trans. Image Process*. 26, 4843–4855. doi: 10.1109/TIP.2017.2725580

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci*. 10, 508. doi: 10.3389/fnins.2016.00508

Li, D., Chen, X., Becchi, M., and Zong, Z. (2016). "Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)* (*BDCloud-SocialCom-SustainCom*), *Vol*. 1 (Atlanta, GA: IEEE), 477–484.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2018). Rethinking the value of network pruning. *arXiv preprint* arXiv:1810.05270.

Luo, Y., Zou, J., Yao, C., Zhao, X., Li, T., and Bai, G. (2018). "HSI-CNN: a novel convolution neural network for hyperspectral image," in *2018 International Conference on Audio, Language and Image Processing (ICALIP), Vol. 1* (Shanghai: IEEE), 464–469.

Melgani, F., and Bruzzone, L. (2004). Classification of hyperspectral remote sensing images with support vector machines. *IEEE Trans. Geosci. Remote Sens*. 42, 1778–1790. doi: 10.1109/TGRS.2004.831865

Meng, Z., Zhao, F., Liang, M., and Xie, W. (2021). Deep residual involution network for hyperspectral image classification. *Remote Sens*. 13. doi: 10.3390/rs13163055

Moons, B., Goetschalckx, K., Van Berckelaer, N., and Verhelst, M. (2017). "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers, Vol. 1* (Pacific Grove, CA: IEEE), 1921–1925.

Moons, B., Uytterhoeven, R., Dehaene, W., and Verhelst, M. (2017). "14.5 envision: a 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *2017 IEEE International Solid-State Circuits Conference (ISSCC), Vol. 1* (San Francisco, CA: IEEE), 246–247.

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag*. 36, 51–63. doi: 10.1109/MSP.2019.2931595

Pal, M. (2003). "Random forests for land cover classification," in *IGARSS 2003. 2003 IEEE International Geoscience and Remote Sensing Symposium. Proceedings (IEEE Cat. No.03CH37477) Vol. 6* (Toulouse: IEEE), 3510–3512.

Panda, P., and Roy, K. (2016). Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. *arXiv preprint* arXiv:1602.01510. doi: 10.1109/IJCNN.2016.7727212

Papp, A., Pegoraro, J., Bauer, D., Taupe, P., Wiesmeyr, C., and Kriechbaum-Zabini, A. (2020). Automatic annotation of hyperspectral images and spectral signal classification of people and vehicles in areas of dense vegetation with deep learning. *Remote Sens*. 12. doi: 10.3390/rs12132111

Park, J., Lee, J., and Jeon, D. (2019). "A 65nm 236.5nJ/Classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC), Vol. 1* (San Francisco, CA: IEEE), 140–142.

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci*. 12, 774. doi: 10.3389/fnins.2018.00774

Rathi, N., Panda, P., and Roy, K. (2017). STDP based pruning of connections and weight quantization in spiking neural networks for energy efficient recognition. *arXiv preprint* arXiv:1710.04734.

Rathi, N., and Roy, K. (2020). DIET-SNN: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint* arXiv:2008.03658.

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint* arXiv:2005.01807.

Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell*. 39, 1137–1149. doi: 10.1109/TPAMI.2016.2577031

Repala, V. K., and Dubey, S. R. (2019). Dual CNN models for unsupervised monocular depth estimation. *arXiv preprint* arXiv:1804.06324. doi: 10.1007/978-3-030-34869-4_23

Roy, S. K., Krishna, G., Dubey, S. R., and Chaudhuri, B. B. (2020). HybridSN: exploring 3-D–2-D CNN feature hierarchy for hyperspectral image classification. *IEEE Geosci. Remote Sens. Lett*. 17, 277–281. doi: 10.1109/LGRS.2019.2918719

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci*. 11, 682. doi: 10.3389/fnins.2017.00682

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci*. 13, 95. doi: 10.3389/fnins.2019.00095

Simon, W., Galicia, J., Levisse, A., Zapater, M., and Atienza, D. (2019). "A fast, reliable and wide-voltage-range in-memory computing architecture," in *2019 56th ACM/IEEE Design Automation Conference (DAC), Vol. 1*. 1–6.

Song, W., Li, S., Fang, L., and Lu, T. (2018). Hyperspectral image classification with deep feature fusion network. *IEEE Trans. Geosci. Remote Sens*. 56, 3173–3184. doi: 10.1109/TGRS.2018.2794326

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1929–1958. Available online at: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Sulaiman, M. B. G., Juang, K. C., and Lu, C. C. (2020). "Weight quantization in spiking neural network for hardware implementation," in *2020 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan), Vol. 1* (Taoyuan: IEEE), 1–2.

Wan, Y., Fan, Y., and Jin, M. (2021). Application of hyperspectral remote sensing for supplementary investigation of polymetallic deposits in huaniushan ore region, northwestern china. *Sci. Rep*. 11, 440. doi: 10.1038/s41598-020-79864-0

Wu, H., Judd, P., Zhang, X., Isaev, M., and Micikevicius, P. (2020). Integer quantization for deep learning inference: principles and empirical evaluation. *arXiv preprint* arXiv:2004.09602.

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). Direct training for spiking neural networks: Faster, larger, better. *Proc. AAAI Conf. Artif. Intell.* 33, 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Xia, J., Yokoya, N., and Iwasaki, A. (2017). Hyperspectral image classification with canonical correlation forests. *IEEE Trans. Geosci. Remote Sens.* 55, 421–431. doi: 10.1109/TGRS.2016.2607755

Zheng, Z., Zhong, Y., Ma, A., and Zhang, L. (2020). FPGA: Fast patch-free global learning framework for fully end-to-end hyperspectral image classification. *IEEE Trans. Geosci. Remote Sens*. 58, 5612–5626. doi: 10.1109/TGRS.2020.2967821

Zhong, Z., Li, J., Luo, Z., and Chapman, M. (2018). Spectral–spatial residual network for hyperspectral image classification: a 3-D deep learning framework. *IEEE Trans. Geosci. Remote Sens*. 56, 847–858. doi: 10.1109/TGRS.2017.2755542

Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. (2017). Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint* arXiv:1702.03044.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Datta, Kundu, Jaiswal and Beerel. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

ORIGINAL RESEARCH
published: 14 April 2022
doi: 10.3389/fnins.2022.760298

Backpropagation With Sparsity Regularization for Spiking Neural Network Learning

Yulong Yan, Haoming Chu, Yi Jin, Yuxiang Huan, Zhuo Zou and Lirong Zheng**

School of Information Science and Technology, Fudan University, Shanghai, China

The spiking neural network (SNN) is a possible pathway for low-power and energy-efficient processing and computing exploiting spiking-driven and sparsity features of biological systems. This article proposes a sparsity-driven SNN learning algorithm, namely backpropagation with sparsity regularization (BPSR), aiming to achieve improved spiking and synaptic sparsity. Backpropagation incorporating spiking regularization is utilized to minimize the spiking firing rate with guaranteed accuracy. Backpropagation realizes the temporal information capture and extends to the spiking recurrent layer to support brain-like structure learning. The rewiring mechanism with synaptic regularization is suggested to further mitigate the redundancy of the network structure. Rewiring based on weight and gradient regulates the pruning and growth of synapses. Experimental results demonstrate that the network learned by BPSR has synaptic sparsity and is highly similar to the biological system. It not only balances the accuracy and firing rate, but also facilitates SNN learning by suppressing the information redundancy. We evaluate the proposed BPSR on the visual dataset MNIST, N-MNIST, and CIFAR10, and further test it on the sensor dataset MIT-BIH and gas sensor. Results bespeak that our algorithm achieves comparable or superior accuracy compared to related works, with sparse spikes and synapses.

Keywords: spiking neural network, backpropagation, sparsity regularization, spiking sparsity, synaptic sparsity

OPEN ACCESS

Edited by:
Angeliki Pantazi,
IBM Research, Switzerland

Reviewed by:
Charles Augustine,
Intel, United States
Chankyu Lee,
Intel, United States

****Correspondence:***
Zhuo Zou
zhuo@fudan.edu.cn
Lirong Zheng
lrzheng@fudan.edu.cn

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *17 August 2021*
Accepted: *22 February 2022*
Published: *14 April 2022*

Citation:
Yan Y, Chu H, Jin Y, Huan Y, Zou Z and Zheng L (2022) Backpropagation With Sparsity Regularization for Spiking Neural Network Learning. Front. Neurosci. 16:760298. doi: 10.3389/fnins.2022.760298

1. INTRODUCTION

Artificial intelligence (AI) has shown impressive abilities in various tasks such as computer vision, natural language processing, and decision making. For example, AlphaGo Zero defeated the world champion of the game of Go (Silver et al., 2017). However, the power consumption of AlphaGo Zero is about 1kW (Frenkel et al., 2021), which is 50× higher than the 20W power budget of the human brain (Roy et al., 2019). The brain-inspired spiking neural network (SNN) plays an important role in addressing the issue of AI energy efficiency. SNN exchanges information through binary spikes between synapses and performs intensive calculation only when spikes are received. Dedicated SNN hardware such as TrueNorth (Akopyan et al., 2015), Loihi (Davies et al., 2018), Tianjic (Pei et al., 2019), and MindWare (Ding et al., 2021) can reduce energy consumption from sparse spikes and synapses through spike-driven computing architecture. Despite the merits of improving energy efficiency, there remain a lot of challenges ahead of the SNN in sparsity learning algorithms and efficient network exploration.

The commonly adopted SNN learning algorithms can be summarized into three different types as follows. (1) *Conversion-based learning*. It uses the same SNN structure as an artificial neural network (ANN) and converts the parameters of the learned ANN to SNN. One conversion idea is to use the spiking firing rate (FR) of SNN to quantify the floating value of ANN and establish an approximate mapping between the parameters of two networks (Sengupta et al., 2019; Kim et al., 2020). This kind of conversion uses rate coding, resulting in dense spikes. Another idea is to use spike timing to represent the floating value in ANN. Methods like time-to-first-spike (TTFS) conversion (Rueckauer and Liu, 2018) and few spikes conversion (FS-conversion) (Stöckl and Maass, 2021) use temporal coding to protect spiking sparsity. However, the time domain is used for coding so that temporal processing structure such as recurrent neural network (RNN) cannot be converted. (2) *Plasticity-based learning*. It is a kind of biologically inspired algorithm. The most famous spike-timing-dependent plasticity (STDP) adjusts synaptic weight according to the spike order between the pre- and post-synaptic neurons. The role of STDP is feature clustering. Combined with lateral inhibition structure, STDP can realize unsupervised classification (Diehl and Cook, 2015; Białas and Mańdziuk, 2021). Reward-modulated STDP draws on the eligibility trace of reinforcement learning to realize supervised learning to further improve performance (Mozafari et al., 2018). The plasticity-based learning algorithm is skilled in computation overhead and weak in network accuracy. (3) *Gradient-based learning*. Like the learning of ANN, it updates the parameters of SNN according to the gradient information from backpropagation. A recent study by Lillicrap et al. (2020) suggests that a similar propagation mechanism may exist in the brain. Spatio-temporal backpropagation (STBP) (Wu et al., 2018, 2019) provides advanced accuracy by calculating gradient in the spatio-temporal domain. Deep continuous local learning (DECOLLE) (Kaiser et al., 2020) reduces the memory overhead through the local error function. Spike-train level recurrent SNN backpropagation (ST-RSBP) (Zhang and Li, 2019) further supports the recurrent layer, to deal with temporal information by mimicking import feedback structure in the brain (Luo, 2021). The above algorithms focus on the accuracy improvement and lack consideration in the sparsity issue. Compared with local learning based on plasticity, gradient-based learning requires global information. It improves accuracy and brings additional calculation burdens. However, in the offline learning scenario, the computational overhead of SNN is mainly contributed by inference rather than learning. Therefore, reducing the computational overhead in inference through sparsity optimization and ensuring accuracy by gradient-based learning, become the major motivation of this work.

Another kind of SNN algorithm aims to improve synaptic sparsity by pruning. Existing studies explore different pruning standards. Liang et al. (2021) prune synapses through random patterns and quantify synaptic weight to reduce storage overhead. Rathi et al. (2018) utilize the synaptic weight threshold to prune and optimize storage through weight quantization and sharing. Cho et al. (2019) prune long-range synaptic connections based on the small world theory of the nervous system. Nguyen et al. (2021) combine pruning with STDP and use the weight adjustment record as the pruning standard. Shi et al. (2019) use spiking count as the pruning threshold and propose a soft pruning method to reduce the computation overhead in learning. Moreover, Guo et al. (2020) prune the neurons rather than synapses according to spiking count, providing a new perspective of sparsity exploration.

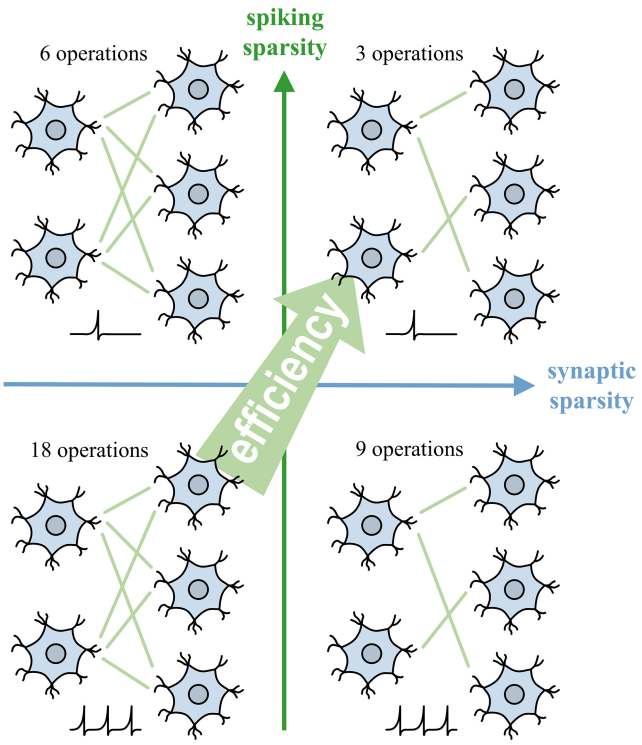


FIGURE 1 | Spiking sparsity and synaptic sparsity facilitate the efficiency of SNN by reducing the number of synaptic operations.

SNN can perform sparse computing due to the event-driven feature. At the same time, the synaptic operation uses membrane potential accumulation instead of matrix multiplication and addition in traditional ANN, which reduces the amount of calculation. In recent years, similar methods have been proposed in the field of ANN to reduce the number of operations. Binarized neural network (BNN) (Hubara et al., 2016) and XNOR-Net (Rastegari et al., 2016) introduce binarized weights and activations and replace most arithmetic operations on synapses with bit-wise operations. AdderNet (Chen et al., 2020) builds ANN only through addition to avoid the expensive multiplication operation and achieves acceleration with low energy consumption. Beyond that, Bartol et al. (2015) believe each synapse stores about 4.7 bits of information. Quantization of synaptic weights can also be an idea to further optimize computational speed and compress storage overhead.

This work proposes an SNN learning algorithm, namely backpropagation with sparsity regularization (BPSR) to facilitate sparsity. As shown in **Figure 1**, the sparse spikes reduce the amount of information that subsequent neurons need to process, meanwhile the sparse synapses prevent each spike from causing intensive calculations. The proposed BPSR enables SNN to improve sparsity during learning and achieve satisfactory energy efficiency in inference. The backpropagation takes advantage of temporal information and adapts the brain-like recurrent structure. BPSR balances the accuracy and FR by combining backpropagation with spiking regularization. Inspired by the fact that the brain learns through synaptic rearrangement (Dempsey et al., 2022), rewiring mechanism is proposed to explore efficient SNN structures, which uses the weight and gradient to regulate synaptic pruning and growth. The experimental result is consistent with the concept that the proposed BPSR can achieve low FR with high accuracy. Spiking sparsity is proved to be beneficial to SNN learning (Tang et al., 2017), because of the suppression of information redundancy. BPSR not only improves the synaptic sparsity but also generates a bionic structure similar to the nervous system of *Caenorhabditis elegans* (*C. elegans*). The result on the visual MNIST dataset (LeCun et al., 1998) with rank order coding (Thorpe and Gautrais, 1998), neuromorphic-MNIST (N-MNIST) (Orchard et al., 2015), and CIFAR10 (Krizhevsky et al., 2009) reach the accuracy of 98.33, 99.21, and 90.74%, respectively. The evaluation on MNIST also shows 30× the inference overhead advantage compared to other SNN works. With post-training quantization (PTQ), SNN can achieves 15× efficiency compared to BNN with 0.22% accuracy drop. BPSR is further tested on sensor datasets like MIT-BIH arrhythmia (Moody and Mark, 2001) and gas senor (Vergara et al., 2013), which achieves 98.41 and 98.30% accuracy.

The remainder of this article is organized as follows. In Section 2, the backpropagation with sparsity regularization is introduced. The suggested heterogeneous neuron dynamic model, the loss function with regularization, and the backpropagation algorithm on the flat and recurrent SNN layers are detailed. In Section 3, the rewiring based on weight and gradient and the corresponding implementation process is introduced. In Section 4, the effect of the proposed BPSR algorithm is tested by experiments, and comparisons with related works on various datasets are reported. In Section 5, we summarize this work and make a discussion.

2. BACKPROPAGATION WITH SPARSITY REGULARIZATION

The backpropagation algorithm with regularization updates SNN parameters while improving sparsity. The spiking sparsity is implemented through backpropagation and spiking regularization. Synaptic sparsity requires the cooperation of regularization and the rewiring mechanism in Section 3. Firstly, a heterogeneous leaky integrate-and-fire (LIF) neuron dynamic model and its differential approximation are suggested. Secondly, a classification loss function with spiking regularization and synaptic regularization is introduced. Finally, the backpropagation algorithm for the flat SNN layer and the brain-like recurrent SNN layer is detailed, respectively.

2.1. Heterogeneous Leaky Integrate-and-Fire Model

As one of the most commonly used neuron models, LIF describes the dynamic process of neurons in SNN. The membrane potential of neurons increases under the stimulation of spikes and leaks spontaneously with time. When the potential reaches the spiking threshold, the neuron generates a spike and resets the membrane potential. In addition, we extend the LIF description to the spiking recurrent layer and support neurons with different time coefficients (heterogeneous), to utilize the brain-like structure and temporal features. We hierarchically describe the SNN. For the *n*-th layer, the LIF process can be described by equations in the discrete-time domain:

$$u_i^t = u_i^{t-1} \cdot \tau_i \cdot \overline{s_i^{t-1}} + \sum_{j \in \mathbb{L}^{n-1}} w_{ij} \cdot x_j^t + \sum_{k \in \mathbb{L}^n} w_{ik} \cdot s_k^{t-1} + b_i, i \in \mathbb{L}^n \tag{1}$$

$$s_i^t = g(u_i^t - U_{th}) \tag{2}$$

where u_i^t is the membrane potential of i-th neuron in layer $\mathbb{L}^n$ at time t ($\mathbb{L}^n$ represents the set of neurons in the n-th layer). $s_i^t \in \{0, 1\}$ is a boolean value where $s_i^t = 1$ denotes a spike activity. $\overline{s_i^{t-1}}$ means to take a logical 'not' operation on s_i^{t-1}. $\tau_i \in [0, 1]$ is the leakage time coefficient, which achieves neuronal heterogeneity. This allows the neuron model to be heterogeneous and facilitates temporal feature extracting. Multiply u_i^{t-1} by $\tau_i \cdot \overline{s_i^{t-1}}$ controls whether the membrane potential leaks by τ_i or drops to the resting potential 0. The neuron bias is denoted by b_i, leading to self-excitation or self-suppression. x_j^t is the input spike from the j-th neuron in layer $\mathbb{L}^{n-1}$. It should be noted that, for the calculation of layer $\mathbb{L}^{n+1}$, $x: = s_i^t$, $i \in \mathbb{L}^n$. In this way, spikes are transmitted layer by layer. As shown in **Figure 2**, the SNN layer can be classified as **Figure 2A** the flat layer and **Figure 2C** the recurrent layer. For the flat layer, w_{ij} represents inter-layer synapse from the j-th neuron in layer $\mathbb{L}^{n-1}$ to i-th neuron in layer $\mathbb{L}^n$. For the recurrent, w_{ik} is appended to indicate intra-layer synapse inside layer $\mathbb{L}^n$, which has the ability to extract temporal features due to the brain-like structure. The Heaviside function $g(\cdot)$ generates a spike when u_i^t is greater than or equal to the spiking threshold U_{th}. Heaviside function and the adopted differential approximation are expressed as:

$$g(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}, \quad g'(x) = \frac{\alpha}{\sqrt{\pi}} e^{-\alpha^2 x^2} \tag{3}$$

Backpropagation requires a differentiable path. The derivative of the Heaviside function $g(\cdot)$ is the Dirac function $\delta(\cdot)$, whose value is $+\infty$ at 0 and impossible to perform the calculation. Thus, the Gaussian function is introduced as the differential approximation of the Heaviside function, where α controls the shape of the function.

2.2. Loss Function With Sparsity Regularization

The loss function measures the error for a classification task and the sparsity of SNN, which is defined as follows. The first

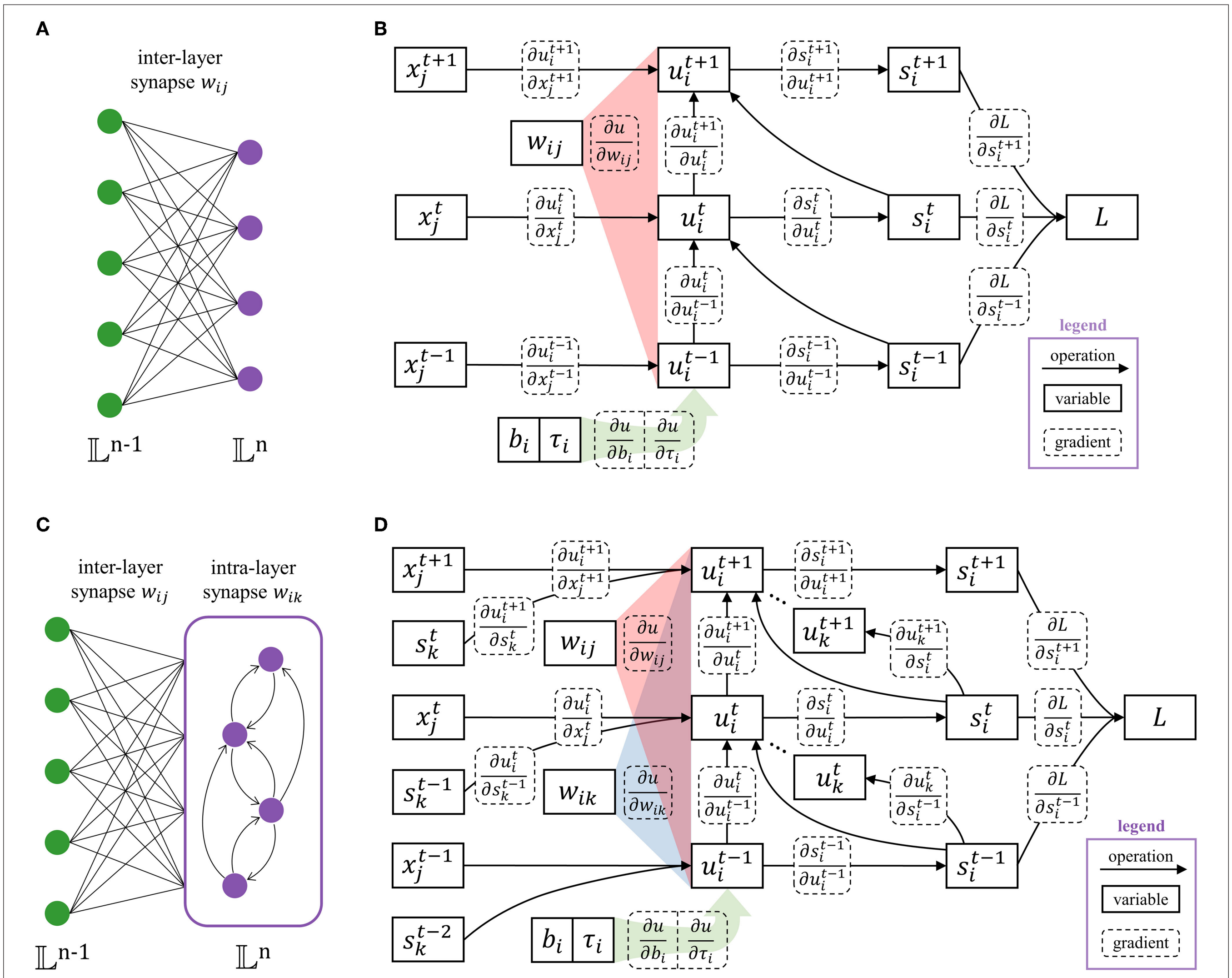

A
inter-layer synapse w_{ij}
$\mathbb{L}^{n-1}$
$\mathbb{L}^{n}$
B
C
inter-layer synapse w_{ij}
intra-layer synapse w_{ik}
D
legend
operation
variable
gradient

FIGURE 2 | The structure of the SNN layer contains **(A)** the flat layer with only inter-layer synapses ($w_{ik} = 0$), and **(C)** the recurrent layer with intra-layer synapses ($w_{ij} \neq 0$). The corresponding computational graphs are **(B,D)**, respectively. The legends of arithmetic operations, neuron state variables, and gradients are marked in the lower right corner.

term measures the classification error through softmax and cross-entropy functions. The second and third terms achieve spiking sparsity and synaptic sparsity, respectively.

$$L = \underbrace{-\sum_{c} y_c \log(p_c)}_{\text{classification error}} + \underbrace{\frac{\lambda_s}{2}\sum_{i\notin\mathbb{L}^{\mathrm{N}}}\sum_{t}\left\|s_i^t\right\|_2^2}_{\text{spiking sparsity}} + \underbrace{\lambda_w \sum_{w\in\mathbb{L}^{\mathrm{n}}}\|w\|_1}_{\text{synaptic sparsity}} \tag{4}$$

$$p_c = \text{softmax}(k\cdot\sum_{t} s_c^t) = \frac{\exp(\sum_t k\cdot s_c^t)}{\sum_{i\in\mathbb{L}^{\mathrm{N}}}\exp(\sum_t k\cdot s_i^t)} \tag{5}$$

where y_c is the ground-truth label of one-hot coding for the c-th class. p_c is the predicted probability given by the output layer $\mathbb{L}^{\mathrm{N}}$. p_c is calculated by summing of the output spikes, multiplied by factor k, and then processing by softmax function. The factor $k = \frac{10}{T}$ corrects the softmax error by scaling the sum of spikes in the time window T. λ_s is the coefficient of l_2 regularization for spiking sparsity. It takes effect on the spikes of the SNN layer, except for the output layer to ensure classification accuracy. λ_w is the coefficient of l_1 regularization for the sparsity of synaptic weight, which is effective for all layers of the SNN.

The regularizations of spiking sparsity and synaptic sparsity have similar forms and can promote each other. But in essence, their mechanism is different (as shown in **Table 1**). The goal of spiking regularization is to reduce FR while ensuring guaranteed accuracy. Therefore, the regular term adjusts the parameters w_{ij}, w_{ik}, b_i, and τ_i to punish dense spikes. Synaptic regularization works together with the rewiring mechanism in Section 3 to realize pruning of the weight w_{ij} and w_{ik}. The gradient of spiking regularization is calculated by the chain

TABLE 1 | Comparison between spiking and synaptic regularization.

Regularization	Purpose	Scope	Gradient of synaptic weight
Spiking	Reduce FR while ensuring accuracy	w_{ij}, w_{ik}, b_i, τ_i	$\nabla_w \propto \lambda_s \cdot s_i^t g'(u_i^t)x_j^t$
Synaptic	Combine with rewiring for pruning	w_{ij}, w_{ik}	$\nabla_w = \lambda_w \cdot \text{sign}(w)$

rule. When either input spike x_j^t or output spike s_i^t is 0, the spiking regularization will not be affected. The gradient of synaptic regularization is calculated directly, and it works continuously until the weight is set to 0. As regularization, both of them improve network performance by preventing overfitting. The difference is that the principle of spiking regularization is simplifying feature expression to improve generalization ability. While synaptic regularization and rewiring work together to take effect by reducing the dimensionality of the parameter space.

2.3. Backpropagation in Flat Layer

The error information is propagated through the gradient and the parameters of SNN are updated accordingly. Therefore, it is necessary to derive the gradient of the loss function to each parameter. There are only inter-layer synaptic connections in the flat layer structure and $w_{ik} = 0$. The computational graph of the flat layer and the corresponding gradient path are shown in **Figure 2B**. For the output layer $\mathbb{L}^{\text{N}}$, the partial derivative $\partial L/\partial s$ can be directly calculated. For the non-output layer ($\mathbb{L}^{\text{n}}, n < N$), the partial derivative $\partial L/\partial s$ is the $\partial L/\partial x$ of the following layer, plus the spiking regularization term.

$$\frac{\partial L}{\partial s_i^t} = \begin{cases} p_i - y_i, \quad i \in \mathbb{L}^{\text{N}} \\ \dfrac{\partial L}{\partial x_j^t} + \lambda_s \cdot x_j^t, \quad i \in \mathbb{L}^{\text{n}},\ j \in \mathbb{L}^{\text{n+1}},\ n < N \end{cases} \tag{6}$$

The spike s_i^t is a function of the membrane potential u_i^t, and the membrane potential changes over time. Although u_i^t is a function of s_i^{t-1} in Equation (1), s_i^{t-1} only gates the information flow in potential along time. Unlike u_i^t accumulating information to s_i^t, or x_j^{t-1} passing w_{ij} of information to u_i^t, s_i^{t-1} has no information contribution to u_i^t. Thus, $\partial u_i^t/\partial s_i^{t-1}$ is ignored in backpropagation. $\partial L/\partial u_i^t$ is expressed as:

$$\frac{\partial L}{\partial u_i^t} = \begin{cases} \dfrac{\partial L}{\partial s_i^t} \cdot \dfrac{\partial s_i^t}{\partial u_i^t} = \dfrac{\partial L}{\partial s_i^t} \cdot g'(u_i^t - U_{th}), \quad t = T \\ \dfrac{\partial L}{\partial s_i^t} \cdot \dfrac{\partial s_i^t}{\partial u_i^t} + \dfrac{\partial L}{\partial u_i^{t+1}} \cdot \dfrac{\partial u_i^{t+1}}{\partial u_i^t} \\ = \dfrac{\partial L}{\partial s_i^t} \cdot g'(u_i^t - U_{th}) + \dfrac{\partial L}{\partial u_i^{t+1}} \cdot \tau_i \cdot \overline{s_i^t}, \quad t < T \end{cases} \tag{7}$$

The part of $t < T$ in Equation (7) takes into account all the errors after time t through iterative calculation and reduces the algorithm complexity to $O(t)$. Assuming the direct error from the loss function at time t is $\varepsilon^t = \partial L/\partial s_i^t \cdot \partial s_i^t/\partial u_i^t$. **Figure 3** shows how the influence from the subsequent time is calculated by one addition and multiplication when $t = T, T-1, T-2$.

Once the gradient to u_i^t is obtained, the gradients to each parameter and input spike are easy to calculate by the following equations, where i belongs to layer $\mathbb{L}^{\text{n}}$ and T is the time window. The initial value $u_i^0 = s_i^0 = 0$. Learning shared parameters such as convolution weights or homogeneous leakage coefficients can be realized by summing the gradient of shared weight. Potential changes well beyond the threshold have no effect, so excessively large w_{ij} and b_i are meaningless and clamped to $[-U_{th}, +U_{th}]$ accordingly. τ is also limited to its range of values $[0, 1]$.

$$\frac{\partial L}{\partial x_j^t} = \sum_{i \in \mathbb{L}^{\text{n}}} \frac{\partial L}{\partial u_i^t} \cdot \frac{\partial u_i^t}{\partial x_j^t} = \sum_{i \in \mathbb{L}^{\text{n}}} \frac{\partial L}{\partial u_i^t} \cdot w_{ij} \tag{8}$$

$$\begin{aligned} \frac{\partial L}{\partial w_{ij}} &= \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^t} \cdot \frac{\partial u_i^t}{\partial w_{ij}} + \lambda_w \cdot \text{sign}(w_{ij}) \\ &= \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^t} \cdot x_j^t + \lambda_w \cdot \text{sign}(w_{ij}) \end{aligned} \tag{9}$$

$$\frac{\partial L}{\partial b_i} = \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^t} \cdot \frac{\partial u_i^t}{\partial b_i} = \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^t} \tag{10}$$

$$\frac{\partial L}{\partial \tau_i} = \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^t} \cdot \frac{\partial u_i^t}{\partial \tau_i} = \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^t} \cdot u_i^{t-1} \cdot \overline{s_i^{t-1}} \tag{11}$$

2.4. Backpropagation in Recurrent Layer

The intra-layer synaptic connections exist in the recurrent layer, i.e., $w_{ik} \neq 0$. This makes the computational graph of the recurrent layer and the gradient path are different from the flat layer, which are shown in **Figure 2D**. The calculation method of the partial derivative $\partial L/\partial s_i^t$ still follows Equation (6). Considering the intra-layer connection within the recurrent, $\partial L/\partial u_i^t$ is modified to:

$$\frac{\partial L}{\partial u_i^t} = \begin{cases} \dfrac{\partial L}{\partial s_i^t} \cdot \dfrac{\partial s_i^t}{\partial u_i^t} = \dfrac{\partial L}{\partial s_i^t} \cdot g'(u_i^t - U_{th}), \quad t = T \\ \dfrac{\partial L}{\partial s_i^t} \cdot \dfrac{\partial s_i^t}{\partial u_i^t} + \dfrac{\partial L}{\partial u_i^{t+1}} \cdot \dfrac{\partial u_i^{t+1}}{\partial u_i^t} + \sum_{k \in \mathbb{L}^{\text{n}}} \dfrac{\partial L}{\partial u_k^{t+1}} \cdot \dfrac{\partial u_k^{t+1}}{\partial s_i^t} \cdot \dfrac{\partial s_i^t}{\partial u_i^t} \\ = \left[\dfrac{\partial L}{\partial s_i^t} + \sum_{k \in \mathbb{L}^{\text{n}}} \dfrac{\partial L}{\partial u_k^{t+1}} \cdot w_{ki} \right] \cdot g'(u_i^t - U_{th}) \\ + \dfrac{\partial L}{\partial u_i^{t+1}} \cdot \tau_i \cdot \overline{s_i^t}, t < T \end{cases} \tag{12}$$

Note that for the intra-layer synaptic weight, we swap the subscripts of the input and the output neurons (denoted as

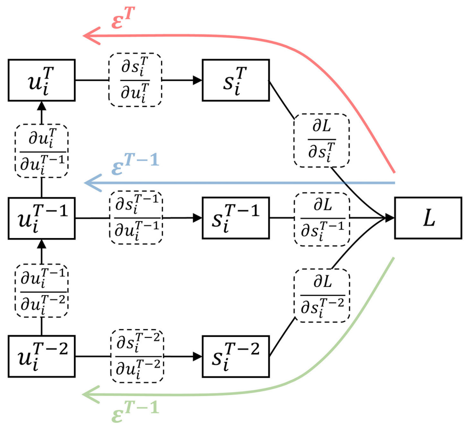
$$\frac{\partial L}{\partial u_i^T} = \frac{\partial L}{\partial s_i^T} \cdot \frac{\partial s_i^T}{\partial u_i^T} = \varepsilon^T$$

$$\frac{\partial L}{\partial u_i^{T-1}} = \frac{\partial L}{\partial s_i^{T-1}} \cdot \frac{\partial s_i^{T-1}}{\partial u_i^{T-1}} + \overset{\text{iteration}}{\boxed{\frac{\partial L}{\partial u_i^T} \cdot \frac{\partial u_i^T}{\partial u_i^{T-1}}}}$$

$$= \varepsilon^{T-1} + \underbrace{\varepsilon^T \cdot \frac{\partial u_i^T}{\partial u_i^{T-1}}}_{\text{error from } t=T}$$

$$\frac{\partial L}{\partial u_i^{T-2}} = \frac{\partial L}{\partial s_i^{T-2}} \cdot \frac{\partial s_i^{T-2}}{\partial u_i^{T-2}} + \overset{\text{iteration}}{\boxed{\frac{\partial L}{\partial u_i^{T-1}} \cdot \frac{\partial u_i^{T-1}}{\partial u_i^{T-2}}}}$$

$$= \varepsilon^{T-2} + \underbrace{\varepsilon^{T-1} \cdot \frac{\partial u_i^{T-1}}{\partial u_i^{T-2}}}_{\text{error from } t=T-1} + \underbrace{\varepsilon^T \cdot \frac{\partial u_i^T}{\partial u_i^{T-1}} \cdot \frac{\partial u_i^{T-1}}{\partial u_i^{T-2}}}_{\text{error from } t=T}$$



ε^T
u_i^T
$\partial s_i^T / \partial u_i^T$
s_i^T
$\partial L / \partial s_i^T$
$\partial u_i^T / \partial u_i^{T-1}$
ε^{T-1}
u_i^{T-1}
$\partial s_i^{T-1} / \partial u_i^{T-1}$
s_i^{T-1}
$\partial L / \partial s_i^{T-1}$
L
$\partial u_i^{T-1} / \partial u_i^{T-2}$
$\partial L / \partial s_i^{T-2}$
u_i^{T-2}
$\partial s_i^{T-2} / \partial u_i^{T-2}$
s_i^{T-2}
ε^{T-1}

FIGURE 3 | Iterative calculation with linear algorithm complexity. At time T, the potential error comes from the direct error ε^T. At time $T-1$, the potential error includes the direct error ε^{T-1} and the backpropagation of ε^T. At time $T-2$, the influence of ε^T, ε^{T-1}, ε^{T-2} are taken into account through iterative calculation, which only requires one addition and multiplication.

w_{ki}). The above equation is still an iterative calculation with time complexity of $O(t)$. The calculation of the gradient of each parameter is still consistent with Equation (8)–(9). As a supplement, $\partial L/\partial w_{ik}$ can be calculated by the following equation, where i and k both belong to layer $\mathbb{L}^n$ and the initial value $s_k^0 = 0$.

$$\begin{aligned}\frac{\partial L}{\partial w_{ik}} &= \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^t} \cdot \frac{\partial u_i^t}{\partial w_{ik}} + \lambda_w \cdot \text{sign}(w_{ik}) \\ &= \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^t} \cdot s_k^{t-1} + \lambda_w \cdot \text{sign}(w_{ik}) \qquad (13)\end{aligned}$$

In this way, the required gradients are obtained. Errors can be passed down layer by layer. Each network parameter can be updated by various general ANN parameter optimization algorithms, such as stochastic gradient descent (SGD), adaptive momentum estimation (Adam) (Kingma and Ba, 2014) or Adam with decoupled weight decay (AdamW) (Loshchilov and Hutter, 2017).

2.5. Post-training Quantization

Fixed-point quantification can compress the storage overhead of SNN, and achieve higher computational efficiency by replacing floating-point arithmetic with fixed-point arithmetic. We use PTQ to quantify parameters, avoiding the overhead of re-learning. After learning, PTQ quantizes w and b into n-bit fixed-point numbers, where the fraction length is n - 1 and the signedness is 1-bit. This allows synaptic operations to be performed through fixed-point addition instead of floating-point addition. τ is rounded to 2^{-m}, so that the multiplication on the potential is replaced by m-bit right shift operation. PTQ brings optimization of storage overhead and energy consumption under the condition of limited accuracy loss.

3. REWIRING BASED ON WEIGHT AND GRADIENT

Rewiring mechanism prunes and grows synapses based on synaptic weights and gradients to improve synaptic sparsity. Synaptic weights are constantly decreasing in learning through synaptic regularization. When the $|w|$ is less than the pruning threshold Θ_w (Equation 14), it means that the influence on the post-synaptic neuron is negligible and synapse can be pruned (**Figure 4A**). Moreover, the pruned synapses have a chance to reconnect through growth. The gradient of the synaptic weight represents a trend of growth. The momentum m is the exponential moving averaging of the synaptic gradient ∇_w, where β_m is the coefficient of moving average. The m measures the strength of the growth trend after smoothing fluctuations. When the m is large enough to satisfy Equation (15), the synapse grows as shown in **Figure 4B**. The growth conditions include a constant threshold Θ_m and a distance term scaled by the ratio μ_m, where $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ represent the spatial coordinates of two neurons. The above condition means that establishing a longer-range synaptic connection requires a stronger growth trend. Dynamic rewiring is coupled with the learning process, using pruning and growth to improve sparsity and ensure performance. SNN is finally stable between rewiring and parameter optimization and acquires a sparse and efficient network structure.

$$\text{pruning}: \ |w| < \Theta_w \qquad (14)$$

$$\begin{aligned}\text{growth}: \ |m| &> \Theta_m \cdot \left(1 + \mu_m \left\|\boldsymbol{c}_i - \boldsymbol{c}_j\right\|_2^{1/2}\right), \\ m &:= m + (1 - \beta_m)\nabla_w \qquad (15)\end{aligned}$$

The rewiring mechanism works together with backpropagation and parameter optimization. The pseudo-code (**Algorithm 1**) takes layer $\mathbb{L}^n$ as an example to illustrate how to implement

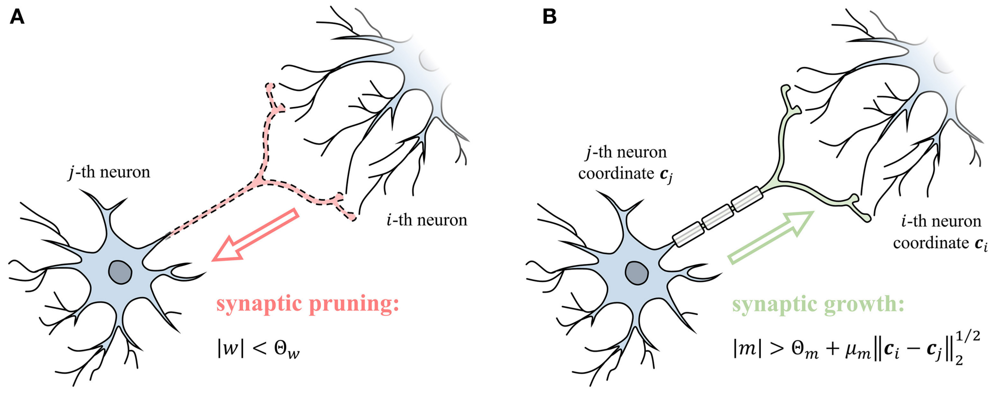

A
j-th neuron
i-th neuron
synaptic pruning:
$|w| < \Theta_w$
B
j-th neuron coordinate $\boldsymbol{c}_j$
i-th neuron coordinate $\boldsymbol{c}_i$
synaptic growth:
$|m| > \Theta_m + \mu_m \|\boldsymbol{c}_i - \boldsymbol{c}_j\|_2^{1/2}$

FIGURE 4 | (A) The weight of synapse *w* controls the synaptic pruning. **(B)** The momentum of the synapse gradient *m* controls the synaptic growth.

the proposed BPSR algorithm with matrix operation. The input spike matrix **X** and the gradient matrix of output spike $\boldsymbol{\Delta}_S$ are required. $\mathcal{N}^n$ represents the number of neurons in layer $\mathbb{L}^n$ and T is the time window. The shape of **X** is $\mathcal{N}^{n-1} \times T$. The gradient $\boldsymbol{\Delta}_S$ with shape of $\mathcal{N}^n \times T$ can be backpropagated by the following layer through Equation (6). The notation $[t]$ is used to represent the matrix slice in the time dimension. The algorithm generates the output spike **S** and the gradient of the input spike $\boldsymbol{\Delta}_X$, and ensures to update the synaptic weight matrix **W**, bias matrix **B** and, leakage coefficient matrix $\mathcal{T}$. For the flat layer, we mark the synaptic weight as $\mathbf{W} = \mathbf{W}_{ij}$. For the recurrent layer, the synaptic weight matrix is the concatenation $\mathbf{W} = [\mathbf{W}_{ij}|\mathbf{W}_{ik}]$. In the initial stage, the weight matrix **W** is set to obey Gaussian distribution $\mathcal{N}(0, 1)$. The bias matrix **B** is initialized to uniform distribution $\mathcal{U}(0, 1)$. The leakage coefficient matrix $\mathcal{T}$ is set to an empirical value of 0.5. The coordinates of neurons **C** are set to the random distribution in the unit cube. Especially, Kaiming initialization (He et al., 2015) is applied to the convolutional layer. The coordinates of neurons **C** are set to the random distribution in the unit cube. The forward and backpropagation processes are described in the previous sections. In the rewiring, Prun and Grow are two boolean matrices, denoting the synapses that meet the conditions 14 and 15. The boolean matrix Mask indicates the existing synapses after rewiring. Logical operations "and" and "or" achieve prune and grow, respectively. The **W** and $\boldsymbol{\Delta}_W$ is superimposed by Mask. Finally, all parameters are updated through the ANN optimization algorithm and clamped.

4. EXPERIMENTAL RESULTS

The proposed BPSR is implemented by PyTorch (Paszke et al., 2019) and runs on a CPU of AMD Ryzen-3970X and a GPU of NVIDIA RTX-3080. Various visual datasets and sensor datasets are used in the experiments. MNIST is a static digital dataset and can be transformed into a spiking dataset by rate coding and rank order coding. Rate coding (**Figure 5A**) takes pixel intensity as the probability and performs Bernoulli sampling in the time domain to produce spikes. Rank order coding (**Figure 5B**) convert higher values to earlier spikes, which is a kind of temporal sparse coding. Unlike rate coding, the spiking timing in rank order is meaningful. This requires the SNN to have the capacity for temporal processing. N-MNIST is a spiking version of MNIST and is acquired by the dynamic vision sensor (DVS). It is widely used in SNN research due to event-driven and neuromorphic. CIFAR10 is another static visual dataset for object classification of color images. We employ the encoding layer proposed by Wu et al. (2019) to convert floating values to spikes. MIT-BIH is an arrhythmia dataset that includes 48 sets of electrocardiographs (ECG). The level-crossing (LC) sampling (Marisa et al., 2017) converts signal into spike. 2-channel ECG generates 4-channel spiking input suitable for SNN, as shown in Section 4.1. The gas sensor dataset is the record from a chemical detection platform in a wind tunnel facility in response to ten high-priority chemical gaseous substances. The 72-channel sensing signal is encoded by rank order to obtain the spiking input.

4.1. Coding Method and Feature Visualization

A 5-class ECG task is used to show how SNN processes temporal information. The SNN model resented in **Figure 6A** is the recurrent MLP (rMLP) of "*r*18 - *fc*8 - *fc*5", where "*r*" denotes the recurrent layer and '*fc*' denotes the fully connected layer. **Figure 6B** demonstrates the original ECG signal and the spiking sequence after LC sampling. The 2 channels of the displayed record 102 are modified lead V2 and V5, and other records may contain modified limb lead II (MLII). Bipolar spikes are generated on the edge of signal changes in each channel. In this way, the spike reflects the changing trend of the signal. 4-channel spikes input to the recurrent layer for temporal feature processing. In the right of **Figure 6C**, the output FR curve of the recurrent layer under different input FR is plotted channel by channel. Neurons can be classified into low-pass, high-pass,

Algorithm 1: The BPSR implementation of layer $\mathbb{L}^n$.

```
Require: Input spike $\mathbf{X}$. The gradient of output spike $\mathbf{\Delta}_S$ obtained by backpropagation.
Ensure: Output spike $\mathbf{S}$. The gradient of input spike $\mathbf{\Delta}_X$. Update parameters $\mathbf{W}$, $\mathbf{B}$ and $\mathcal{T}$.

    Initialization:
 1: $\mathbf{W} \leftarrow \mathcal{N}(0,1)$, $\mathbf{B} \leftarrow \mathcal{U}(0,1)$, $\mathcal{T} \leftarrow 0.5$, $U_{th} \leftarrow 1$, $\mathbf{C} \leftarrow \mathcal{U}(0,1)$   // Initialize if applicable.
    Forward:
 2: for $t = 1$ to $T$ do
 3:   $\mathbf{U}[t] \leftarrow \mathrm{CalU}(\mathbf{U}[t-1], \mathbf{S}[t-1], \mathbf{X}[t], \mathbf{W}, \mathbf{B}, \mathcal{T})$   // Calculate potential by Equation (1). Specially $\mathbf{S}[0] = 0$.
 4:   $\mathbf{S}[t] \leftarrow \mathrm{CalS}(\mathbf{U}[t])$   // Calculate spike by Equation (2).
 5: end for
    Backpropagation:
 6: // Calculate gradient of potential by Equation (7) and (12).
 7: $\mathbf{\Delta}_U[T] \leftarrow \mathrm{CalG_U}(\mathbf{\Delta}_S[t], \mathbf{U}[t])$
 8: for $t = T-1$ to 1 do
 9:   $\mathbf{\Delta}_U[t] \leftarrow \mathrm{CalG_U}(\mathbf{\Delta}_S[t], \mathbf{\Delta}_U[t+1], \mathbf{U}[t], \mathbf{S}[t], \mathcal{T})$
10: end for
11: $\mathbf{\Delta}_X \leftarrow \mathrm{CalG_X}(\mathbf{\Delta}_U, \mathbf{W})$   // Calculate gradient of input spike by Equation (8).
12: $\mathbf{\Delta}_W \leftarrow \mathrm{CalG_W}(\mathbf{\Delta}_U, \mathbf{S}, \mathbf{X}, \mathbf{W})$, $\mathbf{\Delta}_B \leftarrow \mathrm{CalG_B}(\mathbf{\Delta}_U)$, $\mathbf{\Delta}_{\mathcal{T}} \leftarrow \mathrm{CalG_{\mathcal{T}}}(\mathbf{\Delta}_U, \mathbf{U}, \mathbf{S})$   // Calculate gradient of parameters by Equations (10)–(9) and (13).
    Rewiring:
13: $\mathrm{M} \leftarrow \mathrm{CalM}(\mathbf{\Delta}_W, \mathrm{M})$   // Calculate gradient momentum by Equation (15).
14: $\underline{\mathrm{Prun}} \leftarrow \mathrm{CalPrun}(\mathrm{W})$, $\underline{\mathrm{Grow}} \leftarrow \mathrm{CalGrow}(\mathrm{M}, \mathbf{Coor})$   // Pruning and growth by Equations (14)–(15).
15: $\mathrm{Mask} = (\mathbf{W}! = 0)$ and $\overline{\mathrm{Prun}}$ or $\mathrm{Grow}$   // Calculate mask of synapse by logical operation.
16: $\mathrm{W} := \mathrm{Mask} \cdot \mathrm{W}$, $\mathbf{\Delta}_W := \mathrm{Mask} \cdot \mathbf{\Delta}_W$   // Mask the weight and gradient.
    Updating:
17: Update $\mathbf{W}$, $\mathbf{B}$ and $\mathcal{T}$ with optimization algorithm such SGD, Adam or AdamW.
18: $\mathbf{W} \in [-U_{th}, +U_{th}]$, $\mathbf{B} \in [-U_{th}, +U_{th}]$, $\mathcal{T} \in [0,1]$   // Clamp parameters.
```

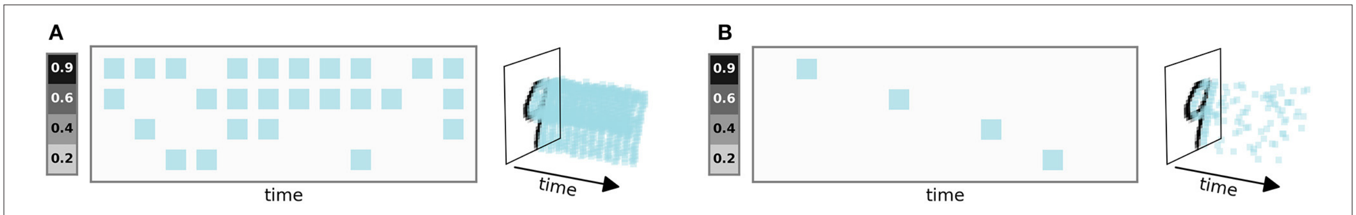


FIGURE 5 | The principle of **(A)** rate coding and **(B)** rank order coding, and the spike sequence of an MNIST image after coding.

band-pass and composite characteristics according to different filter effects. The left of **Figure 6C** shows the spike output of the recurrent layer and its influence on the prediction result. All neurons have a positive effect (green) on the prediction results, except for neuron 11 marked by the black box. In addition, neurons 0, 10, and 15 make more contributions, revealing that the corresponding frequency features are more important for predicting this class. **Figure 6D** is the spike output of the hidden layer. The role of this layer is the feature mapping before prediction. All neurons also make a positive effect except for one neuron. The final prediction result (**Figure 6E**) is the spike sum of 5 output neurons and is normalized to probability. It can be seen that the SNN makes the correct prediction for a normal heartbeat.

4.2. Algorithm Efficiency

The runtime and memory overhead reflect the efficiency of the algorithm, the accuracy and convergence epoch number prove its effectiveness. The proposed BPSR is compared with the other three SNN gradient descent algorithms, namely DECOLLE, STBP, and graph-based STBP (G-STBP) (Yan et al., 2021a). The four algorithms are all implemented based on PyTorch and accelerated by the GPU to get a fair comparison. The MNIST is encoded by rate coding as the time window T and the learning batch size is set to 32. The SNN model is a three-layer multilayer perceptron (MLP), where the size of the input layer is 784 and the output layer is 10. The number of neurons in the hidden layer ($\mathcal{N}^1$) is a variable in the experiment. **Figure 7** shows the algorithm runtime of a single epoch, the graphic

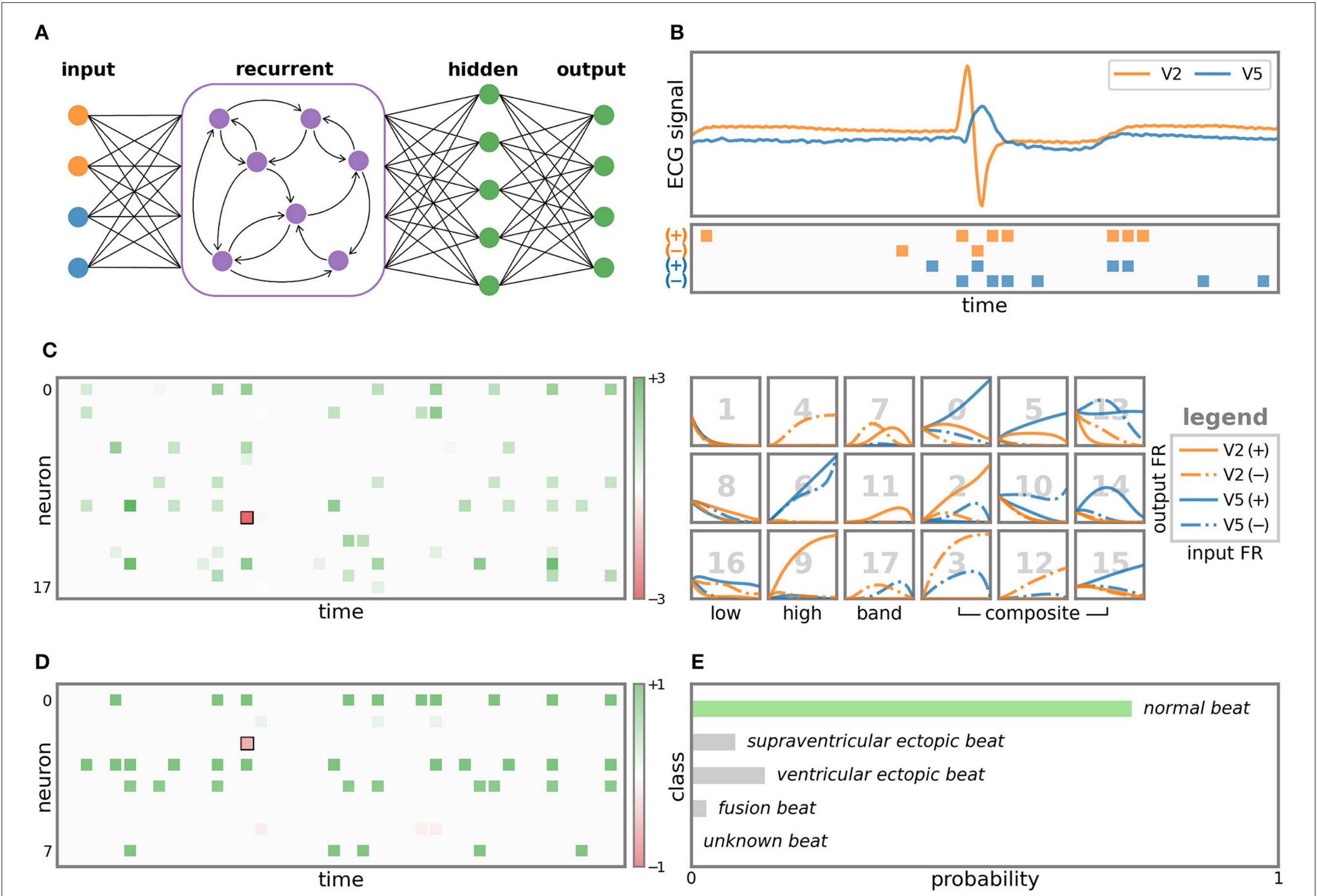


FIGURE 6 | Visualization of LC sampling and each layer of SNN. **(A)** The structure of the SNN model. **(B)** The ECG signal and the input spikes after LC sampling. **(C)** Output spikes and corresponding FR response curves of 18 neurons in the recurrent layer. The coordinates of the spikes represent the occurrence time and the neuron index. The color indicates the impact on the prediction result, where green is positive and red is negative. Response curves are plotted channel-by-channel. The x-axis and y-axis are the input and output FR, respectively. The 18 neurons are classified according to the filter effect, and the corresponding neuron index is marked in gray number. The output spikes of the hidden layer are drawn on **(D)**, and the predicted probability for the 5 ECG classes is shown on **(E)**.

memory overhead on the GPU, the accuracy with rate coding in different situations, and the number of epochs required for SNN learning to reach convergence. It can be seen that G-STBP has the smallest runtime in any case, also accompanied by the highest memory overhead. G-STBP describes the network as a whole adjacency graph. This allows backpropagation to be carried out on the entire network together instead of layer by layer, but the inter-layer connection is expressed as zero resulting in memory overhead. BPSR simplifies the storage and calculation burden of intermediate quantities through iterative calculations, bringing faster runtime (2.1× than STBP) and smaller memory overhead. BPSR also achieves the highest accuracy in all cases, with the second most convergence epoch, verifying its effectiveness.

4.3. Spiking Sparsity and Synaptic Sparsity

The effect of spiking sparsity regularization is tested on the MNIST dataset encoded by rank order. The used SNN model is rMLP of "*r*1000 - *fc*100 - *fc*10." The accuracy and average FR of the test set are counted under different spiking regularization coefficients λ_s. The count of FR excludes the input spike because it is controlled by the encoding method rather than the regularization. It can be seen from **Figure 8A** that FR decreases as the spiking regularization coefficient λ_s increases. Spiking regularization forces SNN to express information with fewer spikes. Through appropriate λ_s, the SNN can achieve high accuracy with low computation overhead in the inference. Moreover, the accuracy is improved with the decrease of FR when $\lambda_s \in [0, 10^{-7}]$. One reason is that SNN learning is a process of FR reduction. As shown in **Figure 8B**, the accuracy and FR are approximately inversely related during the learning process. SNN learns important features by suppressing redundant information. Setting a high initial threshold ($U_{th} = 10$) causes the FR to increase first and then become an inversely proportional learning process. Inappropriately high threshold ($U_{th} = 12$) can even lead to network divergence. The learning curve in **Figure 8C** verifies that spiking regularization can prevent overfitting. Under the same training error, the SNN with spike regularization achieves improved test accuracy and shows better generalization.

The effect of synaptic sparsity regularization is tested on the gas sensor dataset and the learned SNN structure is compared

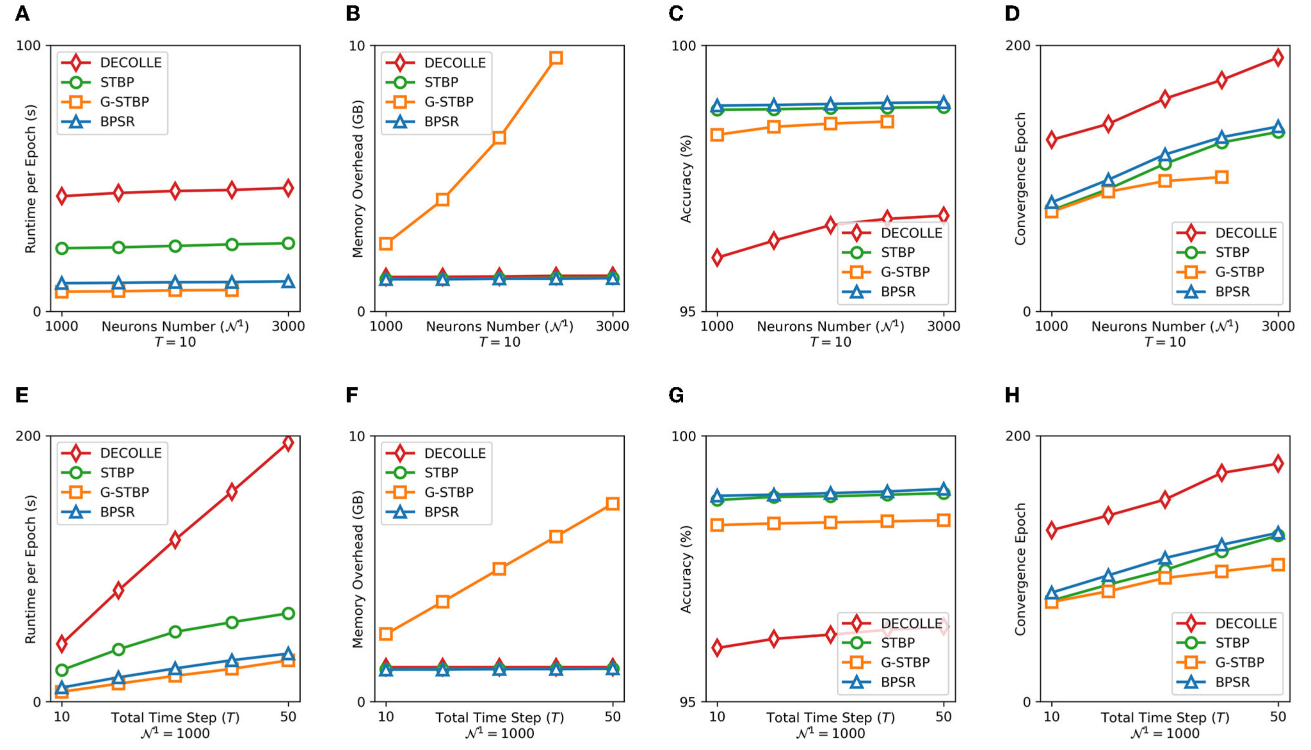


FIGURE 7 | **(A)** Runtime, **(B)** graphic memory overhead, **(C)** accuracy, and **(D)** convergence epoch of four learning algorithms are counted under the different number of hidden layer neurons $\mathcal{N}^1$. Panels **(E–H)** are the corresponding indicator under the different length of time window T.

with the nervous system of *Caenorhabditis elegans* (*C. elegans*). The neuron connection graph of *C. elegans* has been fully studied (Cook et al., 2019). The hermaphrodite and the male have 302 neurons and 385 neurons, respectively. 83 sensory neurons and 81 interneurons are the same for all genders. The tested SNN model is "*r*81 - *fc*36 - *fc*10." The input layer and the first hidden layer have a similar number of neurons as the *C. elegans*, which is convenient for structural comparison. SNN learns under synaptic regularization coefficient $\lambda_w = 0.01$. The line in **Figure 8D** shows the number of synapses in the input layer and the first hidden layer. The point cloud plots the network structure (topological connection) during the rewiring process. After 117 epochs, the network can be 8× in the recurrent layer. In **Figure 8E**, the above network obtained by rewiring is re-initialized to evaluate the convergence speed. SNN with the same number of synapses but a random structure is also tested. Experiment shows that the SNN without rewiring will reach the lowest error 4 epochs earlier than the SNN with rewiring. SNN with random structures has higher errors, demonstrating the effect of rewiring.

The efficacy of rewiring is further verified by significance profile (SP) (Milo et al., 2004), a method of analyzing the similarity of network structure. It measures the structural characteristics of the network by comparing the number of occurrences of different induced subgraphs (i.e., motifs) in the network. The possible connection modes between the three nodes are used as 13 motifs. A set of random networks is generated as the reference based on the degree sequence of the network to be tested. The numbers of occurrences of 13 motifs in the network to be tested and the random network set are recorded as the 13-dimensional vector $\mathbf{N}_{test}$ and vector set $\mathbf{N}_{rand}$, respectively. The SP is the vector normalization of $(\mathbf{N}_{test} - \overline{\mathbf{N}}_{rand})/\mathrm{std}(\mathbf{N}_{rand})$. The SP of hermaphrodite (herm) and male *C. elegans*, and the SP of SNN before and after learning are plotted in **Figure 8F**. It can be seen that the hermaphrodite and the male *C. elegans* have the same structural characteristics. After BPSR learning, the structure of SNN is more similar to the nervous system of *C. elegans*, which means that the rewiring mechanism can generate an effective and bionic network structure.

4.4. Evaluation of Performance

Table 2 provides the network structure and hyper-parameters used in the various experiments below. Convolutional indicator "8*c*5/2" means kernel size 5, output channel 8 and stride 2. "*r*" and "*fc*" denote the recurrent layer and the fully connected layer, respectively. [·] means a residual block (He et al., 2016). For convolutional neurons, τ is homogeneous and shared while learning. For neurons in other layers, τ is heterogeneous.

4.4.1. MNIST Dataset

Table 3 shows the comparison results of the proposed BPSR and related SNN works on the MNIST dataset. The pooling is taken into account of the number of synapses, and shared weight in the convolution is repeatedly added. The introduction of recurrent layers enhances accuracy but brings additional

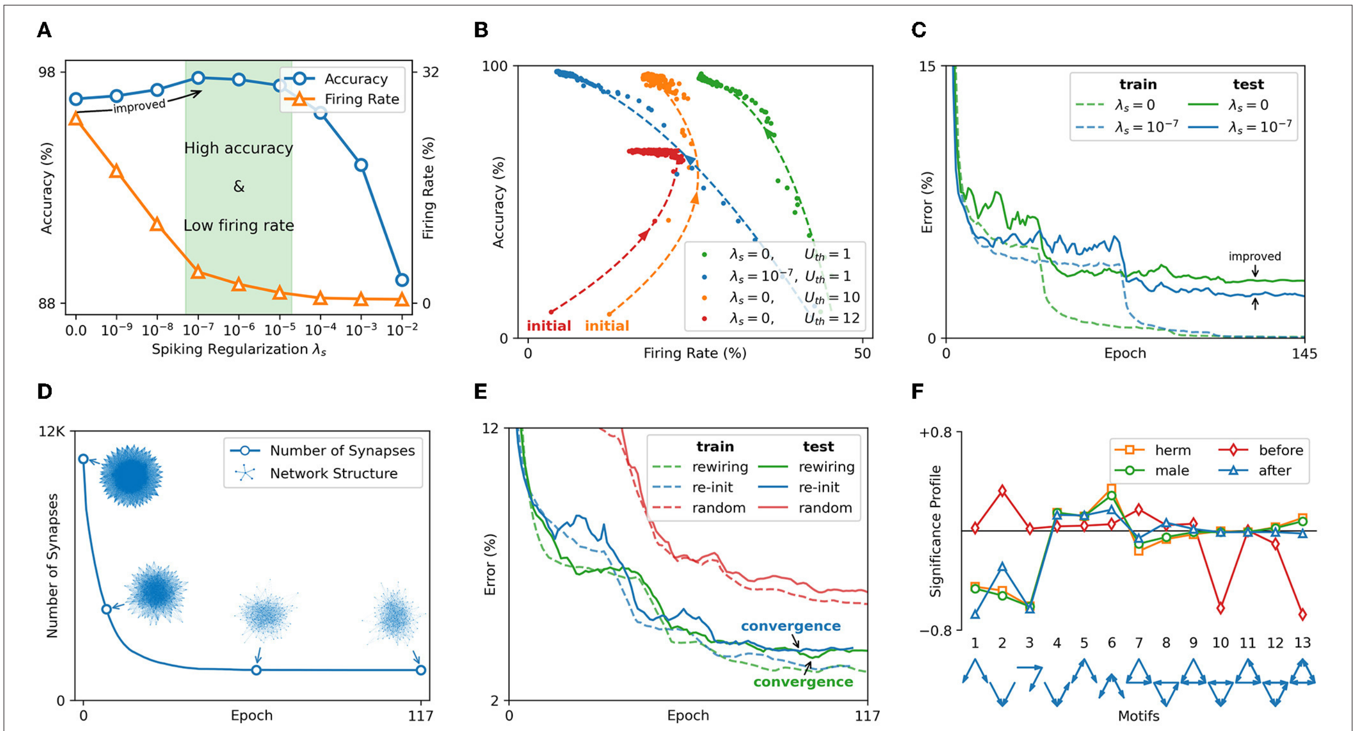


FIGURE 8 | (A) The test accuracy and FR under different spiking regularization coefficients λ_s. **(B)** Accuracy and FR change law in the learning process. **(C)** Learning curves under different λ_s (after smoothing filtering). **(D)** The number of synapses and network structure changes in the recurrent layer. **(E)** The learning curve with and without rewiring mechanism (after smoothing filtering). **(F)** Significance profile of *C. elegans* nervous system and the gas sensor network.

TABLE 2 | SNN structures and hyper-parameters setup.

Structure	
MNIST	8c5/2 - 16c3/2 - *r*100 - *fc*10
N-MNIST	4c5/2 - 16c3/2 - 32c3 - *r*100 - *fc*10
CIFAR10	$64c7/2 - \begin{bmatrix}128c3\\128c3\end{bmatrix} - \begin{bmatrix}256c3/2\\256c3\end{bmatrix} - \begin{bmatrix}512c3/2\\512c3\end{bmatrix} - \begin{bmatrix}1024c3/2\\1024c3\end{bmatrix} - fc1024 - fc10$
MIT-BIH	*r*256 - *fc*96 - *fc*18 (18 classes) *r*192 - *fc*64 - *fc*5 (5 classes)
Gas sensor	*r*128 - *fc*64 - *fc*10
Hyper-parameter	
Potential threshold	$U_{th} = 1$
Leakage coefficient	$\tau = 0.5$ (initial). *Homogeneous* for conv, otherwise *heterogeneous*.
Coefficient of $g(\cdot)$	$\alpha = 0.7$
Learning rate	CIFAR10: $lr = 10^{-3}$, otherwise: 10^{-2}
Sparsity coefficient	CIFAR10: $\lambda_s = 10^{-9}/10^{-8}$, otherwise: 10^{-7}; $\lambda_w = 10^{-2}$
Rewiring parameter	$\Theta_w = 10^{-2}$; $\Theta_m = 10^{-4}$; $\mu_m = 5$; $\beta_m = 0.99$

overhead, which is further improved by sparsity regularization. Compared to other sparse networks using pruning, the proposed BPSR acquired the least number of synapses, with the best spiking sparsity except for G-STBP. Floating-point operations (FLOPs) show the computational overhead of SNN in the learning and inference process. Conversion-based algorithm (Diehl et al., 2015) learns parameters through ANN, avoiding the backpropagation in the time window. It has the lowest learning FLOPs and high accuracy (the conversion cost is underlined and only occurs once after learning). Plasticity-based algorithm is generally considered to be efficient due to local learning rules. However, Diehl and Cook (2015) used a large network to improve the accuracy, resulting in the learning burden. Gradient-based algorithms have high backpropagation overhead but also

TABLE 3 | Comparison of different spiking models on MNIST dataset.

	Coding	Pruning	Model	Synapses	Spikes	FLOPs/sample		Accuracy (%)
						learning	inference	
Diehl et al. (2015)	Rate	×	MLP	2.4M	10.0K*	24.0+7.2M*	6.3M*	98.6
			CNN	1.4M	14.7K*	7.5+2.9M*	2.0M*	99.1
Diehl and Cook (2015)	Rate	×	rMLP	46.0M	2.3K	74.7M	15.0M	95.0
Wu et al. (2018)	Rate	×	MLP	0.6M	6.7K*	78.9M*	2.6M*	98.89
			CNN	1.4M	41.4K*	162.3M*	5.1M*	99.42
Yan et al. (2021a)	Rank	×	rMLP	0.3M	392	17.3M	84.5K	97.3
Tang et al. (2020)	Rank	×	CNN	0.6M		——— N/A** ———		90.2
Comşa et al. (2021)	Rank	×	MLP	0.3M		——— N/A** ———		97.96
Shi et al. (2019)	Rate	√	MLP	0.2M		——— N/A** ———		94.05
Guo et al. (2020)	Rate	√	rMLP	0.5M		——— N/A** ———		88.71
Liang et al. (2021)	Rank	√	MLP	0.4M		——— N/A** ———		96
BPSR (this work)	Rank	×	CNN	**98K**	**859**	**10.1M**	**86.8K**	**97.56**
		×	rCNN	**0.1M**	**2.6K**		**0.19M**	**98.43**
		√	rCNN	**73K**	**542**		**67.6K**	**98.33**

*The result is estimated based on the open source code.
**Data is not available (N/A) due to the lack of experimental result and source code. The bold values mark our metrics for this work.

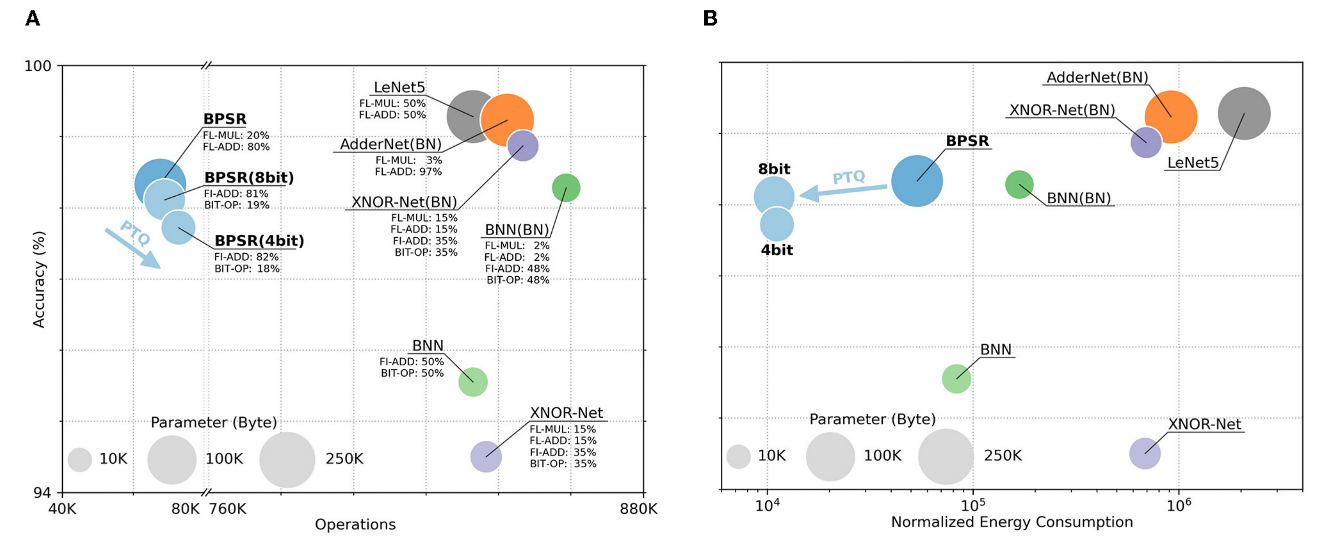


FIGURE 9 | Accuracy, operations, normalized energy consumption, and parameter size of different networks. The area of the circle represents the storage overhead of the parameters. The y-coordinate of the center represents the network accuracy. The x-coordinate represents **(A)** the number of operations and **(B)** the energy consumption of each inference. The proportions of different operations are marked in **(A)**. Additionally, the x-axis of (a) is folded and the x-axis of **(B)** is logarithmic.

bring performance optimization. Wu et al. (2018) and Yan et al. (2021a) have improved the SNN with the goal of better accuracy and sparser spikes, respectively. The proposed BPSR achieves a low learning overhead due to its extremely sparse network. Moreover, rank order coded data has a higher learning difficulty due to sparse temporal representation. The accuracy of BPSR is only 0.8–1.1% lower than rate coding, with a 30× inference overhead advantage.

Networks such as BNN and AdderNet improve energy efficiency by reducing computational overhead, which is similar to SNN. We also compare the performance of the proposed BPSR and other ANN in **Figure 9**. The network structure used is LeNet5 and their variant. As mentioned in the original work, batch normalization (BN) (Ioffe and Szegedy, 2015) is introduced to improve accuracy. The involved operations include floating-point multiplication (FL-MUL), floating-point addition (FL-ADD), fixed-point addition (FI-ADD), and bitwise operation (BIT-OP). The network energy consumption in inference is counted by normalization. FL-ADD is considered as unit overhead. FL-MUL is estimated to be 4× of floating-point

TABLE 4 | Comparison of different spiking models on N-MNIST dataset.

	Model	T	Synapses	Accuracy (%)
Wu et al. (2018)	MLP	30	1.9M	98.78
Jin et al. (2018)	MLP	N/A*	1.9M	98.93
Wu et al. (2019)	CNN	30	202.4M	99.53
Vaila et al. (2019)	Mixed CNN + SVM	N/A*	0.98M	98.32
Kaiser et al. (2020)	CNN	300	315.5M	99.04
BPSR (this work)	CNN	20	**0.26M**	**99.15**
	rCNN		**0.26M**	**99.21**

Data is not available (N/A) due to the lack of result reports. The bold values mark our metrics for this work.

TABLE 5 | Comparison of different spiking models on CIFAR10 dataset.

	Model	T	Synapses	Spikes	Accuracy (%)
Cao et al. (2015)	5-layer CNN	400	5.7M	N/A*	77.43
Wu et al. (2018)	4-layer CNN	N/A*	2.9M	N/A*	50.7
Wu et al. (2019)	8-layer CNN	12	519.8M	N/A*	90.53
Sengupta et al. (2019)	VGG16	2500	315.5M	N/A*	91.55
Allred et al. (2020)	LeNet5	N/A*	0.66M	89.9K	66.45
BPSR (this work)	11-layer ResNet	12	**260.7M**	**136.1K** ($\lambda_s = 10^{-9}$)	**90.74**
		8		**89.6K** ($\lambda_s = 10^{-8}$)	**90.24**

Data is not available (N/A) due to the lack of result reports. The bold values mark our metrics for this work.

addition (Cheng et al., 2019), and FI-ADD is estimated to be 20% of FL-ADD (Finnerty and Ratigner, 2017). The overhead of BIT-OP is negligible.

Under the same structure, AdderNet reduces the computational cost by approximating multiplication by addition. BNN and XNOR-Net further reduce storage burden and energy overhead through bitwise operations. The proposed BPSR achieves optimized energy consumption through lightweight structure and sparse spike while ensuring accuracy. PTQ quantizes the parameters of SNN to 8-bit or 4-bit, and further uses fixed-point addition and bitwise right shift instead of floating-point addition and floating-point multiplication to reduce the energy cost. After PTQ, the proposed BPSR reaches $15 \sim 60\times$ energy efficiency than BNN or XNOR-Net, with a 0.22–0.61% accuracy drop of unquantized SNN.

4.4.2. N-MNIST Dataset

Table 4 shows the comparison of N-MNIST. Event-driven N-MNIST is usually converted to frame-based data. Time step T matches the time length of the frame sequence. Most networks take few time steps, except Kaiser et al. (2020) which uses 60 steps to warm up the network and 240 steps to learn and infer. Kaiser et al. (2020) uses a shallow network, but the readout layer followed by each regular layer greatly increases the synaptic overhead. The network used by Vaila et al. (2019) is a mixture of ANN and SNN, and the prediction results are given by SVM. Wu et al. (2019) uses the deepest network and most synapses to get the best accuracy. BPSR has minimal synaptic overhead and achieves the second-best accuracy. Introducing a recurrent layer improves the accuracy in the case where the number of tiny synapses grows, proving that the recurrent structure is useful for frame sequence processing.

4.4.3. CIFAR10 Dataset

We applied the residual SNN on CIFAR10 to verify the performance of the BPSR on the deep model. **Table 5** compares BPSR with other SNN works. Sengupta et al. (2019) achieves the best accuracy on VGG16 with a conversion-based learning algorithm. However, the conversion takes 2500 time steps to rate encoding, much higher than other methods. Wu et al. (2019) uses a gradient-based learning algorithm to achieve high accuracy while keeping small time steps. Although in the work of Allred et al. (2020), the accuracy of SNN is limited by the network size, the sparsity resulting from regularization is further explored. We test BPSR on an 11-layer residual network composed of 4 residual blocks. The number of synapses is less than that of other deep networks. The proposed BPSR can reach 90.24% accuracy with the same number of spikes as Allred et al. (2020), or achieve the accuracy of 90.74% with 50% additional spike overhead.

TABLE 6 | Comparison of different spiking models on MIT-BIH dataset.

	Model	T	Synapses	Accuracy (%)
Kolağasioğlu (2018)	wavelet + rMLP	N/A*	N/A*	95.5 (17 classes)
Corradi et al. (2019)	rMLP + SVM	250	25.6K	95.6 (18 classes)
Amirshahi and Hashemi (2019)	rMLP	300	968.0K	97.9 (4 classes)
Bauer et al. (2019)	rMLP	N/A*	34.8K	97.3 (2 classes)
Wu et al. (2020)	GRU + MLP	N/A*	20.8K	97.8 (5 classes)
Yan et al. (2021b)	CNN	180	184.3K	90 (4 classes)
BPSR (this work)	rMLP	40	**15.3K**	**97.82 (18 classes)**
			10.4K	**98.41 (5 classes)**

Data is not available (N/A) due to the lack of result reports.

4.4.4. MIT-BIH Dataset

Table 6 are the comparison results between BPSR and related spiking models on the MIT-BIH dataset. Most of the work introduces recurrent structures such as lateral inhibition to process temporal signals. In addition, Kolağasioğlu (2018) use wavelet transform for signal preprocessing, Wu et al. (2020) adopt the gated recurrent unit (GRU), and Corradi et al. (2019) use the support vector machine (SVM) for prediction. These make the implementation no longer pure SNN. MIT-BIH dataset contains various ECG arrhythmia types with a long-tailed distribution. The classification of the fewer sample has a higher learning difficulty. Most works achieve 2-5 classification tasks by selecting subsets and merging certain classes. Kolağasioğlu (2018) and Corradi et al. (2019) take 17 or 18 classes for fine-grained classification. Thus, we used the two models 18 classes and 5 classes. BPSR can make inferences from the compressed time window ($T = 40$), which is more efficient. The proposed

TABLE 7 | Comparison of different models on gas senor dataset.

	Model	T	Synapses	Accuracy (%)
Vergara et al. (2013)	SVM	–	–	87.14–96.55
Imam and Cleland (2020)	EPL	16	55.4K	92
BPSR (this work)	rMLP	16	**7.7K**	**98.30**

– The indicator is not applicable.

BPSR achieves the highest accuracy in fine-grained classification and coarse-grained classification. With the proposed sparsity regularization, the learned models under different classification tasks both achieve optimal synaptic sparsity.

4.4.5. Gas Sensor Dataset

Table 7 shows the comparison results of BPSR and related works on the gas sensor dataset. Vergara et al. (2013) use the SVM method to obtain high accuracy. Imam and Cleland (2020) implement the spiking method on Loihi through the external plexiform layer (EPL) structure. Although this method does not perform well in network accuracy, the reported results show high robustness and biological inspiration. BPSR achieves better accuracy and synaptic overhead than related works. At the same time, the proposed SNN with sparsity regularization only needs 762 spikes per sample to achieve the inference.

5. DISCUSSION

SNN promises to realize efficient AI through its brain-inspired mechanism and spike-driven computing architecture. However, the efficiency advantage of the SNN cannot be fully exploited because of the lack of sparsity exploration. This work provides a learning algorithm, namely Backpropagation with Sparsity Regularization (BPSR), to improve efficiency through advanced spiking sparsity and synaptic sparsity. Firstly, a backpropagation algorithm with sparsity regularization is proposed to update parameters and improve sparsity. A heterogeneous LIF neuron dynamics model and a classification loss function with spiking and synaptic regularization are defined. The backpropagation algorithm of the flat and recurrent layer is detailed to calculate the gradient of each parameter. Secondly, the rewiring mechanism based on weight and gradient is proposed to improve synaptic sparsity through pruning and growth. Then, the experimental results show that the proposed BPSR has the advantages of runtime and graphic memory overhead compared with other gradient-based learning algorithms. The improved spiking sparsity can balance the accuracy and FR, and promotes the network performance by simplifying the information representation. Through the BPSR, SNN acquires a structure similar to the nervous system of *C. elegans*, proving its effectiveness. The proposed BPSR reaches the accuracy of 98.33% on the MNIST dataset while achieving 30× inference overhead than other SNN work and 15× energy efficiency compared to BNN after PTQ (with 0.22% accuracy drop). Finally, BPSR is also evaluated on two visual datasets (N-MNIST and CIFAR10) and two sensor datasets (MIT-BIH and gas sensor). The experimental results show comparable or superior accuracy (99.21, 90.74, 98.41, and 98.30%, respectively), with spiking sparsity and synaptic sparsity.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding authors.

AUTHOR CONTRIBUTIONS

YY proposed the idea and did the math and engineering work. YY, HC, and YJ designed the experiments and wrote the first draft of the manuscript. YH, ZZ, and LZ directed the projects and provided overall guidance. ZZ and LZ provided the supervision and project administration. All authors contributed to manuscript revision, read, and approved the submitted version.

FUNDING

This work was supported in part by the National Natural Science Foundation of China under Grants 61876039, 62076066, 62004045, and 92164301, Shanghai Municipal Science and Technology Major Projects Nos. 2021SHZDZX0103, 2018SHZDZX01, 17DZ2260900, and NSFC-STINT project No. 62011530132.

REFERENCES

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Allred, J. M., Spencer, S. J., Srinivasan, G., and Roy, K. (2020). Explicitly trained spiking sparsity in spiking neural networks with backpropagation. *arXiv [Preprint]*. arXiv:2003.01250. Available online at: https://arxiv.org/pdf/2003.01250.pdf (accessed March 2, 2020).

Amirshahi, A., and Hashemi, M. (2019). ECG classification algorithm based on STDP and R-STDP neural networks for real-time monitoring on ultra low-power personal wearable devices. *IEEE Trans. Biomed. Circ. Syst.* 13, 1483–1493. doi: 10.1109/TBCAS.2019.2948920

Bartol, T. M. Jr., Bromer, C., Kinney, J., Chirillo, M. A., Bourne, J. N., Harris, K. M., et al. (2015). Nanoconnectomic upper bound on the variability of synaptic plasticity. *eLife*, 4:e10778. doi: 10.7554/eLife.10778

Bauer, F. C., Muir, D. R., and Indiveri, G. (2019). Real-time ultra-low power ECG anomaly detection using an event-driven neuromorphic processor. *IEEE Trans. Biomed. Circ. Syst.* 13, 1575–1582. doi: 10.1109/TBCAS.2019.2953001

Białas, M., and Mańdziuk, J. (2021). Spike-timing-dependent plasticity with activation-dependent scaling for receptive fields development. *IEEE Trans. Neural Netw. Learn. Syst.* 1–14. doi: 10.1109/TNNLS.2021.3069683

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis*. 113, 54–66. doi: 10.1007/s11263-014-0788-3

Chen, H., Wang, Y., Xu, C., Shi, B., Xu, C., Tian, Q., et al. (2020). "AdderNet: do we really need multiplications in deep learning?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Virtual), 1468–1477. doi: 10.1109/CVPR42600.2020.00154

Cheng, Z., Wang, W., Pan, Y., and Lukasiewicz, T. (2019). Distributed low precision training without mixed precision. *arXiv preprint arXiv:1911.07384*. Available online at: https://arxiv.org/pdf/1911.07384

Cho, S.-G., Beigné, E., and Zhang, Z. (2019). "A 2048-neuron spiking neural network accelerator with neuro-inspired pruning and asynchronous network on chip in 40nm CMOS," in *2019 IEEE Custom Integrated Circuits Conference (CICC)* (Austin, TX: IEEE), 1–4. doi: 10.1109/CICC.2019.8780116

Comşa, I.-M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., and Alakuijala, J. (2021). Temporal coding in spiking neural networks with alpha synaptic function: learning with backpropagation. *IEEE Trans. Neural Netw. Learn. Syst*.

Cook, S. J., Jarrell, T. A., Brittin, C. A., Wang, Y., Bloniarz, A. E., Yakovlev, M. A., et al. (2019). Whole-animal connectomes of both *Caenorhabditis elegans* sexes. *Nature* 571, 63–71. doi: 10.1038/s41586-019-1352-7

Corradi, F., Pande, S., Stuijt, J., Qiao, N., Schaafsma, S., Indiveri, G., et al. (2019). "ECG-based heartbeat classification in neuromorphic hardware," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest: IEEE), 1–8. doi: 10.1109/IJCNN.2019.8852279

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dempsey, W. P., Du, Z., Nadtochiy, A., Smith, C. D., Czajkowski, K., Andreev, A., et al. (2022). Regional synapse gain and loss accompany memory formation in larval zebrafish. *Proc. Natl. Acad. Sci. U.S.A*. 119, e2107661119. doi: 10.1073/pnas.2107661119

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci*. 9, 99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280696

Ding, C., Huan, Y., Jia, H., Yan, Y., Yang, F., Zou, Z., et al. (2021). "An ultra-low latency multicast router for large-scale multi-chip neuromorphic processing," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Washington, DC: IEEE), 1–4. doi: 10.1109/AICAS51828.2021.9458445

Finnerty, A., and Ratigner, H. (2017). *Reduce Power and Cost by Converting From Floating Point to Fixed Point*. Available online at: https://japan.xilinx.com/support/documentation/white_papers/wp491-floating-to-fixed-point.pdf

Frenkel, C., Bol, D., and Indiveri, G. (2021). Bottom-up and top-down neural processing systems design: neuromorphic intelligence as the convergence of natural and artificial intelligence. *arXiv preprint arXiv:2106.01288*. Available online at: https://arxiv.org/pdf/2106.01288

Guo, W., Yantır, H. E., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2020). Towards efficient neuromorphic hardware: unsupervised adaptive neuron pruning. *Electronics* 9, 1059. doi: 10.3390/electronics9071059

He, K., Zhang, X., Ren, S., and Sun, J. (2015). "Delving deep into rectifiers: surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision* (Santiago), 1026–1034. doi: 10.1109/ICCV.2015.123

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV), 770–778. doi: 10.1109/CVPR.2016.90

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). "Binarized neural networks," in *Advances in Neural Information Processing Systems, Vol. 29*, eds D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Barcelona: MIT Press).

Imam, N., and Cleland, T. A. (2020). Rapid online learning and robust recall in a neuromorphic olfactory circuit. *Nat. Mach. Intell*. 2, 181–191. doi: 10.1038/s42256-020-0159-4

Ioffe, S., and Szegedy, C. (2015). "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning* (Lille: PMLR), 448–456.

Jin, Y., Zhang, W., and Li, P. (2018). Hybrid macro/micro level backpropagation for training deep spiking neural networks. *Adv. Neural Inf. Process. Syst*. 31, 1–11. Available online at: https://proceedings.neurips.cc/paper/2018/file/3fb04953d95a94367bb133f862402bce-Paper.pdf

Kaiser, J., Mostafa, H., and Neftci, E. (2020). Synaptic plasticity dynamics for deep continuous local learning (DECOLLE). *Front. Neurosci*. 14, 424. doi: 10.3389/fnins.2020.00424

Kim, S., Park, S., Na, B., and Yoon, S. (2020). "Spiking-YOLO: spiking neural network for energy-efficient object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence* (New York, NY), 11270–11277. doi: 10.1609/aaai.v34i07.6787

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. Available online at: https://arxiv.org/pdf/1412.6980

Kolağasioğlu, E. (2018). *Energy efficient feature extraction for single-lead ECG classification based on spiking neural networks* (Master thesis). Delft University of Technology, Delft, Netherlands.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images*. Available online at: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Liang, M., Zhang, J., and Chen, H. (2021). "A 1.13 μJ/classification spiking neural network accelerator with a single-spike neuron model and sparse weights," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)* (Daegu: IEEE), 1–5. doi: 10.1109/ISCAS51556.2021.9401607

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. (2020). Backpropagation and the brain. *Nat. Rev. Neurosci*. 21, 335–346. doi: 10.1038/s41583-020-0277-3

Loshchilov, I., and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*. Available online at: https://arxiv.org/pdf/1711.05101

Luo, L. (2021). Architectures of neuronal circuits. *Science* 373, eabg7285. doi: 10.1126/science.abg7285

Marisa, T., Niederhauser, T., Haeberlin, A., Wildhaber, R. A., Vogel, R., Goette, J., et al. (2017). Pseudo asynchronous level crossing ADC for ECG signal acquisition. *IEEE Trans. Biomed. Circ. Syst*. 11, 267–278. doi: 10.1109/TBCAS.2016.2619858

Milo, R., Itzkovitz, S., Kashtan, N., Levitt, R., Shen-Orr, S., Ayzenshtat, I., et al. (2004). Superfamilies of evolved and designed networks. *Science* 303, 1538–1542. doi: 10.1126/science.1089167

Moody, G. B., and Mark, R. G. (2001). The impact of the MIT-BIH arrhythmia database. *IEEE Eng. Med. Biol. Mag*. 20, 45–50. doi: 10.1109/51.932724

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., and Ganjtabesh, M. (2018). First-spike-based visual categorization using reward-modulated STDP. *IEEE Trans. Neural Netw. Learn. Syst*. 29, 6178–6190. doi: 10.1109/TNNLS.2018.2826721

Nguyen, T. N. N., Veeravalli, B., and Fong, X. (2021). Connection pruning for deep spiking neural networks with on-chip learning. (Knoxville, TN). *arXiv [Preprint]*. arXiv: 2010.04351. Available online at: https://arxiv.org/pdf/2010.04351.pdf (accessed July 31, 2021).

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci*. 9:437. doi: 10.3389/fnins.2015.00437

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inform. Process. Syst*. 32, 8026–8037.

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). "XNOR-Net: imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision* (Amsterdam: Springer), 525–542. doi: 10.1007/978-3-319-46493-0_32

Rathi, N., Panda, P., and Roy, K. (2018). STDP-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst*. 38, 668–677. doi: 10.1109/TCAD.2018.2819366

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Rueckauer, B., and Liu, S.-C. (2018). "Conversion of analog to spiking neural networks using sparse temporal coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (Florence: IEEE), 1–5. doi: 10.1109/ISCAS.2018.8351295

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci*. 13, 95. doi: 10.3389/fnins.2019.00095

Shi, Y., Nguyen, L., Oh, S., Liu, X., and Kuzum, D. (2019). A soft-pruning method applied during training of spiking neural networks for in-memory computing applications. *Front. Neurosci*. 13, 405. doi: 10.3389/fnins.2019.00405

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature* 550, 354–359. doi: 10.1038/nature24270

Stöckl, C., and Maass, W. (2021). Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nat. Mach. Intell*. 3, 230–238. doi: 10.1038/s42256-021-00311-4

Tang, H., Cho, D., Lew, D., Kim, T., and Park, J. (2020). Rank order coding based spiking convolutional neural network architecture with energy-efficient membrane voltage updates. *Neurocomputing* 407, 300–312. doi: 10.1016/j.neucom.2020.05.031

Tang, P. T. P., Lin, T.-H., and Davies, M. (2017). Sparse coding by spiking neural networks: convergence theory and computational results. *arXiv preprint arXiv:1705.05475*. Available online at: https://arxiv.org/pdf/1705.05475

Thorpe, S., and Gautrais, J. (1998). "Rank order coding," in *Computational Neuroscience*, ed J. M. Bower (Boston, MA: Springer), 113–118. doi: 10.1007/978-1-4615-4831-7_19

Vaila, R., Chiasson, J., and Saxena, V. (2019). "Feature extraction using spiking convolutional neural networks," in *Proceedings of the International Conference on Neuromorphic Systems* (Knoxville, TN), 1–8. doi: 10.1145/3354265.3354279

Vergara, A., Fonollosa, J., Mahiques, J., Trincavelli, M., Rulkov, N., and Huerta, R. (2013). On the performance of gas sensor arrays in open sampling systems using Inhibitory Support Vector Machines. *Sens. Actuat. B Chem*. 185, 462–477. doi: 10.1016/j.snb.2013.05.027

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci*. 12, 331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). "Direct training for spiking neural networks: faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence* (Honolulu, HI), 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Wu, Y., Liu, Y., Liu, S., Yu, Q., Chen, T., and Liu, Y. (2020). Spike-driven gated recurrent neural network processor for electrocardiogram arrhythmias detection realised in 55-nm CMOS technology. *Electron. Lett*. 56, 1230–1232. doi: 10.1049/el.2020.2224

Yan, Y., Chu, H., Chen, X., Jin, Y., Huan, Y., Zheng, L., et al. (2021a). "Graph-based spatio-temporal backpropagation for training spiking neural networks," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Washington, DC: IEEE), 1–4. doi: 10.1109/AICAS51828.2021.9458461

Yan, Z., Zhou, J., and Wong, W.-F. (2021b). Energy efficient ECG classification with spiking neural network. *Biomed. Signal Process. Control* 63, 102170. doi: 10.1016/j.bspc.2020.102170

Zhang, W., and Li, P. (2019). Spike-train level backpropagation for training deep recurrent spiking neural networks. *Adv. Neural Inf. Process. Syst*. 32, 1–12. Available online at: https://web.ece.ucsb.edu/~lip/publications/ST-RSBP-NeurIPS2019.pdf

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Yan, Chu, Jin, Huan, Zou and Zheng. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

ORIGINAL RESEARCH
published: 28 April 2022
doi: 10.3389/fnins.2022.855753

Modeling the Repetition-Based Recovering of Acoustic and Visual Sources With Dendritic Neurons

Giorgia Dellaferrera[1,2*†], *Toshitake Asabuki*[1] *and Tomoki Fukai*[1]

[1] Neural Coding and Brain Computing Unit, Okinawa Institute of Science and Technology, Okinawa, Japan, [2] Institute of Neuroinformatics, University of Zurich and Swiss Federal Institute of Technology Zurich (ETH), Zurich, Switzerland

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Lyes Khacef,
University of Groningen, Netherlands
Dylan Richard Muir,
University of Basel, Switzerland
Tom Tetzlaff,
Helmholtz Association of German Research Centres (HZ), Germany

****Correspondence:***
Giorgia Dellaferrera
gde@zurich.ibm.com

†Present address:
Giorgia Dellaferrera,
Department of Ophthalmology, Children's Hospital, Harvard Medical School, Boston, MA, United States

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *15 January 2022*
Accepted: *31 March 2022*
Published: *28 April 2022*

Citation:
Dellaferrera G, Asabuki T and Fukai T (2022) Modeling the Repetition-Based Recovering of Acoustic and Visual Sources With Dendritic Neurons. Front. Neurosci. 16:855753. doi: 10.3389/fnins.2022.855753

In natural auditory environments, acoustic signals originate from the temporal superimposition of different sound sources. The problem of inferring individual sources from ambiguous mixtures of sounds is known as blind source decomposition. Experiments on humans have demonstrated that the auditory system can identify sound sources as repeating patterns embedded in the acoustic input. Source repetition produces temporal regularities that can be detected and used for segregation. Specifically, listeners can identify sounds occurring more than once across different mixtures, but not sounds heard only in a single mixture. However, whether such a behavior can be computationally modeled has not yet been explored. Here, we propose a biologically inspired computational model to perform blind source separation on sequences of mixtures of acoustic stimuli. Our method relies on a somatodendritic neuron model trained with a Hebbian-like learning rule which was originally conceived to detect spatio-temporal patterns recurring in synaptic inputs. We show that the segregation capabilities of our model are reminiscent of the features of human performance in a variety of experimental settings involving synthesized sounds with naturalistic properties. Furthermore, we extend the study to investigate the properties of segregation on task settings not yet explored with human subjects, namely natural sounds and images. Overall, our work suggests that somatodendritic neuron models offer a promising neuro-inspired learning strategy to account for the characteristics of the brain segregation capabilities as well as to make predictions on yet untested experimental settings.

Keywords: dendritic neurons, spiking neural networks, blind source separation, sound source repetition, spatio-temporal structure

1. INTRODUCTION

Hearing a sound of specific interest in a noisy environment is a fundamental ability of the brain that is necessary for auditory scene analysis. To achieve this, the brain has to unambiguously separate the target auditory signal from other distractor signals. In this vein, a famous example is the "cocktail party effect" (Cherry, 1953), i.e., the ability to distinguish a particular speaker's voice against a multi-talker background (Brown et al., 2001; Mesgarani and Chang, 2012). Many psychophysical and neurobiological studies have been conducted to clarify the psychophysical properties and underlying mechanisms of the segregation of mixed signals (Asari et al., 2006; Bee and Micheyl, 2008; Narayan et al., 2008; McDermott, 2009; McDermott et al., 2011; Schmidt and Römer, 2011; Lewald and Getzmann, 2015; Li et al., 2017; Atilgan et al., 2018), and computational theories and models have also been proposed for this computation (Amari et al., 1995; Bell and Sejnowski, 1995; Sagi et al., 2001; Haykin and Chen, 2005; Elhilali and Shamma, 2009; Thakur et al., 2015; Dong et al., 2016; Kameoka et al., 2018; Karamatli et al., 2018; Sawada et al., 2019). However, how the brain attains its remarkable sound segregation remains elusive. Various properties of auditory cues such as spatial cues in binaural listening (Ding and Simon, 2012) and temporal coherence of sound stimuli (Teki et al., 2013; Krishnan et al., 2014) are known to facilitate the listener's ability to segregate a particular sound from the background. Auditory signals that reached to ears first undergo the analysis of frequency spectrums by cochlea (Oxenham, 2018). Simultaneous initiation and termination of the component signals and the harmonic structure of the frequency spectrums help the brain to identify the components of the target sound (Popham et al., 2018). Prior knowledge about the target sound, such as its familiarity to listeners (Elhilali, 2013; Woods and McDermott, 2018), and top-down attention can also improve their ability to detect the sound (Kerlin et al., 2010; Xiang et al., 2010; Ahveninen et al., 2011; Golumbic et al., 2013; O'Sullivan et al., 2014; Bronkhorst, 2015). Selective attention as the combination of the auditory (sound) and visual (lip movements, visual cues) modalities has also been suggested to be beneficial to solve the cocktail party problem (Yu, 2020; Liu et al., 2021). However, many of these cues are subsidiary and not absolutely required for hearing the target sound. For example, a mixture sound can be separated by monaural hearing (Hawley et al., 2004) or without spatial cues (Middlebrooks and Waters, 2020). Therefore, the crucial mechanisms of sound segregation remain to be explored.

Whether or not biological auditory systems segregate a sound based on principles similar to those invented for artificial systems remains unclear (Bee and Micheyl, 2008; McDermott, 2009). Among such principles, independent component analysis (ICA) (Comon, 1994) and its variants are the conventional mathematical tools used for solving the sound segregation problem, or more generally, the blind source decomposition problem (Amari et al., 1995; Bell and Sejnowski, 1995; Hyvärinen and Oja, 1997; Haykin and Chen, 2005). Owing to its linear algebraic features, the conventional ICA requires as many input channels (e.g., microphones) as the number of signal sources, which does not appear to be a requirement for sound segregation in biological systems. In this context, however, recent works for single-channel source separation based on techniques such as Non-Negative Matrix Factorization (NNMF) have demonstrated that ICA can be applied with a lower number of channels than the number of sources (Krause-Solberg and Iske, 2015; Mika et al., 2020). In addition, NNMF has been shown to extract regular spatio-temporal patterns within the audio and to achieve good performance in applications such as music processing (Smaragdis and Brown, 2003; Cichocki et al., 2006; Santosh and Bharathi, 2017; López-Serrano et al., 2019). It has been suggested as an alternative possibility that human listeners detect latent recurring patterns in the spectro-temporal structure of sound mixtures for separating individual sound sources (McDermott et al., 2011). This was indicated by the finding that listeners could identify a target sound when the sound was repeated in different mixtures in combination with various other sounds but could not do so when the sound was presented in a single mixture.

The finding represents an important piece of information about the computational principles of sound source separation in biological systems. Here, we demonstrate that a computational model implementing a pattern-detection mechanism accounts for the characteristic features of human performance observed in various task settings. To this end, we constructed a simplified model of biological auditory systems by using a two-compartment neuron model recently proposed for learning regularly or irregularly repeated patterns in input spike trains (Asabuki and Fukai, 2020). Importantly, this learning occurs in an unsupervised fashion based on the minimization principle of regularized information loss, showing that the essential computation of sound source segregation can emerge at the single-neuron level without teaching signals. Furthermore, it was previously suggested that a similar repetition-based learning mechanism may also work for the segregation of visual objects (McDermott et al., 2011). To provide a firm computational ground, we extended the tasks of our framework to predictions on visual images.

2. RESULTS

2.1. Learning of Repeated Input Patterns by a Two-Compartment Neuron Model

We used a two-compartment spiking neuron model which learns recurring temporal features in synaptic input, as proposed in Asabuki and Fukai (2020). In short, the dendritic compartment attempts to predict the responses of the soma to given synaptic input by modeling the somatic responses. To this end, the neuron model minimizes information loss within a recent period when the somatic activity is replaced with its model generated by the dendrite. Mathematically, the learning rule minimizes the Kullback–Leibler (KL) divergence between the probability distributions of somatic and dendritic activities. The dendritic membrane potential of a two-compartment neuron obeys $v(t) = \sum_j w_j e_j(t)$, where w_j and e_j stand for the synaptic weight and the unit postsynaptic potential of the j-th presynaptic input, respectively. The somatic activity evolves as

$$\dot{u}(t) = -\frac{1}{\tau}u(t) + g_D[-u(t) + v(t)] - \sum_j G_k \phi^{som}(u_k(t))/\phi_0, \quad (1)$$

where the last term describes lateral inhibition with modifiable synaptic weights G_k (≥ 0), as shown later. The soma generates a Poisson spike train with the instantaneous firing rate $\phi^{som}(u(t))$, where ${\phi_i}^{som}(u_i) = \phi_0[1 + e^{\beta(-u_i+\theta)}]^{-1}$, and the parameters β and θ are modified in an activity-dependent manner in terms of the mean and variance of the membrane potential over a sufficiently long period t_0. To extract the repeated patterns from temporal input, the model compresses the high dimensional data carried by the input sequence onto a low dimensional manifold of neural dynamics. This is performed by modifying the weights of dendritic synapses to minimize the time-averaged mismatch between the somatic and dendritic activities over a certain interval [0,T]. In a stationary state, the somatic membrane potential $u_i(t)$ can be described as an attenuated version $v_i^*(t)$ of the dendritic membrane potential. At each time point, we compare the attenuated dendritic membrane potential with the somatic membrane potential, on the level of the two Poissonian spike distributions with rates $\phi_i^{som}(u(t))$ and $\phi(v_i^*(t))$, respectively, which would be generated if both soma and dendrite were able to emit spikes independently. In practice, the neuron model minimizes the following cost function for synaptic weights w, which represents the averaged KL-divergence between somatic activity and dendritic activity, and in which we explicitly represent the dependency of u_i and v_i^* on X:

$$E(\mathbf{w}) = \int_{\Omega_X} dXP^*(\mathbf{X})$$
$$\int_0^T dt \sum_i D_{KL}[\phi_i^{som}(u_i(t;\mathbf{X}))||\phi^{dend}(v_i^*(t;\mathbf{X}))], \qquad (2)$$

with $P^*(\mathbf{X})$ and Ω_X being the true distribution of input spike trains and the entire space spanned by them, and $\phi^{dend}(x) = \phi_0[1 + e^{\beta_0(-x+\theta_0))}]^{-1}$. To search for the optimal weight matrix, the cost function $E(w)$ is minimized through gradient descent: $\Delta w_{ij} \propto -\partial E/\partial w_{ij}$. Introducing the regularization term $-\gamma \mathbf{w}_i$ and a noise component ξ_i with its intensity g gives the following learning rule (for the derivation see Asabuki and Fukai, 2020):

$$\dot{\mathbf{w}}_i(t) = \eta\{\psi(v_i^*(t))[\{f(\phi_i^{som}+\phi_0 g\xi_i)-\phi^{dend}(v_i^*(t))\}/\phi_0]\mathbf{e}(t)-\gamma \mathbf{w}_i\}, \qquad (3)$$

where $\mathbf{w}_i = [w_{i1,...,w_{iN_{in}}}]$, $\mathbf{e}(t) = [e_1, ...e_{Nin}]$, ξ_i obeys a normal distribution, $\psi(x) = \frac{d}{dx}log(\phi^{dend}(x))$, ϕ^{som} and ϕ^{dend} follow Poisson distributions, η is the learning rate, and

$$f(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \le x < \phi_0, \\ \phi_0 & \text{if } x \ge \phi_0 \end{cases}$$

Finally, if a pair of presynaptic and postsynaptic spikes occur at the times t_{pre} and t_{post}, respectively, lateral inhibitory connections between two-compartment neurons i and j are modified through a symmetric anti-Hebbian STDP as

$$\Delta G_{ij} = C_p exp\left(-\frac{t_{pre}-t_{post}}{\tau_p}\right) - C_d exp\left(-\frac{t_{pre}-t_{post}}{\tau_d}\right) \qquad (4)$$

See Section 4 and **Supplementary Note** for additional details. The prediction is learnable when input spike sequences from presynaptic neurons are non-random and contain recurring temporal patterns. In such a case, the minimization of information loss induces a consistency check between the dendrite and soma, eventually enforcing both compartments to respond selectively to one of the patterns. Mathematically, the somatic response serves as a teaching signal to supervise synaptic learning in the dendrite. Biologically, backpropagating action potentials may provide the supervising signal (Larkum et al., 1999; Larkum, 2013).

We constructed an artificial neural network based on the somatodendritic consistency check model and trained the network to perform the task of source recovering from embedded repetition. The network consisted of two layers of neurons. The input layer encoded the spectrogram of acoustic stimuli into spike trains of Poisson neurons. For each sound, the spike train was generated through a sequence of 400 time steps, where each time step corresponds to a "fire" or "non-fire" event. The output layer was a competitive network of the two-compartment models that received synaptic input from the input layer and learned recurring patterns in the input (**Figure 1**). We designed the output layer and the learning process similarly to the network used previously (Asabuki and Fukai, 2020) for the blind signal separation (BSS) within mixtures of multiple mutually correlated signals. In particular, lateral inhibitory connections between the output neurons underwent spike-timing-dependent plasticity for self-organizing an array of feature-selective output neurons (Section 4). In the spike encoding stage, the spectrogram is flattened into a one-dimensional array where the intensity of each element is proportional to the Poisson firing probability of the associated input neuron. This operation disconnects the signal's temporal features from the temporal dynamics of the neurons. Although this signal manipulation is not biologically plausible and introduces additional latency as the whole sample needs to be buffered, it allows the input layer to encode simultaneously all the time points of the audio signal. Thanks to this strategy, the length of the input spike trains does not depend on the duration of the audio signal, and a sufficiently large population of input neurons can encode arbitrarily long sounds, possibly with some redundancy in the encoding for short sounds. We remark that, while the somatodendritic mismatch learning rule was conceived to capture temporal information in an online fashion, in our framework it is applied to a flattened spectrogram, thus to a static pattern. Furthermore, in order to relate the signal intensity with the encoding firing rate, we normalized the spectrogram values to the interval [0,1]. This strategy is suited to our aim of reproducing the experiments with synthetic sounds and custom naturalistic stimuli. However, in a real-world application any instantaneous outlier in signal intensity would destroy other temporal features of an input signal. Nonetheless, the normalization is performed independently for each mixture, so if the outlier affects a masker sound and not a target, and the target is presented in at least two other mixtures, we expect that the normalization does not affect the ability of the network of identifying sounds presented in different mixtures.

2.2. Synthesized and Natural Auditory Stimuli

We examined whether the results of our computational model are consistent with the outcomes of the experiments on human listeners on artificially synthesized sounds described previously (McDermott et al., 2011). To provide a meaningful comparison with the human responses, we adopted for our simulations settings as close as possible to the experiments, both in terms of dataset generation and performance evaluation (Section 4). In McDermott et al. (2011), the generation of synthetic sounds is performed by first measuring the correlations between pairs of spectrograms cells of natural sounds (spoken words and

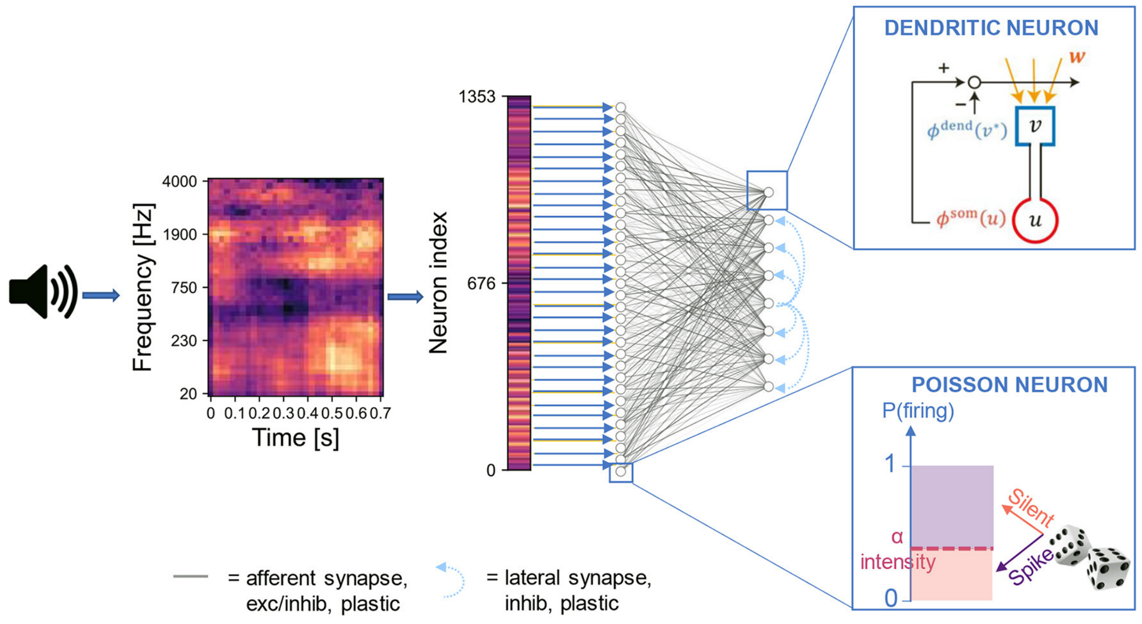


FIGURE 1 | Network architecture. The input signal is pre-processed into a two-dimensional image (i.e., the spectrogram) with values normalized in the range [0,1]. The image is flattened into a one-dimensional array where the intensity of each element is proportional to the Poisson firing probability of the associated input neuron. The neurons in the input layer are connected to those in the output layer through either full connectivity or random connectivity with connection probability $p = 0.3$. The output neurons are trained following the artificial dendritic neuron learning scheme (Asabuki and Fukai, 2020).

animal vocalizations). Then such correlations are averaged across different pairs to obtain temporal correlation functions. The correlation functions in turn are used to generate covariance matrices, in which each element is the covariance between two spectrogram cells. Finally, spectrograms are drawn from the resulting Gaussian distribution and applied to samples of white noise, leading to the synthesis of novel sounds. In our experiments we synthesized the sounds using the toolbox provided at https://mcdermottlab.mit.edu/downloads.html. In the human experiments, a dataset containing novel sounds was generated such that listeners' performance in sound source segregation was not influenced by familiarity with previously experienced sounds. To closely reproduce the experiment, we created a database of synthesized sounds according to the same method as described in McDermott et al. (2011) (Section 4). The synthesized stimuli retained similarity to real-world sounds except that they lacked grouping cues related to temporal onset and harmonic spectral structures. Furthermore, unlike human listeners, our neural network was trained and built from scratch, and had no previous knowledge of natural sounds that could bias the task execution. We exploited this advantage to investigate whether and how the sound segregation performance was affected by the presence of grouping cues in real sounds. To this goal we also built a database composed of natural sounds (Section 4).

To build the sequence of input stimuli, we randomly chose a set of sounds from the database of synthesized or natural sounds, and we generated various mixtures by superimposing them—i.e., we summed element-wise the spectrograms of the original sounds and then normalized the sum to the interval [0,1]. We refer to the main sound, which is always part of mixtures, as the *target*, and to all the other sounds, which were either presented as mixing sounds with the target (i.e., masker sounds) or presented alone, as *distractors*. The target sound is shown in red in the training protocols. Following the protocol in McDermott et al. (2011), we concatenated the mixtures of target and distractors into input sequences. For certain experiments, we also included unmixed distractor sounds. We presented the network with the input sequence for a fixed number of repetitions. As each input signal—both unmixed sounds and mixtures—is flattened into one input vector, each input signal is one element of the input sequence. During the input presentation, the network's parameters evolved following the learning rule described in Asabuki and Fukai (2020). Then, we examined the ability of the trained network to identify the target sound by using probe sounds, which were either the target or distractor sound composing the mixtures presented during training (*correct probe*) or a different sound (*incorrect probe*). Incorrect probes for synthesized target sounds were generated similarly as described in McDermott et al. (2011). Specifically, we synthesized the incorrect probe by using the same covariance structure of the target sound, and then we set a randomly selected time slice of the incorrect probe (1/8 of the sound's duration) to be equal to a time slice of the target of the same duration. Examples of target sounds, distractor sounds and incorrect probes are shown in **Figures 2A–C**, respectively. A further beneficial aspect of our model is the possibility of freezing plasticity during the inference stage, so that the synaptic

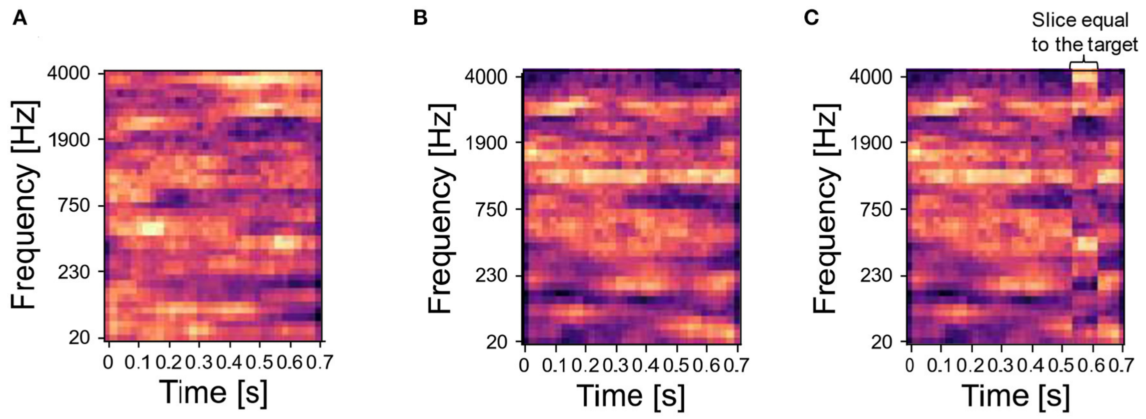


FIGURE 2 | Synthesized sounds—target and associated distractor. **(A)** Spectrogram of one target sound. **(B)** Step 1 to build the spectrogram of an incorrect probe related to the target in **(A)**: a sound is randomly selected from the same Gaussian distribution generating the target. **(C)** Step 2 to build the incorrect probe: after the sampling, a randomly selected time slice equal to 1/8 of the sound duration is set to be equal to the target. In the figure, the temporal slice is the vertical stripe around time 0.5 s.

connections do not change during the probe presentation. This allows us to investigate whether the trained network can identify not only the target but also the masker sounds.

2.3. Learning of Mixture Sounds in the Network Model

Our network model contained various hyperparameters such as number of output neurons, number of mixtures and connectivity pattern. A grid search was performed to find the best combination of hyperparameters. **Figures 3A,B** report the learning curves obtained on synthesized and natural sounds, respectively, for random initial weights and different combinations of hyperparameters. For both types of sounds, synaptic weights changed rapidly in the initial phase of learning. The changes were somewhat faster for synthesized sounds than for natural sounds, but the learning curves behaved similarly for both sound types. The number of output neurons little affected the learning curves, while they behaved differently for different connectivity patterns or different numbers of mixtures. Because familiarity to sounds enhances auditory perception in humans (Jacobsen et al., 2005), we investigated whether pretraining with a sequence containing target and distractors improves learning in our model for various lengths of pretraining. Neither the training speed nor the final accuracy were significantly improved by the pretraining (**Figures 3C–E**). This suggests that the model was "forgetting" about the pretraining stage and learning the mixture sounds from scratch, not exploiting any familiarity with previously seen sounds. We suspect that this behavior is related to the well know limitation of ANNs of lack of continual learning (French, 1999) rather than to a specific feature of our model. Furthermore, we cannot provide a comparison in the learning curve between the model and the psychophysical data, since the model was trained for multiple epochs, while the human listeners were presented with the training sequence only once and then tested on the probe immediately after.

To reliably compare the performance of our model with human listeners, we designed a similar assessment strategy to that adopted in the experiment. In McDermott et al. (2011), listeners were presented with mixtures of sounds followed by a probe which could be either a correct probe (i.e., the target sound present in the training mixtures) or an incorrect probe (i.e., sounds unseen during the training). The subjects had to say whether they believed the probe was present in the training mixture by using one of the four responses "sure no," "no," "yes," and "sure yes." The responses were used to build a receiver operating characteristics (ROC) as described in Wickens (2002), and the area under the curve (AUC) was used as performance measure, with AUC = 0.5 and 1 corresponding to chance and perfect correct, respectively. In our algorithm, we mimicked this protocol for reporting by using the likelihood as a measure of performance. To this goal, first, for each tested probe, we projected the response of the N output neurons (**Figures 4A,D**) to a two-dimensional PCA projection plane. We defined the PCA space based on the response to the correct probes and later projected on it the datapoints related to the incorrect probes (**Figures 4B,E**). We remark that other clustering approaches such as K-means and self-organizing maps could be used instead of PCA without reducing the output dimension. Second, we clustered the datapoints related to the correct probes through a Gaussian Mixture Model (GMM) with as many classes as the number of correct probes (**Figures 4C,F**). Third, for each datapoint we computed the likelihood that it belonged to one of the clusters. The target likelihood values are fixed to 1 and 0 for datapoints related to correct and incorrect probes respectively. We highlight that the labels introduced in this post-processing phase are not specific for each sound, but rather depend on the role of the sound in the tasks, i.e., if sound X is presented during training as a target or masker sound it is associated to label 1, while if, in another simulation, the same sound X is used to build an incorrect probe (not used during training) then it is associated with label 0. We binned the likelihood range into

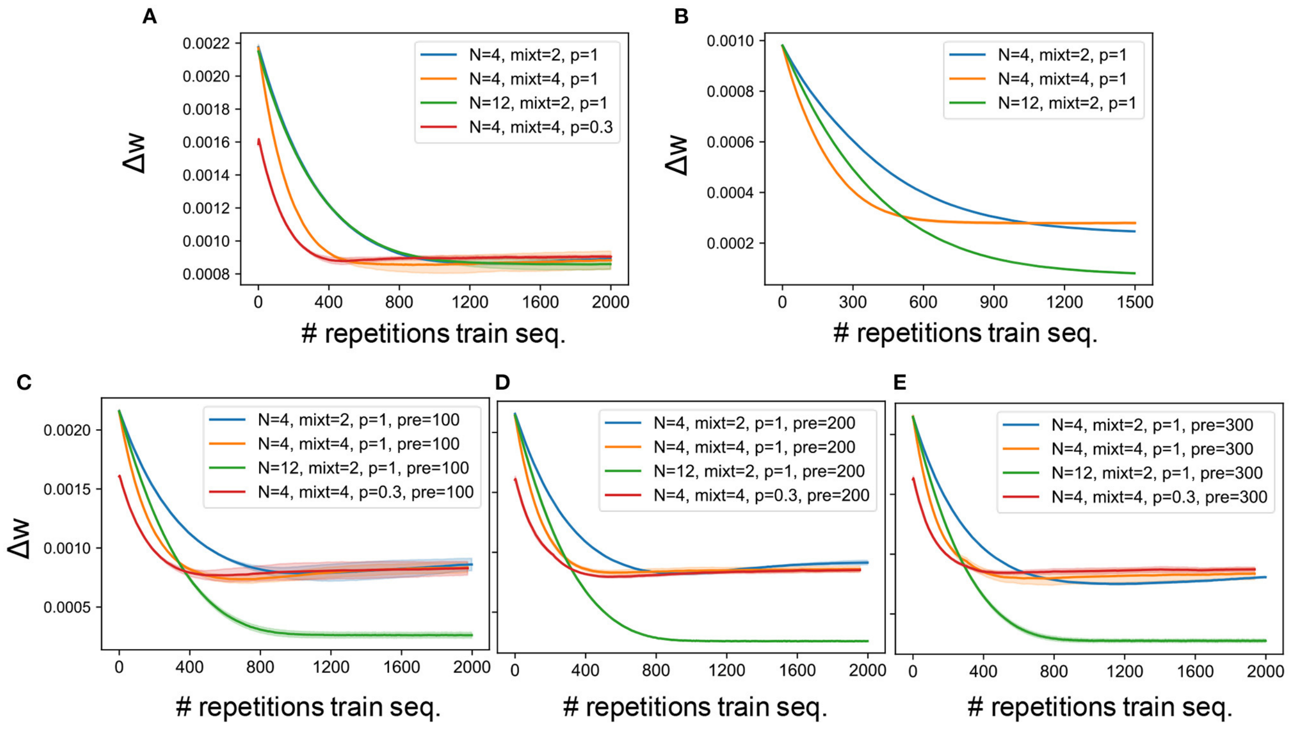


FIGURE 3 | Learning curves. **(A)** Average synaptic weight change for the experiments carried out on the synthetized sounds, the network being initialized with random values. **(B)** Average synaptic weight change for the experiments carried out on the natural sounds, the network being initialized with random values. **(C)** Average synaptic weight change for the experiments carried out on the synthetized sounds, the network being pretrained on the targets set presented for 100 epochs. **(D)** Average synaptic weight change for the experiments carried out on the synthetized sounds, the network being pretrained on the targets set presented for 200 epochs. **(E)** Average synaptic weight change for the experiments carried out on the synthetized sounds, the network being pretrained on the targets set presented for 300 epochs. The solid line and the shaded area represent the mean and standard deviation over 3 independent runs, respectively. Without pretraining, when the number of output neurons is varied no significant change is found, while with pretraining when a larger number of neurons is used, the weight change curve saturates at a lower value, as shown by the blue ($N = 4$) and green ($N = 12$) curves. Furthermore, the figures show that both when a larger number of training mixtures is presented (yellow curves) and when only 30% of the connections are kept (red curves) the slope of the learning curve is steeper. The weight change is computed by storing the weights values every 2,000 time steps (i.e., "fire" or "non-fire" events) and computing the standard deviation over the last 100 recorded values. The standard deviation is then averaged across all connections from input to output neurons. Therefore, each point on the curve reports the average weight change over the past 2000×100 time steps. Note that each sound/mixture is presented for 400 time steps. Finally, the x-axis shows the number of repetitions of the training mixture sequence (2,000 for synthetic sounds and 1,500 for naturalistic sounds).

four intervals corresponding, in an ascending order, to the four responses "sure no," "no," "yes," and "sure yes." Finally, based on the four responses, we built the receiver operating characteristic (ROC) curve: the datapoints falling in the interval (i) $L > 0$ (sure yes) were assigned the probability value $p = 1.0$, those in (ii) $-5 < L < 0$ (yes) $p = 0.66$, those in (iii) $-15 < L < -5$ (no) $p = 0.33$, and those in (iv) $L < -15$ (sure no) $p = 0.0$. The AUC of the ROC is used as the "accuracy" metric to evaluate the performance of the model. For additional details see Section 4. Now, we are ready to examine the performance of the model in a series of experiments. We show examples of the different behavior of the network trained on single (**Figures 4A–C**) or four mixtures (**Figures 4D–F**). As expected, the ability of the model to learn and distinguish the targets from the distractors depended crucially on the number of mixtures.

The algorithm was implemented in Python and a sample code used to simulate Experiment 1 is available at the repository https://github.com/GiorgiaD/dendritic-neuron-BSS.

2.4. Experiment 1: Sound Segregation With Single and Multiple Mixtures of Synthesized Sounds

To begin with, we compared how the number of mixtures influences the learning performance between human subjects and the model. The number of mixtures presented during training was varied from 1, where no learning was expected, to 2 or more, where the model was expected to distinguish the target sounds from their respective distractors. The simulation protocol is shown in **Figure 5A** (bottom). As reported in **Figure 5A** (top), we obtained that, when one mixture only was shown, neither the target nor the mixing sound was learnt, and performance was close to chance. An immediate boost in the performance was observed when the number of mixtures was raised to two. The network managed to distinguish the learnt targets from the incorrect probes with an accuracy greater than 90%. As the number of mixtures increased up to six, the accuracy worsened slightly, remaining above 80%.

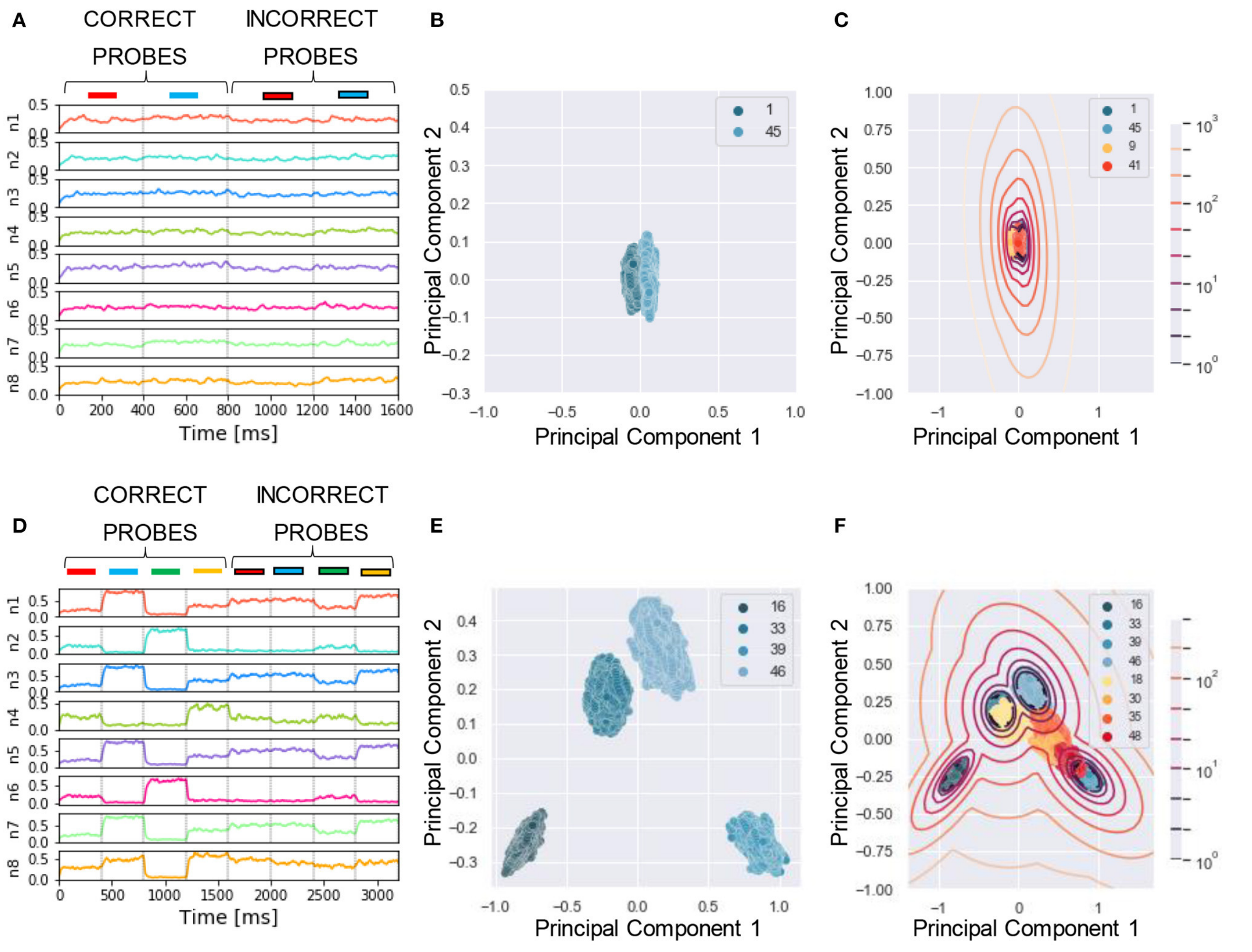


FIGURE 4 | Experiment 1—output dynamics and clustering. **(A–C)** refer to the results of Experiment 1 on synthesized sounds with a single mixture presented during training. **(D–F)** refer to the results of Experiment 1 on synthesized sounds with three mixtures presented during training. The "correct probes" are the target and the distractor sounds composing the mixtures presented during training, while the "incorrect probes" are sounds not presented during training. The numbers in the legends indicate the sound IDs. **(A)** Voltage dynamics of the 8 output neurons during inference, when the target, the distractor and the two associated incorrect probes are tested. The neuron population is not able to respond with different dynamics to the four sounds, and the voltage of all the output neurons fluctuates randomly throughout the whole testing sequence. **(B)** The PCA projection of the datapoints belonging to the two targets (in blue) shows that the clusters are collapsed into a single cluster. **(C)** When GMM is applied, all the datapoints representing both the correct probes (in blue) and the incorrect probes (in orange and red) fall within the same regions, making it impossible to distinguish the different sounds based on the population dynamics. **(D)** Voltage dynamics of the 8 output neurons during inference, when the four targets and the associated distractors are tested. As expected, the neuron population has learnt the feature of the different sounds and responds with different dynamics to the eight sounds. Each output neuron exhibits an enhanced response to one or few sounds. **(E)** The PCA projection of the datapoints belonging to the four correct probes (in blue) shows that the clusters are compact and spatially distant one from the other. **(F)** When GMM is applied, the model shows that the network is, most of the times, able to distinguish the target and distractors (in blue) from the incorrect probes (in yellow, orange and red). The correct probes are never overlapped. Three of the four distractors fall far from the targets' region, while the fourth (in yellow) overlaps with one of the targets. These results are overall coherent with the human performance. In **(C,F)**, the contour lines represent the landscape of the log-likelihood that a point belongs to one of the clusters associated to the correct probes.

A significant drop in the performance was observed for a greater number of mixtures. From a comparison with the results shown in **Figure 5B**, which were replicated for human subjects (McDermott et al., 2011), it emerged that our model was able to partially reproduce human performance: the success rate was at chance levels when training consists of a single mixture only; the target sounds could be distinguished to a certain accuracy if more than a mixture was learnt. We also verified that the model performance was robust for variations of the network architecture, both in terms of the number of output neurons N and the connection probability p (**Supplementary Figure 1**). Furthermore we observe that, while none of the output neurons exhibits an enhanced high firing rate when presented with the target sound, the overall population response to the target is substantially different from the response to the masker sounds and to the incorrect probes.

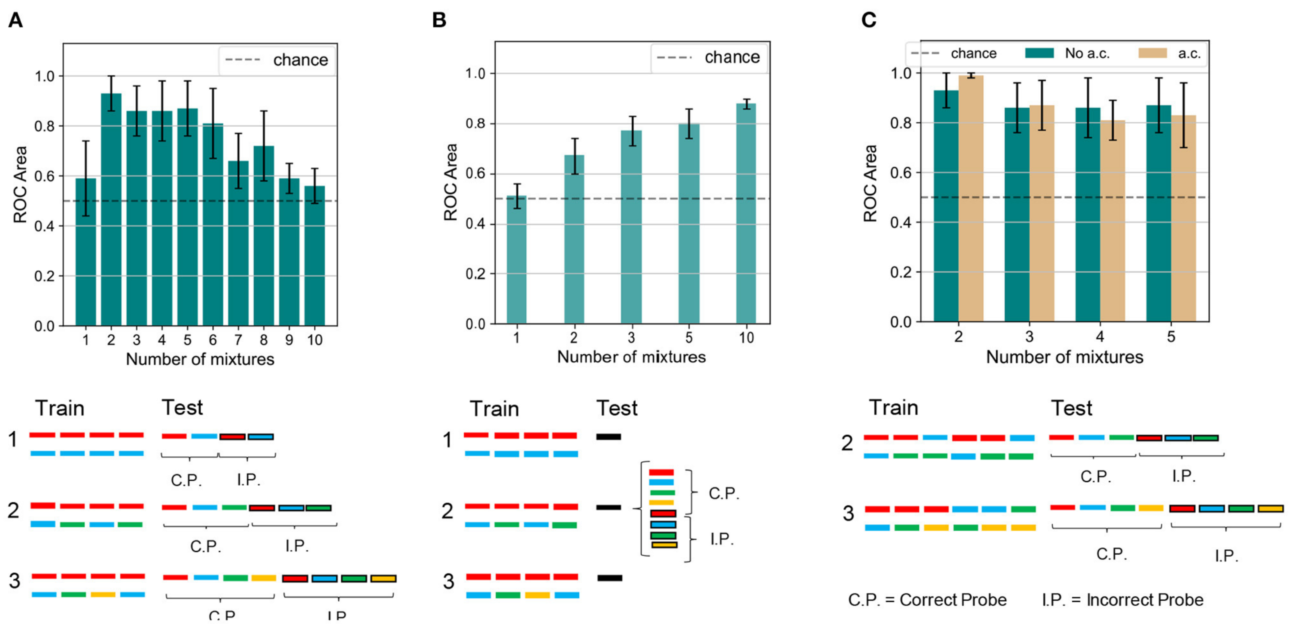


FIGURE 5 | Experiment 1 and 1 a.c.—results and comparison with human performance. **(A)** Results and schematics for Experiment 1 on the dendritic network model. The number of mixtures is varied from 1 to 10. Performance is close to chance for a single training mixture. The performance is boosted as two mixtures are presented. As the number of mixtures is further increased, the clustering accuracy slowly decreases toward chance values. The protocol shown at the bottom of the panel illustrates that (i) in the training phase we feed the network only with the mixture(s), i.e., target+masker sound(s). (ii) in the inference phase we feed the network only with the unmixed sounds (target, distractor separately) and with the incorrect probes (also unmixed sounds). We remark that in the case of one mixture (condition 1) the target and the masker sounds play the same role, while in the case of multiple mixtures (conditions 2 and 3) the target has a different role in the protocol as it is present in more than one mixture while the masker sounds are presented in one mixture only in the training sequence. **(B)** Results and schematics for Experiment 1 on the human experiment. The number of mixtures presented are 1, 2, 3, 5, and 10. For a single mixture the performance is close to chance. As the number of mixtures increases, the classification accuracy improves steadily. Figure reproduced based on data acquired by McDermott et al. (2011). **(C)** Results and schematics for Experiment 1 a.c. on the dendritic network model. The number of mixtures is varied from 2 to 5. Combining all the mixing sounds in mixtures slightly improves the mean performance for two mixing sounds, while it slightly worsens it for a larger number of mixtures. The height of the bars and the error bars show, respectively, mean and standard deviation of the AUC over 10 independent runs.

Our model and human subjects also exhibited interesting differences. When the mixture number was increased to two, performance improved greatly in our model but only modestly in human subjects. Unlike human subjects, our model showed a decreasing accuracy as the number of mixtures further increased. We consider that such discrepancies may arise from a capacity limitation of the network. Indeed, the network architecture is very simple and consists of two layers only, whose size is limited by the spectrogram dimensions for the input layer and by the number of output neurons for the last layer. Therefore the amount of information that the network can learn and store is limited with respect to the significantly more complex structure of the human auditory system. We also suspect that the two-dimensional PCA projection might limit the model performance when a large number of distractors is used. Indeed the PCA space becomes very crowded and although the datapoints are grouped in distinct clusters, the probability that such a cluster lie close to each other is high. To verify this hypothesis, we tested a modification of the inference protocol of the algorithm. During test, we presented the network only with the target sound and one incorrect probe, and performed BSS on the PCA space containing the two sounds. Under this configuration, the model performance is above chance level for two or more different mixtures, and the accuracy does not significantly decrease for large number of mixtures (**Supplementary Figure 2**).

We may use our model for predicting performance of human subjects in auditory perception tasks not yet tested experimentally. To this end, we propose an extension of the paradigm tested previously: for set-ups with the number of mixtures between two and five, we investigated whether presenting all possible combinations of the mixing sounds among themselves, rather than only the distractors with the target, affects the performance. The experiment is labeled "Experiment 1 a.c.," where a.c. stands for "all combinations," and its training scheme is reported in **Figure 5C**. Because all sounds are in principle learnable in the new paradigm, we expect an enhanced ability of distinguishing the correct probes from the incorrect ones. Somewhat unexpectedly, however, our model indicated no drastic changes in the performance when the mixture sequence presented during training contained all possible combinations of the mixing sounds. Such a scheme resulted in a minor improvement in the accuracy only for the experiments with two mixing sounds. Indeed, in the "all

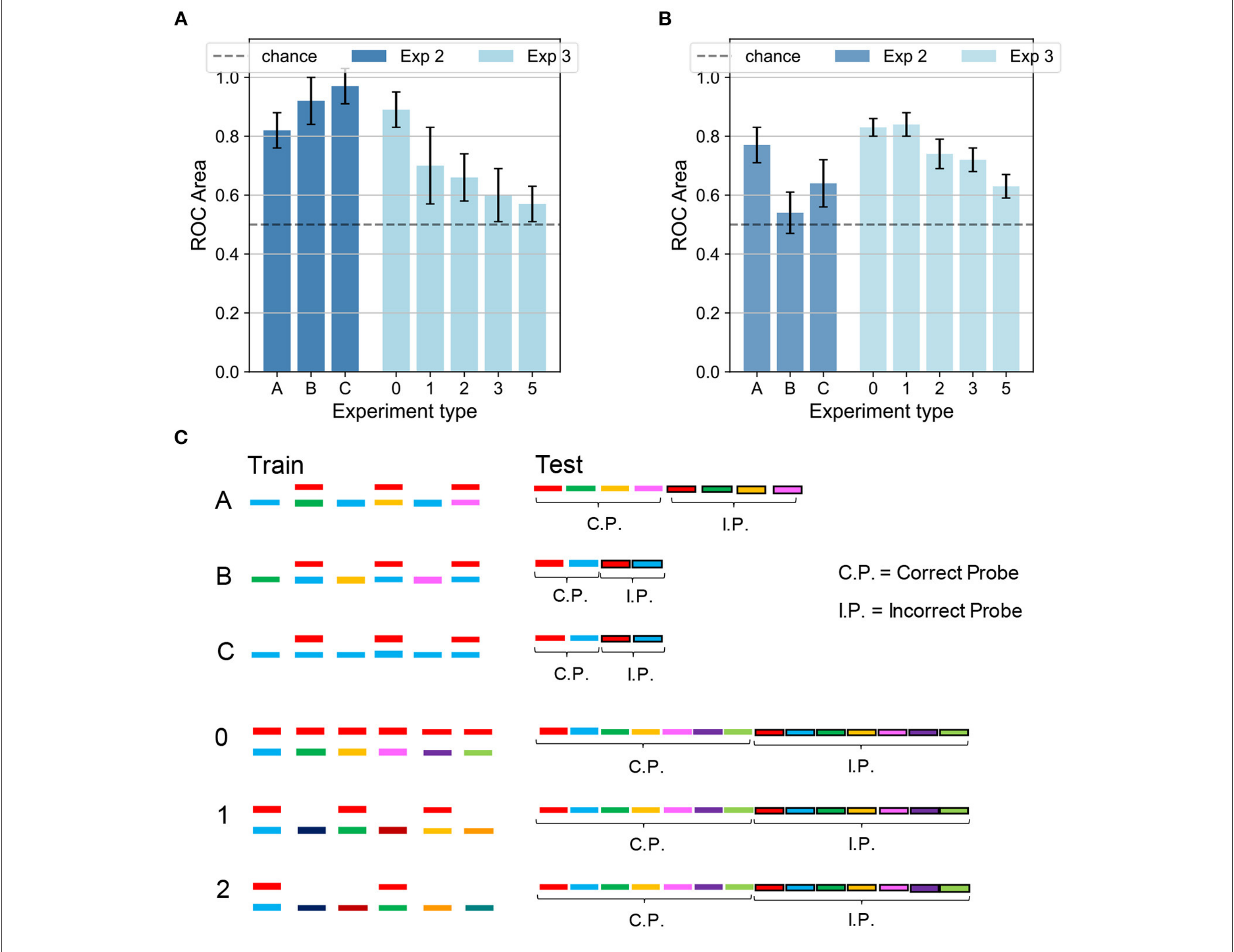


FIGURE 6 | Experiments 2 and 3—results and comparison with human performance. **(A)** Results for Experiments 2 (dark blue) and 3 (light blue) on the dendritic network model. In Experiment 2 the performance is above chance for the three conditions. In Experiment 3 the accuracy decreases as the number of isolated sounds alternating with the mixtures increases. **(B)** Results for Experiments 2 (dark blue) and 3 (light blue) on the human experiment. In Experiment 2 the performance is above chance in the conditions A and C, while it is random for condition B. In Experiment 3 the accuracy decreases as the target presentation is more delayed. Figure reproduced based on data acquired by McDermott et al. (2011). **(C)** Schematics for Experiments 2 and 3. The training is the same for both the dendritic network model and the human experiment. The schematics is omitted for delays 3 and 5. The testing refers to the dendritic network model, while the testing for the human experiment (same as in **Figure 5B**) is omitted. In **(A,B)**, the height of the bars and the error bars show respectively mean and standard deviation of the AUC over 10 independent runs.

combinations" protocol, during training the distractor was presented in more than one different mixture, while in the original task setting only the target was combined with different sounds. We hypothesize that the "all combinations" protocol makes it easier for the network to better distinguish the distractor sound. For four or five mixing sounds, instead, the performance slightly worsened. It is likely that this behavior is related to the already mentioned capacity restraints of the network. Indeed, the length of the training sequence grows as the binomial coefficient $\binom{n}{k}$ where $k = 2$, therefore for four and five targets (i.e., for $n = 4$ or 5) the number of mixtures is increased to 6 and 10, respectively.

2.5. Experiment 2: Sound Segregation With Alternating Multiple Mixtures of Synthesized Sounds

Next, we investigate the model's performance when the training sequence alternated mixtures of sounds with isolated sounds. An analogous protocol was tested in a psychophysical experiment (see experiment 3 in McDermott et al., 2011). **Figures 6A,B** show the network accuracy and human performance, respectively, for the protocols A,B,C in **Figure 6C**. Only the target and the masker sounds were later tested since recognizing the sounds presented individually during training would have been trivial (see conditions B, 1, and 2 in **Figure 6C**). In the alternating task, the network was only partially able to reproduce the human results, displaying an interesting contrast to human behavior. In condition A, in which the sounds mixed with the main target (in red) changed during training, the listeners were able to learn the targets with an accuracy of about 80%, and so did our model. In contrast, our network behaved radically differently with respect to human performance under condition B, in which the training sequence consisted of the same mixture alternating with different sounds. As reported in **Figure 5B**, the listeners were generally not able to identify the single sounds composing the mixture. Our model, instead, unexpectedly achieved a performance well above chance. The output dynamics could distinguish the distractors from the two targets with accuracy surprisingly above 90%. The behavioral discrepancy under condition B could be explained by considering that in the training scheme the network is presented with three different sounds besides the mixture. With respect to Experiment 1 with a single mixture, in this protocol the network could learn the supplementary features of the isolated sounds and could exploit them during inference to respond differently to the distractors. From the spectrograms shown in **Figure 2**, it is evident that some regions of overlap exist between the higher-intensity areas of different sounds. Therefore, the network presented during training with isolated sounds in addition to the single mixture, could detect some similarities between the training sounds and the tested distractors and respond with a more defined output dynamics than in Experiment 1. Finally, under condition C, both human subjects and our model performed above chance. While human performance was slightly above 60%, the network achieved more than 90% accuracy. This result should be interpreted considering that during inference also the isolated sound (blue) was tested together with the associated distractor, which was a trivial task for the nature of our network and thus boosted its overall performance.

2.6. Experiment 3: Effect of Temporal Delay in Target Presentation With Synthesized Sounds

Temporal delay in the presentation of mixtures containing the target degraded performance similarly in the model and human subjects. We presented the network with a training sequence of six mixtures containing the same target mixed each time with a different distractor (**Figure 6C**, protocols 0,1,2: c.f. experiment 4 in McDermott et al., 2011). The mixtures alternated with an increasing number of isolated sounds, hence increasing the interval between successive presentations of the target. The human ability to extract single sounds from mixtures was previously shown to worsen as the interval between target presentations increased, as replicated in **Figure 6B**. The network presented a similar decreasing trend, as reported in **Figure 6A**. An interesting difference, however, is that the performance of our model drastically dropped even with one isolated sound every other mixture while the human performance was affected when at least two isolated sounds separated the target-containing mixtures. The discrepant behavior indicates that the insertion of isolated sounds between the target-containing mixtures more strongly interferes the learning of the target sound in the model compared to human subjects. This stronger performance degradation may partly be due to the capacity constraint of our simple neural model, which uses a larger amount of memory resource as the number of isolated sounds increases. In contrast, such a constraint may be less tight in the human auditory systems.

Also for Experiments 2 and 3, we tested a modification of the inference protocol, by presenting the network only with the target sound and one incorrect probe. Under this configuration, the model performance of Experiment 2 improves compared to the original protocol, while no substantial changes are noted for Experiment 3 (**Supplementary Figure 3**).

2.7. Experiment 4: Sound Segregation With Single and Multiple Mixtures of Real-World Sounds

We applied the same protocol of Experiments 1 to the dataset of natural sounds. Although such experiments were previously not attempted on human subjects, it is intriguing to investigate whether the model can segregate target natural sounds by the same strategy. The spectrograms of two isolated sounds and of their mixtures are shown in **Figures 7A–C**, together with the respective sound waves (**Figures 7D–F**). The qualitative performance was very similar to that obtained with the synthesized sounds. Specifically, the output dynamics learned from the repetition of a single mixture was randomly fluctuating for both seen and randomly chosen unseen sounds (**Figure 8A**), whereas the network responses to targets and unseen sounds were clearly distinct if multiple mixtures were presented during training (**Figure 8D**). The output dynamics were not quantitatively evaluated because it was not possible to rigorously generate incorrect probes associated with the learnt targets and distractors. Therefore, we qualitatively assessed the performance of the model by observing the clustering of network responses to the learnt targets vs. unseen natural sounds (**Figures 8B–F**). We observed that, in the case of multiple mixtures, the clusters related to natural sounds (**Figures 8E,F**) were more compact than those of synthetic sounds (**Figures 4E,F**). Furthermore, these clusters were more widely spaced on the PCA projection plane: the intraclass correlation in the response to the same target was greater while the interclass similarity in the response to different targets or distractors was lower. These results indicate that grouping cues, such as harmonic structure and temporal onset, improve the performance of the model.

2.8. Experiment 5: Image Segregation With Single and Multiple Mixtures of Real-World Images

Finally, we examined whether the source segregation through repetition scheme can also extend to vision-related tasks, as previously suggested (McDermott et al., 2011). To this end, we employed the same method as developed for sound sources and performed the recovery of visual sources with the protocol of Experiment 1. The mixtures were obtained by overlapping black-and-white images sampled from our visual dataset (Section 4), as shown in **Figure 9**. Similarly to Experiment 4, the performance of the model was assessed only qualitatively in the

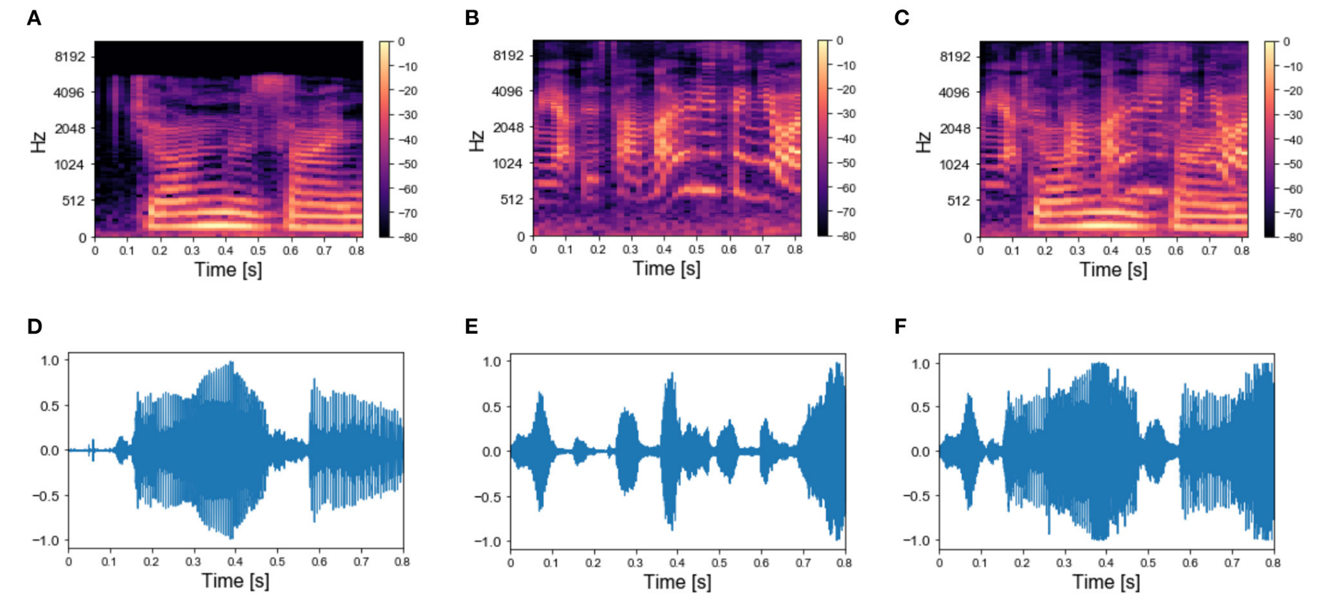


FIGURE 7 | Real-world sounds—targets and mixture. **(A)** Spectrogram of a spoken sentence 800 ms-long. **(B)** Spectrogram of 800 ms-long recording of chimes sounds. **(C)** Spectrogram of the mixture of the sounds in **(A,B)**. **(D)** Sound wave associated with the spectrogram in **(A)**. **(E)** Sound wave associated with the spectrogram in **(B)**. **(F)** Sound wave associated with the spectrogram in **(C)**.

visual tasks. As in the acoustic tasks, the clustering of network responses showed that the model was able to retrieve the single images only when more than one mixture was presented during training. The network responses are shown in **Figure 10**. We remark that the model is presented with the visual stimuli following the same computational steps as for sounds. Indeed, as previously described, the acoustic stimuli are first pre-processed into spectrograms and then encoded by the input layer. While it is not unexpected that similar computational steps lead to consistent results, we remark that the nature of the "audio images," i.e., the spectrograms, is substantially different to that of the naturalistic images, leading to very different distributions of the encoding spike patterns. Therefore, successful signal discrimination in the visual task strengthens our results, proving that our model is robust with respect to different arrangements of signal intensity.

3. DISCUSSION

The recovery of individual sound sources from mixtures of multiple sounds is a central challenge of hearing. Based on experiments on human listeners, sound segregation has been postulated to arise from prior knowledge of sound characteristics or detection of repeating spectro-temporal structure. The results of McDermott et al. (2011) show that a sound source can be recovered from a sequence of mixtures if it occurs more than once and is mixed with more than one masker sound. This supports the hypothesis that the auditory system detects repeating spectro-temporal structure embedded in mixtures, and interprets this structure as a sound source. We investigated whether a biologically inspired computational model of the auditory system can account for the characteristic performance of human subjects. To this end, we implemented a one-layer neural network with dendritic neurons followed by a readout layer based on GMM to classify probe sounds as seen or unseen in the training mixtures. The results in McDermott et al. (2011) show that source repetition can be detected by integrating information over time and that the auditory system can perform sound segregation when it is able to recover the target sound's latent structure. Motivated by these findings, we trained our dendritic model with a learning rule that was previously demonstrated to detect and analyze the temporal structure of a stream of signals. In particular, we relied on the learning rule described by Asabuki and Fukai (2020), which is based on the minimization of regularized information loss. Specifically, such a principle enables the self-supervised learning of recurring temporal features in information streams using a family of competitive networks of somatodendritic neurons. However, while the learning rule has been designed to capture temporal information in an online fashion, in our framework we flatten the spectrogram before encoding it, making the spike pattern static during the stimulus presentation. Therefore, the temporal fluctuations are determined by the stochastic processes in the rate encoding step.

We presented the network with temporally overlapping sounds following the same task protocols as described in McDermott et al. (2011). First, we carried out the segregation task with the same dataset of synthesized sounds presented to human listeners in McDermott et al. (2011). We found that the model was able to segregate sounds only when one of the masker sounds varied, not when both sounds of the mixture were repeated.

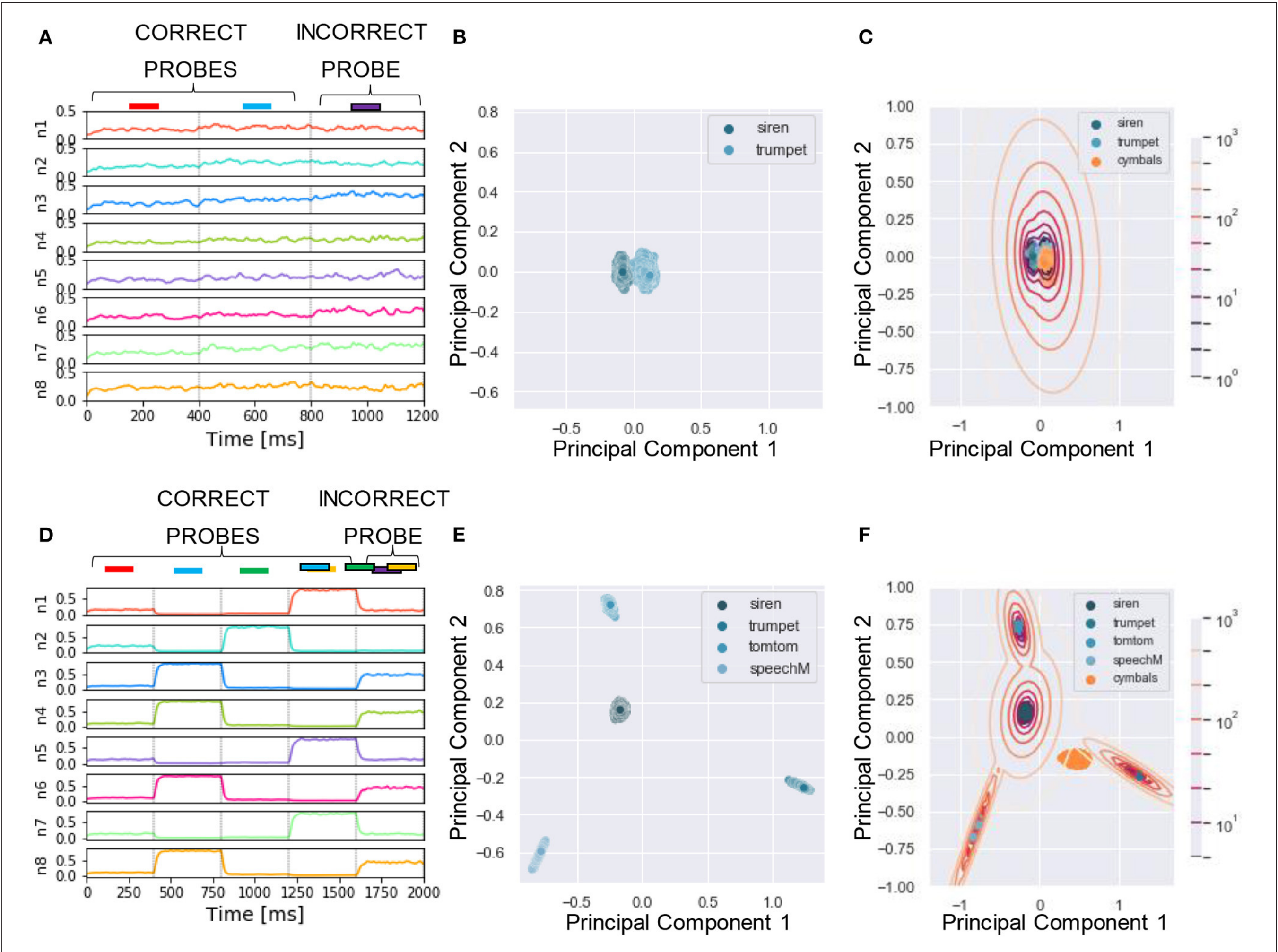


FIGURE 8 | Experiment 4—output dynamics and clustering. **(A–C)** refer to the results of Experiment 4 on real-world sounds with a single mixture presented during training. **(D–F)** refer to the results of Experiment 4 on real-world sounds with three mixtures presented during training. **(A)** Voltage dynamics of the 8 output neurons during inference, when the target, the distractor and one unseen sound are tested. As expected, the neuron population is not able to respond with different dynamics to the three sounds, and the voltage of all the output neurons fluctuates randomly throughout the whole testing sequence. **(B)** The PCA projection of the datapoints belonging to the target and distractor (in blue) shows that the clusters are collapsed into a single cluster. **(C)** When GMM is applied, all the datapoints representing both the learnt sounds (in blue) and the unseen sound (in orange) fall within the same regions, making it impossible to distinguish the different sounds based on the population dynamics. **(D)** Voltage dynamics of the 8 output neurons during inference, when the target, the three distractors, and one unseen sound are tested. As expected, the neuron population has learnt the feature of the different sounds and responds with different dynamics to the five sounds. Each output neuron has an enhanced response to one or few sounds. **(E)** The PCA projection of the datapoints belonging to the four correct probes (in blue) shows that the clusters are more compact and more spatially distant one from the other with respect to the result obtained with the synthetized sounds. **(F)** When GMM is applied, the model shows that the network clearly distinguished the learnt sounds (in blue) from the unseen sound (in orange). These results show that the grouping cues improve the model accuracy with respect to the synthesized dataset.

Our findings bear a closer resemblance to the experimental findings of human listeners over a variety of task settings. Earlier works have proposed biologically inspired networks to perform BSS (Pehlevan et al., 2017; Isomura and Toyoizumi, 2019; Bahroun et al., 2021). However, to our knowledge, this is the first attempt to reproduce the experimental results of recovering sound sources through embedded repetition. For this reason, we could not compare our results with previous works. Additionally, we demonstrated that our network can be a powerful tool for predicting the dynamics of brain segregation capabilities under settings difficult to test on humans. In particular, the recovery of natural sounds is expected to be a trivial task for humans given their familiarity with the sounds, whereas our model is built from scratch and has no prior knowledge about natural sounds. We find that the hallmarks of natural sounds make the task easier for the network when the target is mixed with different sounds, but, as for the synthetic dataset, the sounds cannot be detected if presented always in the same mixture. Furthermore, we extended the study to investigate BSS of visual stimuli and observed a similar qualitative performance as in the auditory settings. This is not surprising from a computational perspective as the computational steps

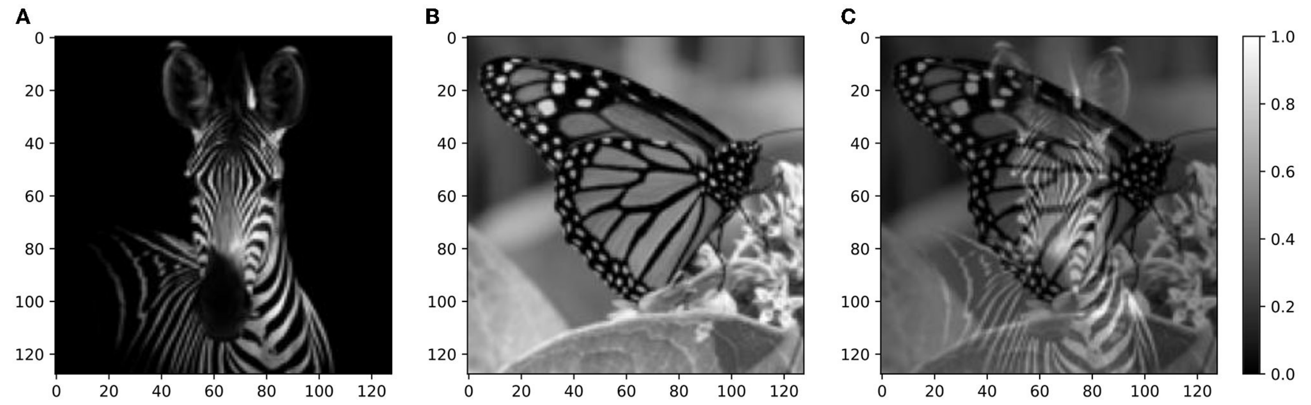


FIGURE 9 | Real-world images—targets and mixture. **(A)** Squared 128 × 128 target image of a zebra. **(B)** Squared 128 × 128 distractor image of a butterfly. **(C)** Mixture of the target and distractor images shown in **(A,B)**. Source: Shutterstock.

of the visual experiment are the same as for the acoustic experiment: there, the sounds are first preprocessed into images, the spectrograms, and then presented to the network in a visual form. From the biological point of view, the neural computational primitives used in the visual and the auditory cortex may be similar, as evidenced by anatomical similarity and by developmental experiments where auditory cortex neurons acquire V1-like receptive fields when visual inputs are redirected there (Sharma et al., 2000; Bahroun et al., 2021). We point out, however, that such a similarity is valid only at high level as there are some substantial differences between visual and auditory processing. For instance, the mechanisms to encode the input signal into spikes rely on different principles: in the retina the spike of a neuron indicates a change in light in the space it represents, while in the cochlea the rate of a neurons represents the amplitude of the frequency it is associated to, like a mechanical FFT. Motivated by these reasons, we suggest extending the experiments of source repetition to vision to verify experimentally whether our computational results provide a correct prediction of the source separation dynamics of the visual system.

Although the dynamics of our model under many aspects matches the theory of repetition-based BSS, the proposed scheme presents a few limitations. The major limitation concerns the discrepancy of the results in experiment 2B. In such a setting, the model performance is well above chance, although the target sound always occurs in the same mixture. We speculate that, in this task settings, the output neurons learn the temporal structure of the distractor sounds presented outside the mixture and that they recognize some similarities in the latent structure of the probes. We note that the degree of similarity among distractors is the same as in the psychophysics experiment. This pushes the neurons to respond differently to the correct and incorrect probes, thereby allowing the output classifier to distinguish the sounds. In contrast, we speculate that human auditory perception relies also on the outcome of the later integration of features detected at early processing stages. This will prevent the misperception of sounds based on unimportant latent features. A second limitation of the selected encoding method consists in the difficulty to model the experiments relying on the asynchronous overlapping of signals and on reversed probe sounds presented by McDermott et al. (2011). Indeed, in our approach, because of the flattening of the spectrogram in the encoding phase, each input neuron responds to one specific time frame, and the output neurons are trained uniquely on this configuration. Hence, temporal shifts or inverting operations are not possible. Third, we observed that in Experiment 1, as the number of mixtures increased over a certain threshold, the model's accuracy degraded. We speculate that, in such settings, substituting PCA with a clustering algorithm not relying on dimensionality reduction, such as K-means, may help mitigate the issue. In addition, an interesting variation of our framework would be replacing the clustering step of the model with an another layer of spiking neurons. Fourth, the flattening of the spectrogram in the spike encoding stage is not biologically plausible and introduces high latency as the entire input signal needs to be buffered before the encoding starts. This strategy exhibits the advantage of making the length of the spike train fixed for any sound length, though modifications of the encoding scheme that preserves the signal's temporal structure might be more suitable for applications tailored for real-world devices. Furthermore, an instantaneous identity coding approach, either from raw signal or *via* a spectrogram, would not be affected by the previously described issues related to the spectrogram normalization in the presence of outliers in signal intensity. Motivated by these points, in a follow up work we intend to explore an extension of the presented framework combining time frame-dependent encoding and spike-based post-processing clustering, which would allow us to integrate the model in embedded neuromorphic applications for sound source separation with reduced response latency. In this context, for further lowering the temporal latency, as well as for reducing the model's energy consumption in neuromorphic devices, the time-to-first-spike

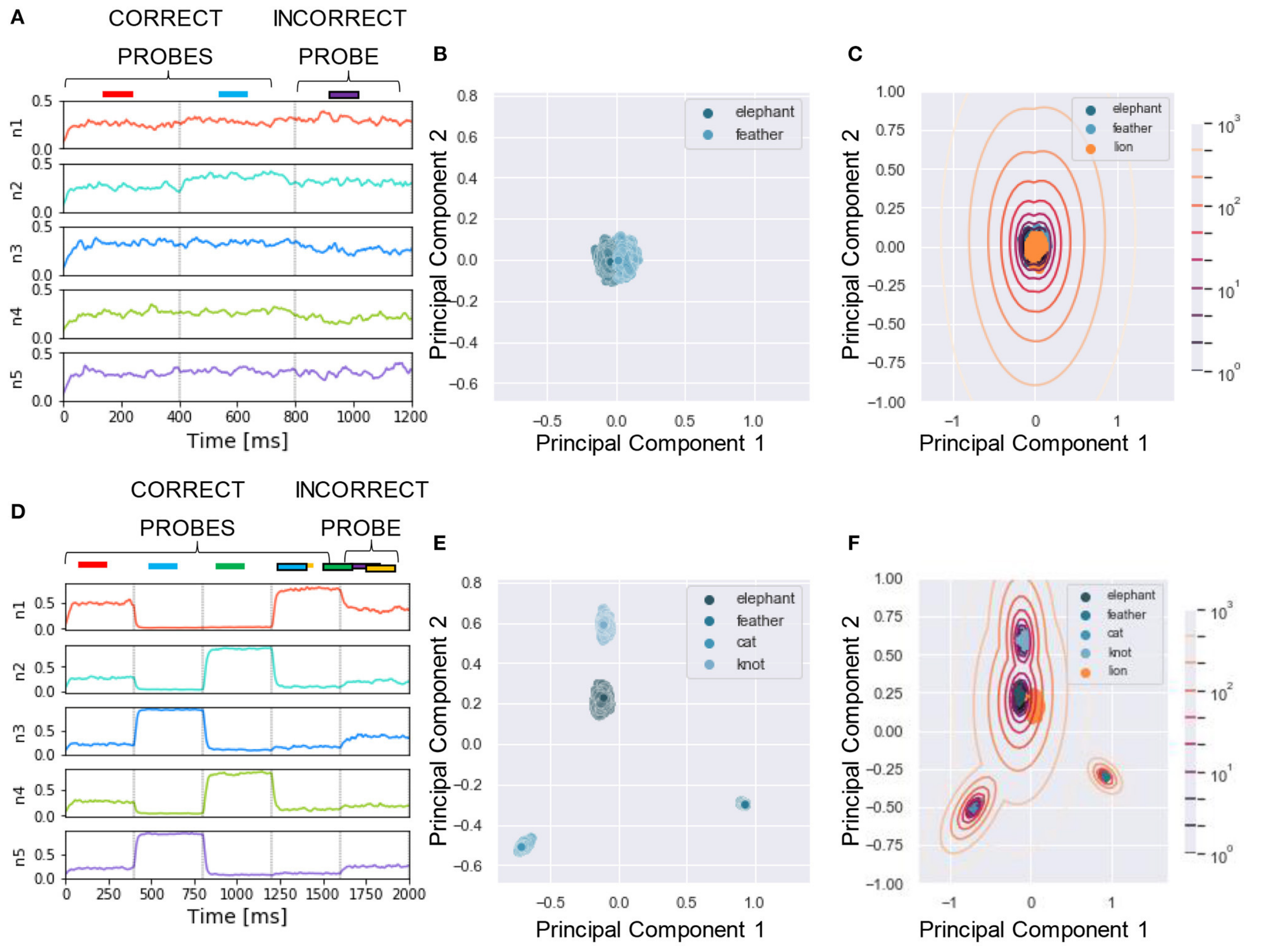


FIGURE 10 | Experiment 5—output dynamics and clustering. **(A–C)** refer to the results of Experiment 5 on real-world images with a single mixture presented during training. **(D–F)** refer to the results of Experiment 5 on real-world images with three mixtures presented during training. **(A)** Voltage dynamics of the 5 output neurons during inference, when the two training images and one unseen image are tested. As expected, the neuron population is not able to respond with different dynamics to the three images, and the voltage of all the output neurons fluctuates randomly throughout the whole testing sequence. **(B)** The PCA projection of the datapoints belonging to the two seen images (in blue) shows that the clusters are collapsed into a single cluster. **(C)** When GMM is applied, all the datapoints representing both the targets (in blue) and the unseen image (in orange) fall within the same regions, making it impossible to distinguish the different images based on the population dynamics. **(D)** Voltage dynamics of the 5 output neurons during inference, when the four targets and one unseen image are tested. As expected, the neuron population has learnt the features of the different images and responds with different dynamics to the five images. Each output neuron has an enhanced response to one or few inputs. **(E)** The PCA projection of the datapoints belonging to the four learnt images (in blue) shows that the clusters are compact and spatially distant one from the other. **(F)** When GMM is applied, the model shows that the network clearly distinguished the target and distractors (in blue) from the unseen image (in orange). These results suggest that humans would be able to distinguish single visual targets if previously seen in different mixtures.

encoding method could be explored as an alternative to the current rate coding approach.

Furthermore, as previously mentioned, the training scheme in Asabuki and Fukai (2020) has proven to be able to learn temporal structures in a variety of tasks. In particular, the model was shown to perform chunking as well as to achieve BSS from mixtures of mutually correlated signals. We underline that our computational model and experiments differ in fundamental ways from the BSS task described by Asabuki and Fukai (2020). First, the two experiments diverge in their primary scope. The BSS task aims at using the average firing rate of the single neurons responding to sound mixtures to decode separately the original sounds. In our work, instead, sound mixtures are included only in the training sequence and, during inference, only individual sounds are presented to the network. Our goal is to verify from the population activity whether the neurons have effectively learned the sounds and can distinguish them from unseen distractors. Furthermore, in Asabuki and Fukai (2020) the stimulus was encoded into spike patterns using one Poisson process proportional to the amplitude of the sound waveform at each time step, disregarding the signal intensity at different frequencies. This method was not suitable for the source segregation through repetition task, where the sound mixtures retain important information on the frequency features of the original sounds at each time frame. Furthermore, we flatten the audio signal spectrogram before encoding it, unlike in the BSS task described by Asabuki and Fukai (2020).

In summary, we have shown that a network of dendritic neurons trained in an unsupervised fashion is able to learn the features of overlapping sounds and, once the training is completed, can perform blind source separation if the individual sounds have been presented in different mixtures. These results account for the experimental performance of human listeners tested on the same task setting. Our study has demonstrated that a biologically inspired simple model of the auditory system can capture the intrinsic neural mechanisms underlying the brain's capability of recovering individual sound sources based on repetition protocols. Furthermore, as the adopted learning scheme in our model is local and unsupervised, the network is self-organizing. Therefore, the proposed framework opens up new computational paradigms with properties specifically suited for embedded implementations of audio and speech processing tasks in neuromorphic hardware.

4. MATERIALS AND METHODS

4.1. Datasets

A dataset of synthesized sounds were created in the form of spectrogram, which shows how signal strength evolves over time at various frequencies, according to the method described previously (McDermott et al., 2011). In short, the novel spectrograms were built as Gaussian distributions based on correlation functions analogous to those of real-world sounds. White noise was later applied to the resulting spectrograms. Five Gaussian distributions were employed to generate each of ten different sounds in **Figure 5A**. The corresponding spectrograms featured 41 frequency filters equally spaced on an ERBN (Equivalent Rectangular Bandwidth, with subscript N denoting normal hearing) scale (Glasberg and Moore, 1990) spanning 20–4,000 Hz, and 33 time frames equally dividing the 700 ms sound length. For our simulations, we used the same MATLAB toolbox and parameters used in the previous study (McDermott et al., 2011). For further details on the generative model for sounds, please refer to the SI Materials and Methods therein.

In addition to the dataset of synthesized sounds, we built a database composed of 72 recordings of isolated natural sounds. The database contained 8 recordings of human speech from the EUSTACE (the Edinburgh University Speech Timing Archive and Corpus of English) speech corpus (White and King, 2003), 23 recordings of animal vocalizations from the Animal Sound Archive (Frommolt et al., 2006), 29 recordings of music instruments by Philharmonia Orchestra (Philarmonia Orchestra Instruments, 2019), and 12 sounds produced by inanimate objects from the BBC Sound Effect corpus (BBC, 1991). The sounds were cut into 800 ms extracts. Then the library librosa (McFee et al., 2015) was employed to extract spectrograms with 128 frequency filters spaced following the Mel scale (Stevens et al., 1937) and 10 ms time frames with 50% overlap.

For image source separation, we built a database consisting of 32 black-and-white pictures of various types, both single objects and landscapes. The images were later squared, and their size was reduced to 128×128 pixels.

4.2. Neuron Model

In this study we used the same two-compartment neuron model as that developed previously (Asabuki and Fukai, 2020). The mathematical details are found therein. Here, we only briefly outline the mathematical framework of the neuron model. Our two-compartment model learns temporal features of synaptic input given to the dendritic compartment by minimizing a regularized information loss arising in signal transmission from the dendrite to the soma. In other words, the two-compartment neuron extracts the characteristic features of temporal input by compressing the high dimensional data carried by a temporal sequence of presynaptic inputs to the dendrite onto a low dimensional manifold of neural dynamics. The model performs this temporal feature analysis by modifying the weights of dendritic synapses to minimize the time-averaged mismatch between the somatic and dendritic activities over a certain recent interval. In a stationary state, the somatic membrane potential of the two-compartment model could be described as an attenuated version of the dendritic membrane potential with an attenuation factor (Urbanczik and Senn, 2014). Though we deal with time-dependent stimuli in our model, we compare the attenuated dendritic membrane potential with the somatic membrane potential at each time point. This comparison, however, is not drawn directly on the level of the membrane potentials but on the level of the two non-stationary Poissonian spike distributions with time-varying rates, which would be generated if both soma and dendrite were able to emit spikes independently. In addition, the dynamic range of somatic responses needs to be appropriately rescaled (or regularized) for meaningful comparison. An efficient learning algorithm for this comparison can be derived by minimizing the Kullback–Leibler (KL) divergence between the probability distributions of somatic and dendritic activities. Note that the resultant learning rule enables unsupervised learning because the somatic response is fed back to the dendrite to train dendritic synapses. Thus, our model proposes the view that backpropagating action potentials from the soma may provide a supervising signal for training dendritic synapses (Larkum et al., 1999; Larkum, 2013).

4.3. Network Architecture

The network architecture, shown in **Figure 1**, consisted of two layers of neurons, either fully connected or with only 30% of the total connections. The input layer contained as many Poisson neurons as the number of pixels present in the input spectrogram (acoustic stimulus) or input image (visual stimulus). The postsynaptic neurons were modeled according to the two-compartment neuron model proposed previously (Asabuki and Fukai, 2020). Their number was varied from a pair to few tenths, depending on the complexity of the task. Unless specified otherwise, 8 and 5 output neurons were set for acoustic and visual task respectively.

In the first layer, the input was encoded into spikes through a rate coding-based method (Almomani et al., 2019). The strength of the signal at each pixel drove the firing rate of the associated input neuron, i.e., the spike trains were drawn from Poisson point processes with probability proportional to the intensity of the pixel. We imposed that, for each input stimulus, the spike pattern was generated through a sequence of 400 time steps, where each time step corresponds to a "fire" or "non-fire" event.

We designed the output layer and the learning process similarly to the previous network used for the blind signal separation (BSS) within mixtures of multiple mutually correlated signals as well as for other temporal feature analyses (Asabuki and Fukai, 2020). As mentioned previously, the learning rule was modeled as a self-supervising process, which is at a conceptual level similar to Hebbian learning with backpropagating action potentials. The soma generated a supervising signal to learn and detect the recurring spatiotemporal patterns encoded in the dendritic activity. Within the output layer, single neurons learned to respond differently to each input pattern. Competition among neurons was introduced to ensure that different neurons responded to different inputs. With respect to the network used for BSS containing only two output neurons, we rescaled the strength of the mutual inhibition among dendritic neurons by a factor proportional to the inverse of the square root of the number of output neurons. This correction prevented each neuron from being too strongly inhibited when the size of the output layer increased (i.e., exceeds three or four). Furthermore, we adopted the same inhibitory spike timing-dependent plasticity (iSTDP) as employed in the previous model. This rule modified inhibitory connections between two dendritic neurons when they coincidently responded to a certain input. The iSTDP allowed the formation of chunk-specific cell assemblies when the number of output neurons was greater than the number of input patterns.

For all parameters but noise intensity ξ_i during learning, we used the same values as used in the original network model (Asabuki and Fukai, 2020). For bigger values of noise intensity g, the neural responses were subject to more fluctuations and neurons tended to group in only one cell assembly. From the analysis of the learning curves shown in **Figure 3**, we decided to train the network from randomly initialized weights and to expose it, during training, to the mixture sequence 3,000 times for the synthesized sounds and 1500 times for the real-world sounds. The learning rate was kept constant throughout the whole process. During testing, the sequence of target sounds and respective distractors was presented 50 times, and the resulting neural dynamics was averaged over 20 trials. The performance results shown in the section 2 were computed as average over 10 repetitions of the same simulation set-up. In each repetition different target sounds and distractors were randomly sampled from the dataset in order to ensure performance independence of specific sounds.

4.4. Experimental Settings and Performance Measure

The synapses were kept fixed during inference in our network, implying that the responses to probes tested later were not affected by the presentation of other previously tested probes. This allowed us to test the trained network on a sequence of probes, rather than only on one probe as in the studies of the human brain where plasticity cannot be frozen during inference (McDermott et al., 2011). In **Figures 5A**, **6C**, the first half of the sequence contained the target and the distractors, the second half the respective incorrect probes, which were also built by using the same method as in human experiment (McDermott et al., 2011). Each incorrect probe was a sound randomly selected from the same Gaussian distribution generating the associated target. After the sampling, a randomly selected time slice equal to 1/8 of the sound duration was set to be equal to the target.

The possibility of presenting more than one probe allowed us to test the performance of the network for all the sounds present in the mixtures. To ensure a stable neural response against the variability of the encoding, we repeated the sequence 50 times. The response of the network consisted of the ensemble activity of the output neurons. As previously explained, 400 time-steps were devoted to the presentation to each stimulus. The response to each probe, therefore, consisted of 400 data points describing the dynamical activity of each output neuron, each point being a collection of N values, where N is the number of output neurons. An example of one testing epoch output is shown in **Figures 4A,C**. We neglected the first 50 data points, since, during the initial transient time, the membrane potential was still decaying or rising after the previous input presentation. For visualization purpose, we applied the principal component analysis (PCA) to reduce the dimensionality of the data from N to 2. In our settings, the two principal components explain approximately 40% of the variance of the neural response. The PCA transformation was based uniquely on the data points obtained with the presentation of the target and the distractors, as shown in **Figures 4B,E**. The same transformation was later exploited to project the points related to the incorrect probes. Only the target and distractors patterns were presented during the learning process, and the responses to unseen patterns were afterwards projected on the space defined by the training.

The two-dimensional projection of the target-related data points were clustered in an unsupervised manner through GMM. We set the number of Gaussians equal to the number of targets such that the covariance matrices had a full rank. With the defined GMM model at hand, we proceeded with fitting all the PCA data points, related to both correct and incorrect probes. The model tells which cluster each data point belonged to and what was the likelihood (L) that the cluster had generated this data point. **Figures 4C,F** show the datapoints projected on the PCA plane together with the GMM clustering and likelihood curves.

We used the likelihood as a measure of performance. The four intervals of the likelihood range corresponding to the four responses "sure no," "no," "yes," and "sure yes" were (i) $L > 0$ (sure yes), (ii) $-5 < L < 0$ (yes), (iii) $-15 < L < -5$ (no), and (iv) $L < -15$ (sure no). In building the receiver operating characteristic (ROC) curve, the datapoints falling in the interval (i) were assigned the probability value 1.0, those in (ii) 0.66, those in (iii) 0.33, and those in (iv) 0.0.

The described evaluation metrics was applied only to the experiments carried on the dataset composed of synthesized sounds. For the experiments based on natural sounds and images, the results of clustering were shown only qualitatively for the target-related datapoints. Indeed, due to the real-world nature of signals, it was not possible to simply use Gaussian functions to build physically consistent incorrect probes. On the real-world sound dataset, we performed all the same protocol of Experiment 1 (Experiment 4). On the image dataset we performed an experiment with a protocol analogous to Experiment 1. Here, the mixtures were obtained by overlapping two images, both with transparency 0.5, similarly to the spectrogram overlapping described for the acoustic task. The input images were normalized to the range [0,1] and the intensity of each pixel was encoded through the firing rate of one input neuron. We followed the same procedure and network setting described for the audio stimuli segregation to assess the ability of the network to separate visual stimuli presented in mixtures.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: https://github.com/GiorgiaD/dendritic-neuron-BSS.

AUTHOR CONTRIBUTIONS

TF, GD, and TA conceived the idea. GD designed and performed the simulations, with input from TA. GD and TF wrote the manuscript. TA and GD wrote the **Supplementary Material**. All authors analyzed the results. All authors contributed to the article and approved the submitted version.

FUNDING

This work was partly supported by JSPS KAKENHI no. 19H04994 to TF.

ACKNOWLEDGMENTS

We are grateful to all the colleagues in the Neural Coding and Brain Computing Unit for fruitful interaction.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2022.855753/full#supplementary-material

REFERENCES

Ahveninen, J., Hämäläinen, M., Jääskeläinen, I. P., Ahlfors, S. P., Huang, S., Lin, F.-H., et al. (2011). Attention-driven auditory cortex short-term plasticity helps segregate relevant sounds from noise. *Proc. Natl. Acad. Sci. U.S.A.* 108, 4182–4187. doi: 10.1073/pnas.1016134108

Almomani, D., Alauthman, M., Alweshah, M., Dorgham, O., and Albalas, F. (2019). A comparative study on spiking neural network encoding schema: implemented with cloud computing. *Cluster Comput.* 22, 419–433. doi: 10.1007/s10586-018-02891-0

Amari, S., Cichocki, A., and Yang, H. (1995). "A new learning algorithm for blind signal separation," in *NIPS'95: Proceedings of the 8th International Conference on Neural Information Processing Systems* (Cambridge, MA), 757–763.

Asabuki, T., and Fukai, T. (2020). Somatodendritic consistency check for temporal feature segmentation. *Nat. Commun.* 11, 1554. doi: 10.1038/s41467-020-15367-w

Asari, H., Pearlmutter, B. A., and Zador, A. M. (2006). Sparse representations for the cocktail party problem. *J. Neurosci.* 26, 7477–7490. doi: 10.1523/JNEUROSCI.1563-06.2006

Atilgan, H., Town, S. M., Wood, K. C., Jones, G. P., Maddox, R. K., Lee, A. K., et al. (2018). Integration of visual information in auditory cortex promotes auditory scene analysis through multisensory binding. *Neuron* 97, 640.e4–655.e4. doi: 10.1101/098798

Bahroun, Y., Chklovskii, D. B., and Sengupta, A. M. (2021). A normative and biologically plausible algorithm for independent component analysis. *arXiv [Preprint]*. arXiv: 2111.08858. doi: 10.48550/arXiv.2111.08858

BBC. (1991). *BBC sound effects library. Compact disc.; Digital and Analog Recordings.; Detailed Contents on Insert in Each Container.;Recorded: 1977–1986.* Princeton, NJ: Films for the Humanities and Sciences.

Bee, M., and Micheyl, C. (2008). The cocktail party problem: what is it? How can it be solved? and why should animal behaviorists study it? *J. Comp. Psychol.* 122, 235–251. doi: 10.1037/0735-7036.122.3.235

Bell, A., and Sejnowski, T. (1995). An information-maximization approach to blind separation and blind deconvolution. *Neural Comput.* 7, 1129–1159.

Bronkhorst, A. (2015). The cocktail-party problem revisited: early processing and selection of multi-talker speech. *Attent. Percept. Psychophys.* 77, 1465–1487. doi: 10.3758/s13414-015-0882-9

Brown, G., Yamada, S., and Sejnowski, T. (2001). Independent component analysis at neural cocktail party. *Trends Neurosci.* 24, 54–63. doi: 10.1016/S0166-2236(00)01683-0

Cherry, E. C. (1953). Some experiments on the recognition of speech, with one and with two ears. *J. Acoust. Soc. Am.* 25, 975–979.

Cichocki, A., Zdunek, R., and Amari, S. (2006). "New algorithms for non-negative matrix factorization in applications to blind source separation," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings* (Toulouse).

Comon, P. (1994). Independent component analysis, a new concept? *Signal Process.* 36, 287–314.

Ding, N., and Simon, J. Z. (2012). Neural coding of continuous speech in auditory cortex during monaural and dichotic listening. *J. Neurophysiol.* 107, 78–89. doi: 10.1152/jn.00297.2011

Dong, J., Colburn, H. S., and Sen, K. (2016). Cortical transformation of spatial processing for solving the cocktail party problem: a computational model. *eNeuro* 3, 1–11. doi: 10.1523/ENEURO.0086-15.2015

Elhilali, M. (2013). "Bayesian inference in auditory scenes," in *Conference Proceedings : Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (Osaka), 2792–2795.

Elhilali, M., and Shamma, S. (2009). A cocktail party with a cortical twist: how cortical mechanisms contribute to sound segregation. *J. Acoust. Soc. Am.* 124, 3751–3771. doi: 10.1121/1.3001672

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends Cogn. Sci.* 3, 128–135. doi: 10.1016/S1364-6613(99)01294-2

Frommolt, K. -H., Bardeli, R., Kurth, F., and Clausen, M. (2006). *The Animal Sound Archive at the Humboldt-University of Berlin: Current Activities in Conservation and Improving Access for Bioacoustic Research.* Ljubljana: Slovenska akademija znanosti in umetnosti.

Glasberg, B. R., and Moore, B. C. (1990). Derivation of auditory filter shapes from notched-noise data. *Hear. Res.* 47, 103–138.

Golumbic, E. Z., Cogan, G. B., Schroeder, C. E., and Poeppel, D. (2013). Visual input enhances selective speech envelope tracking in auditory cortex at a "cocktail party". *J. Neurosci.* 33, 1417–1426. doi: 10.1523/JNEUROSCI.3675-12.2013

Hawley, M. L., Litovsky, R. Y., and Culling, J. F. (2004). The benefit of binaural hearing in a cocktail party: effect of location and type of interferer. *J. Acoust. Soc. Am.* 115, 833–843. doi: 10.1121/1.1639908

Haykin, S., and Chen, Z. (2005). The cocktail party problem. *Neural Comput.* 17, 1875–1902. doi: 10.1162/0899766054322964

Hyvärinen, A., and Oja, E. (1997). A fast fixed-point algorithm for independent component analysis. *Neural Comput.* 9, 1483–1492.

Isomura, T., and Toyoizumi, T. (2019). Multi-context blind source separation by error-gated Hebbian rule. *Sci. Rep.* 9, 7127. doi: 10.1038/s41598-019-43423-z

Jacobsen, T., Schröger, E., Winkler, I., and Horváth, J. (2005). Familiarity affects the processing of task-irrelevant auditory deviance. *J. Cogn. Neurosci.* 17, 1704–1713. doi: 10.1162/089892905774589262

Kameoka, H., Li, L., Inoue, S., and Makino, S. (2018). Semi-blind source separation with multichannel variational autoencoder. *arXiv preprint arXiv:1808.00892*. doi: 10.48550/arXiv.1808.00892

Karamatli, E., Cemgil, A. T., and Kirbiz, S. (2018). "Weak label supervision for monaural source separation using non-negative denoising variational autoencoders," in *2019 27th Signal Processing and Communications Applications Conference (SIU)* (Sivas).

Kerlin, J., Shahin, A., and Miller, L. (2010). Attentional gain control of ongoing cortical speech representations in a "cocktail party". *J. Neurosci.* 30, 620–628. doi: 10.1523/JNEUROSCI.3631-09.2010

Krause-Solberg, S., and Iske, A. (2015). "Non-negative dimensionality reduction for audio signal separation by NNMF and ICA," in *2015 International Conference on Sampling Theory and Applications, SampTA 2015* (Washington, DC), 377–381.

Krishnan, L., Elhilali, M., and Shamma, S. (2014). Segregating complex sound sources through temporal coherence. *PLoS Comput. Biol.* 10, e1003985. doi: 10.1371/journal.pcbi.1003985

Larkum, M. (2013). A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends Neurosci.* 36, 141–151. doi: 10.1016/j.tins.2012.11.006

Larkum, M., Zhu, J., and Sakmann, B. (1999). A new cellular mechanism for coupling inputs arriving at different cortical layers. *Nature* 398, 338–341.

Lewald, J., and Getzmann, S. (2015). Electrophysiological correlates of cocktail-party listening. *Behav. Brain Res.* 292, 157–166. doi: 10.1016/j.bbr.2015.06.025

Li, Y., Wang, F., Chen, Y., Cichocki, A., and Sejnowski, T. (2017). The effects of audiovisual inputs on solving the cocktail party problem in the human brain: an fMRI study. *Cereb. Cortex* 28, 3623–3637. doi: 10.1093/cercor/bhx235

Liu, Q., Huang, Y., Hao, Y., Xu, J., and Xu, B. (2021). LiMuSE: Lightweight multi-modal speaker extraction. *arXiv [Preprint]*. arXiv: 2111.04063.

López-Serrano, P., Dittmar, C., Özer, Y., and Müller, M. (2019). NMF toolbox: music processing applications of nonnegative matrix factorization.

McDermott, J. H. (2009). The cocktail party problem. *Curr. Biol.* 19, R1024–R1027. doi: 10.1016/j.cub.2009.09.005

McDermott, J. H., Wrobleski, D., and Oxenham, A. J. (2011). Recovering sound sources from embedded repetition. *Proc. Natl. Acad. Sci. U.S.A.* 108, 1188–1193. doi: 10.1073/pnas.1004765108

McFee, B., Raffel, C., Liang, D., Ellis, D., McVicar, M., Battenberg, E., et al. (2015). "librosa: Audio and music signal analysis in Python," in *Proc. of the 14th Python in Science Conf. (SCIPY 2015)* (Austin), 18–24.

Mesgarani, N., and Chang, E. (2012). Selective cortical representation of attended speaker in multi-talker speech perception. *Nature* 485, 233–236. doi: 10.1038/nature11020

Middlebrooks, J. C., and Waters, M. F. (2020). Spatial mechanisms for segregation of competing sounds, and a breakdown in spatial hearing. *Front. Neurosci.* 14, 571095. 10.3389/fnins.2020.571095

Mika, D., Budzik, G., and Józwik, J. (2020). "ICA-based single channel source separation with time-frequency decomposition," in *2020 IEEE 7th International Workshop on Metrology for AeroSpace (MetroAeroSpace)* (Pisa), 238–243.

Narayan, R., Best, V., Ozmeral, E., McClaine, E., Dent, M., Shinn-Cunningham, B., et al. (2008). Cortical interference effects in the cocktail party problem. *Nat. Neurosci.* 10, 1601–1607. doi: 10.1038/nn2009

O'Sullivan, J., Power, A., Mesgarani, N., Rajaram, S., Foxe, J., Shinn-Cunningham, B., Slaney, M., et al. (2014). Attentional selection in a cocktail party environment can be decoded from single-trial EEG. *Cereb. Cortex* 25, 1697–1706. doi: 10.1093/cercor/bht355

Oxenham, A. J. (2018). How we hear: the perception and neural coding of sound. *Annu. Rev. Psychol.* 69, 27–50. doi: 10.1146/annurev-psych-122216-011635

Pehlevan, C., Mohan, S., and Chklovskii, D. B. (2017). Blind nonnegative source separation using biological neural networks. *Neural Comput.* 29, 2925–2954. doi: 10.1162/neco_a_01007

Philarmonia Orchestra Instruments. (2019). Available online at: https://philharmonia.co.uk/resources/instruments/

Popham, S., Boebinger, D., Ellis, D., Kawahara, H., and McDermott, J. (2018). Inharmonic speech reveals the role of harmonicity in the cocktail party problem. *Nat. Commun.* 9, 2122. doi: 10.1038/s41467-018-04551-8

Sagi, B., Nemat-Nasser, S. C., Kerr, R., Hayek, R., Downing, C., and Hecht-Nielsen, R. (2001). A biologically motivated solution to the cocktail party problem. *Neural Comput.* 13, 1575–1602. doi: 10.1162/089976601750265018

Santosh, K. S., and Bharathi, S. H. (2017). "Non-negative matrix factorization algorithms for blind source sepertion in speech recognition," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)* (Bangalore), 2242–2246.

Sawada, H., Ono, N., Kameoka, H., Kitamura, D., and Saruwatari, H. (2019). A review of blind source separation methods: two converging routes to ilrma originating from ICA and NMF. *APSIPA Trans. Signal Inform. Process.* 8, 1–14. doi: 10.1017/ATSIP.2019.5

Schmidt, A. K. D., and Römer, H. (2011). Solutions to the cocktail party problem in insects: selective filters, spatial release from masking and gain control in tropical crickets. *PLoS ONE* 6, e28593. doi: 10.1371/journal.pone.0028593

Sharma, J., Angelucci, A., and Sur, M. (2000). Induction of visual orientation modules in auditory cortex. *Nature* 404, 841–847. doi: 10.1038/35009043

Smaragdis, P., and Brown, J. (2003). "Non-negative matrix factorization for polyphonic music transcription," in *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics* (New Paltz, NY), 177–180.

Stevens, S. S., Volkmann, J., and Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *J. Acoust. Soc. Am.* 8, 185–190.

Teki, S., Chait, M., Kumar, S., Shamma, S., and Griffiths, T. D. (2013). Segregation of complex acoustic scenes based on temporal coherence. *eLife* 2, e00699. doi: 10.7554/eLife.00699.009

Thakur, C., Wang, R., Afshar, S., Hamilton, T., Tapson, J., Shamma, S., et al. (2015). Sound stream segregation: a neuromorphic approach to solve the "cocktail party problem" in real-time. *Front. Neurosci.* 9, 309. doi: 10.3389/fnins.2015.00309

Urbanczik, R., and Senn, W. (2014). Learning by the dendritic prediction of somatic spiking. *Neuron* 81, 521–528. doi: 10.1016/j.neuron.2013.11.030

White, L., and King, S. (2003). *The Eustace Speech Corpus*. Centre for Speech Technology Research, University of Edinburgh.

Wickens, T. D. (2002). *Elementary Signal Detection Theory*. New York, NY: Oxford University Press.

Woods, K. J. P., and McDermott, J. H. (2018). Schema learning for the cocktail party problem. *Proc. Natl. Acad. Sci. U.S.A.* 115, E3313–E3322. doi: 10.1073/pnas.1801614115

Xiang, J., Simon, J., and Elhilali, M. (2010). Competing streams at the cocktail party: exploring the mechanisms of attention and temporal integration. *J. Neurosci.* 30, 12084–12093. doi: 10.1523/JNEUROSCI.0827-10.2010

Yu, D. (2020). "Solving cocktail party problem–from single modality to multi-modality," in *Proc. 6th International Workshop on Speech Processing in Everyday Environments (CHiME 2020)* (Virtual workshop).

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Dellaferrera, Asabuki and Fukai. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

frontiers | Frontiers in Neuroscience

ORIGINAL RESEARCH
published: 26 May 2022
doi: 10.3389/fnins.2022.759900

A Little Energy Goes a Long Way: Build an Energy-Efficient, Accurate Spiking Neural Network From Convolutional Neural Network

Dengyu Wu[1]*, *Xinping Yi*[2] *and Xiaowei Huang*[1]

[1] *Department of Computer Science, University of Liverpool, Liverpool, United Kingdom,* [2] *Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool, United Kingdom*

This article conforms to a recent trend of developing an energy-efficient Spiking Neural Network (SNN), which takes advantage of the sophisticated training regime of Convolutional Neural Network (CNN) and converts a well-trained CNN to an SNN. We observe that the existing CNN-to-SNN conversion algorithms may keep a certain amount of residual current in the spiking neurons in SNN, and the residual current may cause significant accuracy loss when inference time is short. To deal with this, we propose a unified framework to equalize the output of the convolutional or dense layer in CNN and the accumulated current in SNN, and maximally align the spiking rate of a neuron with its corresponding charge. This framework enables us to design a novel explicit current control (ECC) method for the CNN-to-SNN conversion which considers multiple objectives at the same time during the conversion, including accuracy, latency, and energy efficiency. We conduct an extensive set of experiments on different neural network architectures, e.g., VGG, ResNet, and DenseNet, to evaluate the resulting SNNs. The benchmark datasets include not only the image datasets such as CIFAR-10/100 and ImageNet but also the Dynamic Vision Sensor (DVS) image datasets such as DVS-CIFAR-10. The experimental results show the superior performance of our ECC method over the state-of-the-art.

Keywords: spiking neural network (SNN), spiking network conversion, deep learning, deep neural networks (DNNs), event-driven neural network

OPEN ACCESS

Edited by:
Angeliki Pantazi,
IBM Research-Zurich, Switzerland

Reviewed by:
Maryam Parsa,
George Mason University,
United States
Jim Harkin,
Ulster University, United Kingdom

****Correspondence:***
Dengyu Wu
dengyu.wu@liverpool.ac.uk

Specialty section:
This article was submitted to Frontiers in Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *17 August 2021*
Accepted: *28 February 2022*
Published: *26 May 2022*

Citation:
Wu D, Yi X and Huang X (2022) A Little Energy Goes a Long Way: Build an Energy-Efficient, Accurate Spiking Neural Network From Convolutional Neural Network. Front. Neurosci. 16:759900. doi: 10.3389/fnins.2022.759900

1. INTRODUCTION

Spiking neural networks (SNNs) are more energy efficient than convolutional neural networks (CNNs) in inference time because they utilize matrix addition instead of multiplication. SNNs are supported by new computing paradigms and hardware. For example, SpiNNaker (Painkras et al., 2012), a neuromorphic computing platform based on SNNs, can run real-time billions of neurons to simulate the human brain. The neuromorphic chips, such as TrueNorth (Akopyan et al., 2015), Loihi (Davies et al., 2018), and Tianji (Pei et al., 2019), can directly implement SNNs with 10,000 neurons being integrated onto a single chip. Moreover, through the combination with sensors, SNNs can be applied to edge computing, robotics, and other fields, to build low-power intelligent systems (Pfeiffer and Pfeil, 2018).

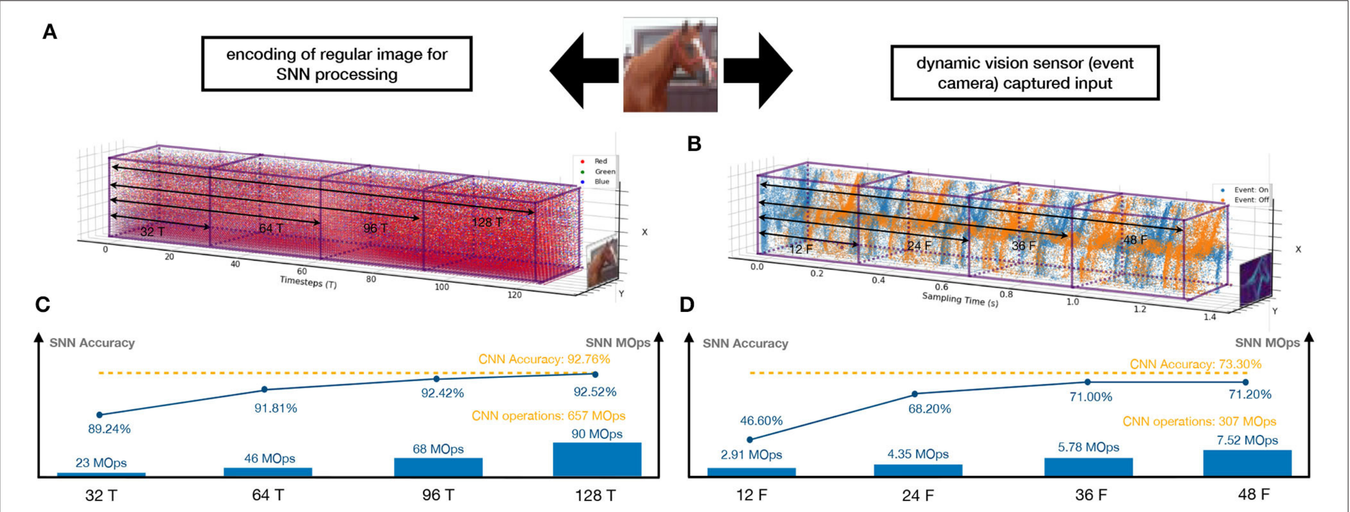


FIGURE 1 | An illustrative diagram showing how SNNs process two different types of inputs and their performance comparison with CNNs. A regular image (left column)—taken from the camera—is preprocessed into a spike train **(A)**, which then runs through the SNN in several timesteps (e.g., 128 timesteps as in the figure). A DVS input—taken from the event camera—can be represented directly as a spike train **(B)**, and processed naturally by the SNN in several frames (e.g., 48 frames as in the figure). **(C,D)** show the SNN's performance with respect to the three objectives (accuracy, energy efficiency, and latency), compared to CNNs.

However, the discrete nature of spikes makes the training of SNNs hard, due to the absence of gradients. This article follows a cutting-edge approach to obtaining a well-performed SNN by converting from a trained CNN of the same structure. This approach has an obvious benefit from the sophisticated training regime of CNNs, i.e., it is able to take advantage of the successful—and still fast improving—training methods on CNNs without extra efforts to adapt them to SNNs. Unfortunately, existing CNN-to-SNN conversion methods either cannot achieve a sufficiently small accuracy loss upon conversion (Rueckauer et al., 2017; Sengupta et al., 2019), or need a high latency (Sengupta et al., 2019), or require a significant increase in the energy consumption of the resulting SNNs (Han et al., 2020). Moreover, recent methods such as Han et al. (2020) do not work with the batch-normalization layer—a functional layer that plays a key role in the training of CNNs (Santurkar et al., 2018).

This article levels up the CNN-to-SNN conversion with the following contributions. First of all, methodologically, we argue that the conversion needs to be multi-objective—in addition to accuracy loss, energy efficiency and latency should be considered altogether. **Figure 1** provides an illustration showing how SNNs process images and DVS inputs, exhibiting how well our methods enable the achievement of the three objectives and its comparison with CNNs. Actually, **Figures 1C,D**, SNNs can have competitive accuracy upon conversion (92.52 vs. 92.76% and 71.20 vs. 73.30%, respectively) and be significantly more energy efficient than CNNs (90 vs. 657 MOps and 7.52 vs. 307 MOps, respectively). While it is hard to compare the latency as SNNs and CNNs work on different settings, our method implements high energy efficiency with low latency (128 timesteps for images and 48 frames for DVS inputs). As shown in our experiments, ours are superior to the state-of-the-art conversions (Rueckauer et al., 2017; Sengupta et al., 2019; Han et al., 2020).

Second, we follow an intuitive view aiming to establish an equivalence between the activations in an original CNN and the current in the resulting SNN. This view inspires us to consider an explicit, and detailed, control of the current flowing through the SNN. Technically, we develop a unifying theoretical framework, which treats both weight normalization (Rueckauer et al., 2017) and threshold balancing (Sengupta et al., 2019) as special cases. Based on the framework, we develop a novel conversion method called explicit current control (ECC), which includes two techniques: current normalization (CN), to control the maximum number of spikes fed into the SNN, and thresholding for residual elimination (TRE), to reduce the residual membranes potential in the neurons.

Third, we include in ECC a dedicated technique called consistency maintenance for batch-normalization (CMB) to deal with the conversion of the batch-normalization layer.

Finally, we implement ECC into a tool SpKeras[1] and conduct an extensive set of experiments on not only the regular image datasets, such as CIFAR-10/100 and ImageNet but also the Dynamic Vision Sensor (DVS) datasets such as DVS-CIFAR-10. Note that, DVS datasets are dedicated to SNN processing. The experimental results show that compared with state-of-the-art methods (Rueckauer et al., 2017; Sengupta et al., 2019; Han et al., 2020), ECC can optimize over three objectives at the same time, and have superior performance. Moreover, we notice that (1) ECC can utilize the conversion of batch-normalization to reduce the latency, and (2) ECC is robust to the hardware deployment because the quantisation—by using 7–10 bits to represent the originally 32-bit weights—does not lead to significant accuracy loss.

[1]https://github.com/Dengyu-Wu/spkeras

We remark that this article is not to argue for the replacement of CNNs with SNNs in general. Instead, we suggest a plausible deployment workflow, i.e., train a CNN → convert into an SNN → deploy on edge devices with e.g., an event camera. The workflow will not be a good option if any of the three objectives is not optimized.

2. RELATED STUDY

2.1. Current "Energy for Accuracy" Trend in CNN-to-SNN Conversion

A few different conversion methods, such as Diehl et al. (2015), Rueckauer et al. (2017), Sengupta et al. (2019), and Han et al. (2020), have been proposed in the past few years. It is not surprising that there is an accuracy loss between SNNs and CNNs. For example, in Rueckauer et al. (2017) and Sengupta et al. (2019), the gap is between 0.15 and 2% for CIFAR10 networks. A recent study by Sengupta et al. (2019) shows that this gap can be reduced if we use a sufficiently long (e.g., 1024 timesteps) spike train to encode an input. However, a longer spike train will inevitably lead to higher latency. This situation was believed to be eased by Han et al. (2020), which claim that the length of the spike train can be drastically shortened in order to achieve near-zero accuracy loss. However, as shown in Section 4.2 (**Figure 3A**), their threshold scaling method can easily lead to a significant increase in the spike-caused synaptic operations (Rueckauer et al., 2017), or spike operations for short, which also lead to a significant increase in the energy consumption.

Other related studies include (Rathi et al., 2020; Li et al., 2021; Rathi and Roy, 2021), which calibrate SNN to a specific timestep by gradient-based optimization. The calibration requires extra training time to find the optimal weights or hyper-parameters, such as some thresholds. In contrast, Deng and Gu (2021) trained a dedicated CNN for an SNN with fixed timesteps by shifting and clipping ReLU activations, although the accuracy loss of these SNNs cannot converge to zero when increasing the timesteps, as shown in **Figure 6B**. Besides, instead of reducing spikes, Lu and Sengupta (2020) explored SNN with binary weights to further improve the energy efficiency by consuming less memory.

2.2. Technical Ingredients in CNN-to-SNN Conversion

Table 1 provides an overview of the existing conversion methods (Cao et al., 2015; Diehl et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019; Han et al., 2020) and ours, from the aspects of technical ingredients and workable layers. In the beginning, most of the techniques, such as Cao et al. (2015) and Diehl et al. (2015), are based on hard reset (HR) spiking neurons, which are reset to fixed reset potential once their membrane potential exceeds the firing threshold. HR is still used in some recent methods such as Sengupta et al. (2019). The main criticism of HR is its significant information loss during the SNN inference. Soft reset (SR) neurons are shown better in other studies such as Rueckauer et al. (2017) and Han et al. (2020).

Weight normalization (WN) is proposed in Diehl et al. (2015) and extended in Rueckauer et al. (2017) to regulate the spiking rate in order to reduce accuracy loss. The other technique, threshold balancing (TB), is proposed by Sengupta et al. (2019) and extended by Han et al. (2020), to assign appropriate thresholds to the spiking neurons to ensure that they operate in the linear (or almost linear) regime. We show in Section 3.2 that both WN and TB are special cases of our theoretical framework.

TABLE 1 | Comparison of key technical ingredients (HR, SR, WN, TB, TS, ECC) and workable layers (BN, MP, AP) with the state-of-the-art methods.

	HR	SR	WN*	TB*	TS	ECC	BN**	MP	AP
Cao et al. (2015)	✓								✓
Diehl et al. (2015)	✓		✓						✓
Rueckauer et al. (2017)		✓	✓				✓	✓	
Sengupta et al. (2019)	✓			✓					✓
Han et al. (2020)		✓		✓	✓				✓
[This paper]		✓	✓	✓		✓	✓		✓

*HR, hard reset; SR, reset by subtraction, or soft reset; WN, weight normalization; TB, threshold balancing; TS, threshold scaling; ECC, explicit current control; BN, batch normalization; MP, max pooling; AP, average pooling. *As a contribution to this article, in Section 3.2, we show that both WN and TB are special cases of our ECC framework. **Among all methods, only those that can handle BN have bias terms in their pre-trained CNNs.*

Another technique called threshold scaling (TS) is suggested by Han et al. (2020). However, as our experimental result is shown in **Figure 3A**, TS leads to significantly greater energy consumption (measured as MOps). On the other hand, our ECC method can achieve smaller accuracy loss and significantly less energy consumption.

We also note in **Table 1** the differences in terms of workable layers in CNNs/SNNs for different methods. For example, the batch-normalization (BN) layer (Ioffe and Szegedy, 2015) is known as important for the optimization of CNNs (Santurkar et al., 2018), but only one existing method, i.e., Rueckauer et al. (2017), can work with it. Similarly, the bias values of neurons are pervasive for CNNs. Actually, the consideration of BN is argued in Sengupta et al. (2019) as the key reason for the higher accuracy loss in Rueckauer et al. (2017). The results of this article show that we can keep both BN and bias without significantly increased energy consumption, by maintaining the consistency between the behavior of SNN and CNN. BN can help with the reduction of latency. As we discussed earlier and in Section 5, our ECC method may be applicable to B-SNN and further improve its performance. Moreover, we follow most SNN research to consider the average pooling (AP) layer instead of the max pooling (MP) layer.

2.3. Direct Training

Spiking Neural Networks process information through non-differentiable spikes, and thus the backpropagation (BP) (LeCun et al., 1989) training algorithm cannot be directly applied. Few attempts by Lee et al. (2016) and Lee et al. (2020) have been made to adapt the BP algorithm by approximating its forward propagation phase. Such direct training requires high computational complexity to achieve an accuracy that is close to CNNs (Wu et al., 2021). Unlike these methods which approximate the BP algorithm (Lee et al., 2016, 2020), both of which may lead to performance degradation, we choose CNN-to-SNN conversion which can take full advantage of the continuously improving CNN training methods. Other than these methods which try to reproduce the success of CNN training, there are other direct training methods, such as approaches based on reservoir computing (Soures and Kudithipudi, 2019) and evolutionary algorithms (Schuman et al., 2020).

3. EXPLICIT CURRENT CONTROL

By leveraging the correspondence between activation in CNNs and current in SNNs,[2] we propose a unifying theoretical framework targeting multiple objectives, including accuracy, latency, and energy efficiency. Going beyond the existing conversion techniques (refer to **Table 1**) that consider some of the objectives individually, we view these multi-objective holistically through the lens of the unifying theoretical framework. Inspired by such a new viewpoint, we develop explicit current control (ECC) techniques to normalize, clip, and maintain the current through the SNNs for the purposes of reducing accuracy loss, latency, and energy consumption.

3.1. Existing CNN-to-SNN Conversion

Without loss of generality, we consider a CNN model of N layers such that layer n has M^n neurons, for $n \in \{1, 2, \ldots, N\}$. The output of the neuron $i \in \{1, \ldots, M^n\}$ at layer n with ReLU activation function is given by

$$a_i^n = \max\left\{0, \sum_{j=1}^{M^{n-1}} W_{ij}^n a_j^{n-1} + b_i^n\right\} \tag{1}$$

where W_{ij}^n is the weight between the neuron j at layer $n-1$ and the neuron i at layer n, b_i^n is the bias of the neuron i at layer n, and a_i^0 is initialized as the input x_i.

The activation a_i^n indicates the contribution of the neuron to the CNN inference. For CNN-to-SNN conversion, the greater a_i^n is, the higher the spiking rate will be, for the corresponding neuron on SNN. An explanation of a conversion method from CNNs to SNNs was first introduced by Rueckauer et al. (2017) by using data-based weight normalization.

The conversion method uses integrated-and-fire (IF) neurons to construct a rate-based SNN without leak and refractory time. If considering practical implementations, the rate-based SNN expects a relatively large interval between input spikes to minimize the effect of refractory time. To convert from a CNN, the spiking rate of each neuron in SNN is related to the activation of its corresponding neuron in the CNN. An iterative algorithm based on the *reset by subtraction* mechanism is described below. The membrane potential $V_i^n(t)$ of the neuron i at the layer n can be described as

$$V_i^n(t) = V_i^n(t-1) + Z_i^n(t) - \Theta_i^n(t) V_{thr}^n \tag{2}$$

where V_{thr}^n represents the threshold value of layer n and $Z_i^n(t)$ is the input current to neuron i at layer n such that

$$Z_i^n(t) = \sum_{j=1}^{M^{n-1}} W_{ij}^n \Theta_j^{n-1}(t) + b_i^n \tag{3}$$

with $\Theta_i^n(t)$ being a step function defined as

$$\Theta_i^n(t) = \begin{cases} 1, & \text{if } V_i^n(t) \geq V_{thr}^n \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

In particular, when the current $V_i^n(t)$ reaches the threshold V_{thr}^n, the neuron i at layer n will generate a spike, indicated by the step function $\Theta_i^n(t)$, and the membrane potential $V_i^n(t)$ will be reset immediately for the next timestep by subtracting the threshold.

3.2. A Unifying Theoretical Framework

The above CNN-to-SNN conversion method is designed specifically for weight normalization (Rueckauer et al., 2017), and cannot accommodate other conversion methods, e.g., threshold balancing (Sengupta et al., 2019). We propose a novel theoretical framework for CNN-to-SNN conversion that covers both weight normalization (Rueckauer et al., 2017) and threshold balancing (Sengupta et al., 2019) as special cases. In particular, the proposed framework improves over (Rueckauer et al., 2017) by adopting a thresholding mechanism to quantify the accumulated current into spikes in SNN and extends the threshold balancing mechanism to be compatible with batch normalization and bias.

We will work with the spiking rate of each SNN neuron i at layer n, defined as $r_i^n(t) = N_i^n(t)/t$, where $N_i^n(t)$ is the number of spikes generates in the first t timesteps by neuron i at layer n. We remark that it is possible that $r_i^n(t) > 1$, i.e., multiple spikes in a single timestep, in which case the latency is increased to process extra spikes.

Our framework is underpinned by Proposition 1.

Proposition 1. *In the CNN-to-SNN conversion, if the first layer CNN activation a_i^1 and the first layer SNN current $Z_i^1(t)$ satisfy the following condition*

$$\frac{1}{T}\sum_{t=1}^{T} Z_i^1(t) = a_i^1, \tag{5}$$

where T is a predefined maximum timestep, then the SNN spiking rate at time step t can be iteratively computed by

$$r_i^n(t) = \frac{1}{V_{thr}^n}\Big(\sum_{j=1}^{M^{n-1}} W_{ij}^n r_j^{n-1}(t) + b_i^n\Big) - \Delta_i^n(t) \tag{6}$$

with $\Delta_i^n(t) \triangleq V_i^n(t)/(tV_{thr}^n)$ representing the residual spiking rate. Initially, the spiking rate of neuron i at the first layer is $r_i^1(t) = a_i^1/V_{thr}^1 - \Delta_i^1(t)$.

[2]The activation values in the original CNNs can be represented by the current through the analog/digital circuits in the resulting SNNs, so that controlling current through spike train in SNNs corresponds to data flow operations in CNNs.

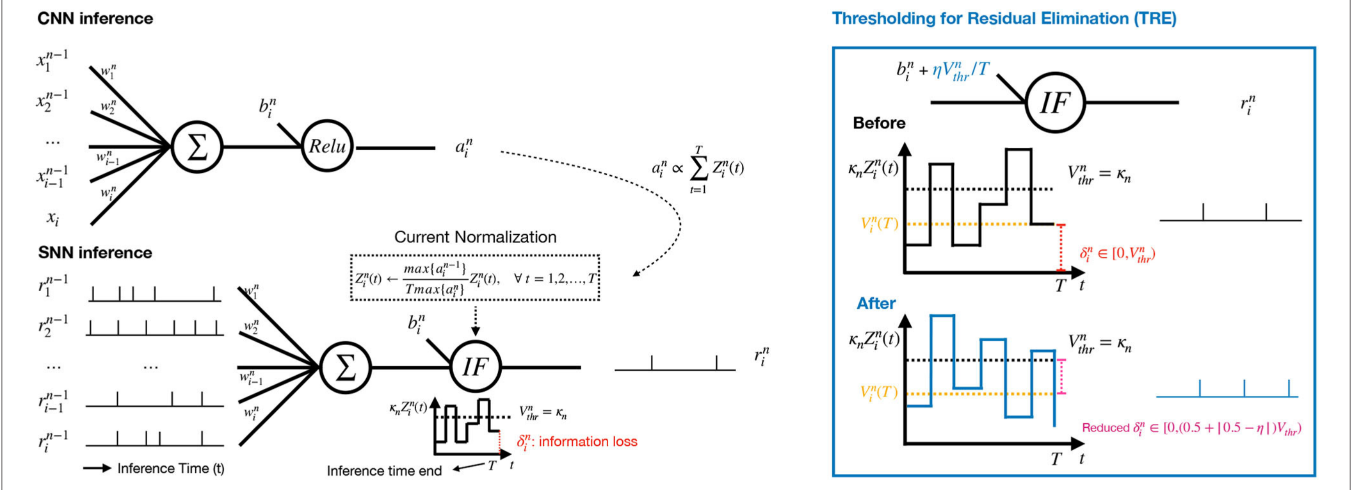

CNN inference
SNN inference
Current Normalization
Thresholding for Residual Elimination (TRE)
Before
After
Inference Time (t)
Inference time end
δ_i^n: information loss
$\delta_i^n \in [0, V_{thr}^n)$
Reduced $\delta_i^n \in [0, (0.5 + |0.5 - \eta|) V_{thr})$
$V_{thr}^n = \kappa_n$
$a_i^n \propto \sum_{t=1}^{T} Z_i^n(t)$
$b_i^n + \eta V_{thr}^n / T$

FIGURE 2 | Left: Our proposed CNN-to-SNN conversion for the n-th layer with a current normalization component and a thresholding mechanism. The activation a_i^n in the CNN (Top) is used for current normalization in the SNN (Bottom). **Right**: The proposed Thresholding for Residual Elimination (TRE) and the illustration of error reduction by TRE.

Remark 1.: The spiking rate in Equation (6) is a generalised form of those using weight normalization (WN) (Rueckauer et al., 2017) and threshold balancing (TB) (Sengupta et al., 2019). When keeping $V_{thr}^1 = 1$, by normalizing W_{ij}^n we obtain WN; when keeping W_{ij}^n unchanged, by normalizing V_{thr}^n we obtain TB. When applying a scaling factor α^n to the threshold V_{thr}^n, Proposition 1 recovers (Han et al., 2020).

The condition in Equation (5) bridges between the activations in CNNs and the accumulated currents in SNNs, i.e., *within the duration of a spike train, the average accumulated current equals the CNN activation*. This is key to our theoretical framework, and different from some previous conversion methods such as Rueckauer et al. (2017), which bridges between activations and firing rates. This *activation-current association* is reasonable because it aligns with the intuitions that (i) given a fixed spiking rate, a greater CNN activation requires a greater accumulated current in the SNN; and (ii) given a pre-trained CNN, more input spikes lead to increased current in the SNN.

Proposition 1 suggests that an explicit, optimized control on the currents may bring benefits to the spiking rate (so as to reduce energy consumption) and the residual current (so as to reduce the accuracy loss) simultaneously. First, a normalization of the currents $Z_i^n(t)$ is able to control the spike number, with its details being given in Section 3.3.1. Second, the error term $\Delta_i^n(t)$ will accumulate in deeper layers, causing a lower spiking rate in the output layer (Rueckauer et al., 2017). The thresholding technique in Section 3.3.2 will be able to reduce the impact of such an error. Third, we need to maintain the consistency between CNN and SNN so that the above control can be effective, as in Section 3.3.3.

The input is encoded into a spike train *via* Poisson event-generation process (Sengupta et al., 2019) or interpreting the input as constant currents (Rueckauer et al., 2017). In this article, we select the latter.

3.3. ECC-Based Conversion Techniques

We develop three ECC-based techniques, including current normalization (CN), thresholding for residual elimination (TRE), and consistency maintenance for batch-normalization (CMB). **Figure 2** illustrates CN and TRE, where the n-th layer of CNN is on the top and the corresponding conversed SNN layer is at the bottom. In the converted SNN layer, the sequences of spikes from the previous layer are aggregated, from which the current $Z_i^n(t)$ is accumulated in the neurons, and normalized by a factor (refer to Equation (7) below) to ensure that the increase of current at each timestep is within the range of [0,1]. The membrane potential $V_i^{n(t)}$ is produced according to Equation (2), followed by a spike generating operation as in Equation (4) once $V_i^n(t)$ exceeds the threshold $V_{thr}^n = \kappa_n$. The parameter κ_n is the current amplification factor, which will be explained in Section 3.3.1. The residual current Δ_i^n at the end of the spike train indicates the information loss in SNNs.

3.3.1. Current Normalization

At layer n, before spike generation, CN normalizes the current $Z_i^n(t)$ by letting

$$Z_i^n(t) \leftarrow \frac{\lambda_{n-1}}{T\lambda_n} Z_i^n(t), \quad \forall\, t = 1, \ldots, T \tag{7}$$

where $\lambda_n \triangleq \max_i\{a_i^n\}$ for $n = 1, 2, \ldots, N$. We have $\lambda_0 = 1$ when the input has been normalized into $[0, 1]$ for every feature. The benefit of CN is 2-fold:

- By CN, the maximum number of spikes fed into the SNN is under control, i.e., we can have direct control of the energy consumption.
- It facilitates the use of a positive integer $V_{thr}^n = \kappa_n$ as the threshold to quantify the current, which is amplified by a factor of κ_n, for spike generation. In doing so, the neuron with maximum current can generate a spike at every time step.

We randomly choose $\kappa_n = 100$ for V_{thr}^n and normalize weights for all experiments, except quantised SNN. Since the scalar of quantized weights in each layer will be absorbed into the threshold, we will get a different threshold for each layer. The quantisation process is explained in Section 4.4.

To achieve CN, the following conversion can be implemented to normalize weights and bias as follows.

$$W_{ij}^n \leftarrow \kappa_n \frac{\lambda_{n-1}}{\lambda_n} W_{ij}^n,\ b_i^n \leftarrow \frac{\kappa_n b_i^n}{\lambda_n},\ V_{thr}^n \leftarrow \kappa_n. \tag{8}$$

Note that, the next layer will amplify the incoming current back to its original scale before its normalization. When $\kappa_n = 1$ *or* λ_n/λ_{n-1}, the conversions correspond to weight normalization in Rueckauer et al. (2017) and threshold balancing in Sengupta et al. (2019), respectively.

3.3.2. Thresholding for Residual Elimination

According to Equation (6), the error increment after conversion is mainly caused by the residual information, $\delta_i^n(T) \in [0, V_{thr}^n]$, which remains with each neuron after T timesteps and cannot be forwarded to higher layers. To mitigate such errors, we propose a technique TRE to keep $\delta_i^n(T)$ under a certain value (half of V_{thr}^n as in our experiments). In particular, we add extra current to each neuron in order to have ηV_{thr}^n increment on each membrane potential, where $\eta \in [0, 1)$. Specifically, we update the bias term b_i^n of neuron i at layer n as follows

$$b_i^n(t) := b_i^n(t) + \eta V_{thr}^n / T \tag{9}$$

for every timestep t. Intuitively, we slightly increase synaptic bias for every neuron at every step so that a small volume of current is pumped into the system continuously.

The following proposition says that this TRE technique will be able to achieve a reduction of error range, which directly lead to the improvement in the accuracy loss.

Proposition 2. *Applying TRE will lead to*

$$\Theta_i^n(T) = \begin{cases} 1, & \textit{if } V_i^n(T) > (1-\eta)V_{thr}^n \\ 0, & \textit{otherwise.} \end{cases} \tag{10}$$

for timestep T as opposed to Equation (4). By achieving this, the possible range of errors is reduced from $[0, V_{thr}^n)$ *to* $[0, (0.5 + |0.5 - \eta|)V_{thr}^n)$.

We remark that, deploying TRE will increase at most one spike per neuron at the first layer and continue to affect the spiking rate at higher layers. This is the reason why we have slightly more spike operations than Rueckauer et al. (2017), as shown in **Supplementary Figure 1C**. A typical value η is 0.5.

3.3.3. Consistency Maintenance for Batchnormalization

Batch normalization (BN) (Ioffe and Szegedy, 2015) accelerates the convergence of CNN training and improves the generalization performance. The role of BN is to normalize the output of its previous layer, which allows us to add the normalized information to weights and biases in the previous layer. We consider a conversion technique CMB to maintain the consistency between SNN and CNN in operating the BN layer, by requiring a constant for numerical stability ϵ, as follows.

$$\hat{W}_{ij}^n = \frac{\gamma_i^n}{\sqrt{\sigma_i^{n2} + \epsilon}} W_{ij}^n \tag{11}$$

$$\hat{b}_i^n = \frac{\gamma_i^n}{\sqrt{\sigma_i^{n2} + \epsilon}} (b_i^n - \mu_i^n) + \beta_i^n \tag{12}$$

where γ_i^n and β_i^n are two learned parameters, μ_i^n and σ_i^n are mean and variance. ϵ is platform dependent: for Tensorflow, it is default as 0.001, and for PyTorch, it is 0.00001. The conversion method in Rueckauer et al. (2017) does not consider ϵ, and we found through several experiments that a certain amount of accuracy loss can be observed consistently. **Figure 5** shows the capability of CMB in reducing the accuracy loss.

4. EXPERIMENT

We implement the ECC method and conduct an extensive set of experiments to validate it. We consider its comparison with the state-of-the-art CNN-to-SNN conversion methods on images and DVS inputs (Sections 4.2, 4.5, respectively), the demonstration of its working with batch-normalization (Section 4.3), its robustness with respect to hardware deployment (Section 4.4), and an ablation study (Section 4.6). Due to the space limit, we present a subset of the results—the **Supplementary Material** includes more experimental results. We fix $\kappa_n = 100$ and $\epsilon = 0.001$ throughout the experiments.

In this section, "2017-SNN" denotes the method proposed in Rueckauer et al. (2017). "RMP-SNN(0.8)" and "RMP-SNN(0.9)" denote the method in Han et al. (2020), with different parameters 0.8 or 0.9 as co-efficient to V_{thr}. 'ECC-SNN' is our method. We remark that it is shown in Han et al. (2020) that its conversion method outperforms that of Sengupta et al. (2019), so we only compare with Han et al. (2020). Moreover, we may write "Method@nT" to represent the specific 'Method' when considering the spike trains of length n. Note that, only the CNN model in **Figure 3A** was trained without bias and BN, in order to have a fair comparison with RMP-SNN techniques. Since BN layers play an important role in training a high performance CNN and have the benefit of lowering the latency (c.f. Section 4.3), we believe it is essential to include them in CNN training. Therefore, we do not compare with Han et al. (2020) (i.e., RMP-SNN) and Sengupta et al. (2019) in other experiments because they do not work with BN.

Before proceeding, we explain how to estimate energy consumption. For CNNs, it is estimated through the multiply-accumulate (MAC) operations

$$\textit{MAC operations for CNNs}: \sum_{n=1}^{N} (2f_{in}^n + 1)M^n \tag{13}$$

where f_{in}^n is the number of input connections of the n-th layer. The number of MAC operations is fixed when the architecture of

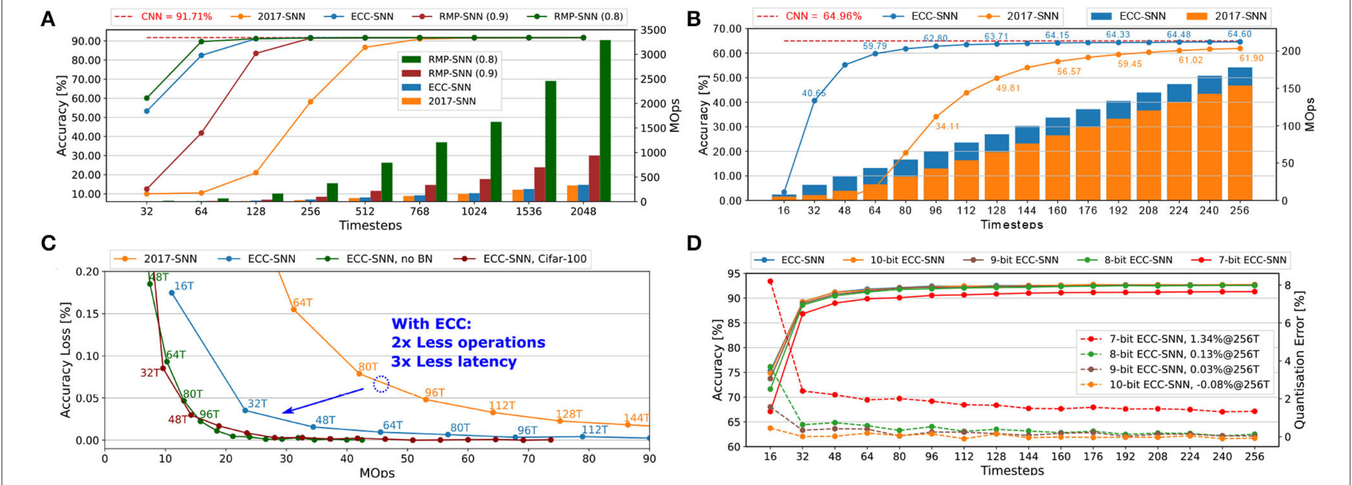


FIGURE 3 | (A) Accuracy and energy consumption (MOps) with respect to timesteps for CIFAR-10. **(B)** Accuracy and energy consumption (MOps) with respect to timesteps, for ImageNet (Top-1 Acc). **(C)** Accuracy loss and latency with respect to energy consumption (MOps), for CIFAR-10 and CIFAR-100. **(D)** Accuracy and quantisation error with respect to timesteps, for CIFAR-10.

the network is determined. For SNNs, the synaptic operations are counted to estimate the energy consumption of SNNs (Merolla et al., 2014; Rueckauer et al., 2017) as follows.

$$\textit{Synaptic operations for SNNs}: \sum_{t=1}^{T} \sum_{n=1}^{N} f_{out}^{n} s^{n} \tag{14}$$

where f_{out}^{n} is the number of output connections and s^{n} is the average number of spikes per neuron, of the n-th layer.

4.1. Experimental Settings

We work with both image datasets [CIFAR-10/100 (Krizhevsky and Hinton, 2009) and ImageNet (Russakovsky et al., 2015)] and DVS datasets (CIFAR-10-DVS Li et al., 2017) on several architectures (Simonyan and Zisserman, 2014) (VGG-16, VGG-19, and VGG-7). All the experiments are conducted on a CentOS Linux machine with two 2080Ti GPUs and 11 GB memory.

4.2. Comparisons With State-of-the-Art

Figure 3A presents a comparison between 2017-SNN, RMP-SNN, and ECC-SNN on both accuracy and energy consumption with respect to the timesteps, on VGG-16 and CIFAR-10. We note that both RMP-SNN and ECC-SNN outperform 2017-SNN, in terms of the number of timesteps to reach near-zero accuracy loss. Furthermore, ECC-SNN is better than RMP-SNN(0.9) and competitive with RMP-SNN(0.8) in terms of reaching near-zero accuracy loss under certain latency. Specifically, both ECC-SNN and RMP-SNN(0.8) require 128 timesteps and RMP-SNN(0.9) requires 256 timesteps. Importantly, we note that both RMP-SNN(0.8) and RMP-SNN(0.9) consume much more energy, measured with MOps, than ECC-SNN. Actually, ECC-SNN does not consume significantly more energy than 2017-SNN.

Similar results can be extended to a large dataset such as ImageNet. Moreover, to investigate further into the energy consumption, **Figure 3B** presents a comparison with 2017-SNN. All the above results show that ECC-SNN significantly reduces the latency, easily reaches the near-zero loss, and costs a minor increase in energy.

The above results, together with those in **Supplementary Figures 1A,C,D,E**), reflect exactly the advantage of using ECC-SNN. That is RMP-SNN(0.8) and ECC-SNN are the best in achieving near-zero accuracy loss with low latency, but RMP-SNN requires significantly more energy than the other two methods. Therefore, *ECC-SNN achieves the best when considering energy, latency, and accuracy loss.*

Batch-normalization (BN) has become indispensable to train CNNs, so we believe a CNN-to-SNN method should be able to work with it. After demonstrating a clear advantage over RMP-SNN, for the rest of this section, we will focus on the comparison with 2017-SNN, which deals with BN. We trained CNNs using Tensorflow by having a batch-normalization layer after each convolutional layer.

4.3. Batch-Normalization

Figure 3C considers the impact of working with BN. Compared with 2017-SNN, ECC-SNN achieves similar accuracy loss by taking 2x less MOps and 3x less latency. Moreover, to achieve the same accuracy loss, ECC-SNN without BN, i.e., ECC applies on CNNs without BN layers, requires significantly more timesteps, with slightly less MOps. Moreover, our other experiments show that RMP-SNN (0.8), without BN in its method, can only achieve 48.32% in 256T. With BN, 2017-SNN can achieve 49.81% in 128T. ECC-SNN further improves on this, achieving 63.71% in 128T. That is, *batch-normalization under ECC-SNN can help reduce the latency*. This is somewhat surprising, and we believe further research is needed to investigate the formal link between BN and latency.

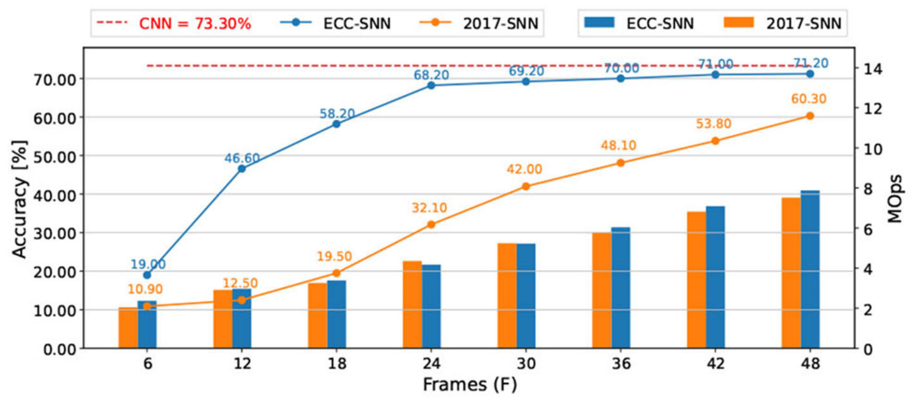


FIGURE 4 | Accuracy and energy consumption (MOps) with respect to frames, between 2017-SNN and ECC-SNN, for CIFAR-10-DVS, and VGG-7.

TABLE 2 | Comparison of SNN accuracy, latency, and energy consumption (MOps), between direct training, 2017-SNN and ECC-SNN, for Cifar-10-DVS.

Method	Publication	Accuracy	N_f^*	MOps
Direct training (VGG7)	Wu et al., 2021	62.50	-	-
2017-SNN (DenseNet)	Kugele et al., 2020	65.61	60	1,551
ECC-SNN (VGG16)	this paper	71.20	**48**	66.79
ECC-SNN (VGG7)	this paper	**71.30**	**48**	**7.52**

4.4. Robustness to Quantisation

Figure 3D and **Supplementary Figure 1B** present how the change in the number of bits to represent weights may affect the accuracy and the quantisation error. This is an important issue, as the SNNs will be deployed on the neuromorphic chip, such as Loihi (Davies et al., 2018) and TrueNorth (Akopyan et al., 2015), or FPGA, which may have different configurations. For example, Loihi can have weight precision at 1-9 bits. Floating-point data, both weights and threshold can be simply converted into fixed-point data after CN in two steps: normalizing the weights into the range [-1,1] and scaling the threshold using the same normalization factor, and then multiplied with 2^b, where b is the bit width (Ju et al., 2019; Sze et al., 2019). From **Figure 3D** and **Supplementary Figure 1B**, the reduction from 32-bit to 10-, 9-, 8-, and 7-bit signed weights does lead to a drop in the accuracy, but unless it goes to 7-bit, the accuracy loss is negligible. This shows that *our ECC method is robust to hardware deployments*.

4.5. DVS Dataset

CIFAR-10-DVS (Li et al., 2017) is a benchmark dataset of DVS inputs, consisting of 10,000 inputs extracted from the CIFAR-10 dataset using a DVS128 sensor. The resolution of data is 128x128. We preprocess the data following Wu et al. (2021) and Kugele et al. (2020), select the first 1.3 s of the event stream and down-scale the input into 42 x 42. For each dimension of input, we calculate the number of spikes over the 1.3 s simulation and normalize with a constant representing the maximum number of spikes. During SNN processing, as shown in **Figure 1B**, the latency is based on spikes. The experiments using VGG7 (**Figure 4**), VGG16, and ResNet-18 (Sengupta et al., 2019; **Supplementary Figures 2A,B**) show that ECC-SNN performs much better than 2017-SNN, who also work with BN layer. Moreover, in **Table 2**, we compare few methods including a recent direct training method and highlight the best results for each objective. We can see that ECC-SNN can always achieve better accuracy, less frames, and less energy consumption.

Moreover, we recall the results shown in **Figure 1** concerning the comparison between DVS inputs and images. For the same problem, if we choose the deployment workflow of "training a CNN → converting into an SNN → deploying on edge devices with e.g., event camera," we may consume 10+ times less energy (7.52 vs. 90 MOps for CIFAR-10) by taking DVS inputs. Both are in stark contrast with the other deployment workflow "training a CNN → deploying on edge devices with camera," which costs much more energy (307MOPs and 657MOps, respectively).

4.6. Ablation Study

To understand the contributions of the three ingredients of ECC-SNN, i.e., CN, CMB, and TRE, we conduct an experiment on VGG-16 and CIFAR-10, by gradually including technical ingredients to see their respective impact on the accuracy loss. **Figure 5** shows the histograms of the mean accuracy losses in 256T, over the 281–283th epochs. We see that every ingredient plays a role in reducing the accuracy loss, with the TRE and CN being lightly Moreover, We also consider the impact of η (as in **Supplementary Figure 1F**).

5. CONVERSION OPTIMIZED THROUGH DISTRIBUTION-AWARE CNN TRAINING

Up to now, all methods we discussed and compared, including our ECC method, are focused on optimizing the CNN-to-SNN conversion, without considering whether or not the CNN itself may also play a role in eventually obtaining a good-performing SNN. In this section, we will discuss several recent techniques that include the consideration of CNN training and show that our ECC method can also improve them by optimizing the CNN-to-SNN conversion.

As noted in Section 3.3.1 that the maximum value of activations is a key parameter in the conversion. Based on this, Rueckauer et al. (2017) and Lu and Sengupta (2020) suggest that a particular percentile from the histogram on the CNN activation may improve conversion efficiency. One step further, Yu et al. (2020) suggest that a good distribution with less outliers on CNN activation can be useful for quantisation. Therefore, we call these techniques distribution-aware CNN training techniques, to emphasise that they are mainly focused on optimizing the CNN training through enforcing good distributions on the activations.

To show that our ECC method is complementary to the distribution-aware CNN training techniques, we implement some existing CNN training techniques that can affect activation distribution and show that ECC can also work with them to achieve optimized conversion. Specifically, Yu et al. (2020) notice that the clipped ReLU can enforce small activation values (i.e., close to zero) to become greater, and eventually reshape the distribution from a Gaussian-like distribution to a uniform distribution. We follow this observation to train CNN models with different clipping methods, including ReLU6 (clipped by 6) from Lin et al. (2019) and Jacob et al. (2018), ReLU-CM (clipped by k-mean) from Yu et al. (2020), and ReLU-SC (shift and clipped) from Deng and Gu (2021). **Figure 6A** shows that all clipping methods can shift the original activation value (as in the top left figure) to values closer to the mean value (i.e., near the peak area). Such a shift of maximum activation value can significantly reduce the possibility for the maximum value to become an outlier.

Figure 6B presents a comparison between SNNs obtained through ReLU6, ReLU-CM and ReLU-SC, with and without the application of ECC. First of all, distribution-aware training can improve performance. For example, with clipping methods, the

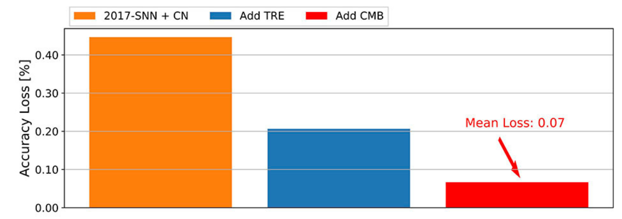


FIGURE 5 | Contribution of CN, CMB, and TRE to the reduction of mean accuracy loss, for CIFAR-10 and VGG-16.

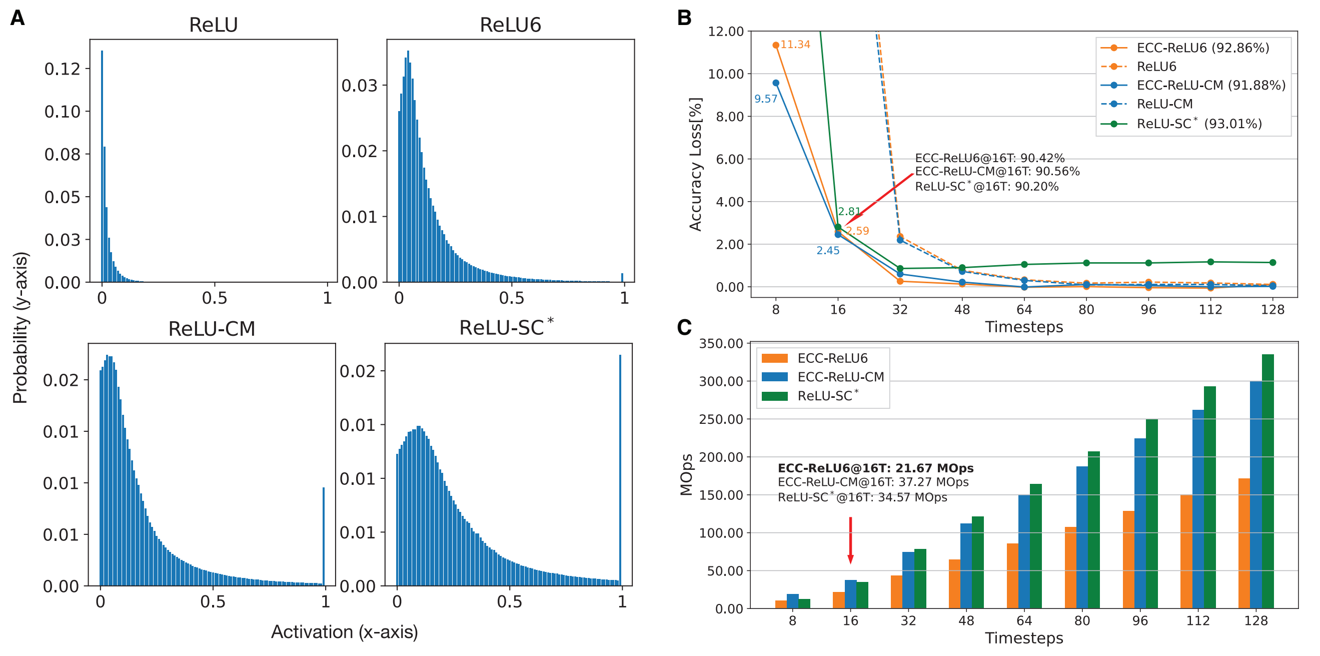


FIGURE 6 | Comparison of SNNs using different clipping methods, ReLU6 (Jacob et al., 2018; Lin et al., 2019), ReLU-CM (Yu et al., 2020), ReLU-SC* (Deng and Gu, 2021), for CIFAR-10 on VGG-16. **(A)** Normalized activation distribution of the first layer. **(B)** Accuracy with respect to timesteps. **(C)** Energy consumption (MOps) with respect to timesteps. *We use ReLU-SC to train an SNN with a fixed timestep (16T), as it does not need extra training.

accuracy loss is less than 2.2%@32T, which is better than ECC-SNN (3.2%@32T) as in **Supplementary Figure 1C**. Then, we can see that, with ECC, ReLU6 and ReLU-CM can achieve 2x less latency with little performance degradation. Although ECC-ReLU6, ECC-ReLU-CM, and ReLU-SC achieve similar accuracy (90.20–90.56%@16T), ECC-ReLU-CM has the best adaptability to different timesteps. By contrast, ECC-ReLU6 uses 1.5–1.7x less operations at 16T than ReLU-SC and ECC-ReLU-CM, as shown in **Figure 6C**. We only apply ECC to ReLU6 and ReLU-CM, as ReLU-SC in Deng and Gu (2021) is not designed to be adaptable to different timesteps.

The above results show that distribution-aware CNN training and our ECC method can both improve the CNN-to-SNN conversion. While distribution-aware CNN training can reduce the accuracy loss, the application of the ECC method can further improve the performance of the resulting SNN model. Furthermore, it is worth mentioning that ECC can take the advantage of the accumulated bias current to optimize a single SNN model with respect to different timesteps.

6. DISCUSSION

Variants to the Unifying Framework

The current unifying framework (Section 3.2) considers the ReLU activation function, which exhibits a linear relation between accumulated current and spiking rate. There are other—arguably more natural—features in biological neurons, such as a leak, refractory time, and adaptive threshold, as discussed in Kobayashi et al. (2009). If considering these features, the relation between accumulated current and spiking rate will become non-linear. To deal with them, it can be an interesting future work to consider extending the unifying framework to address the connection between nonlinear activation functions (e.g., sigmoid) on CNN and the dynamic properties on SNN.

Hyper-Parameters in ECC

Most of the hyper-parameters in ECC-SNN are determined with reasons, such as κ_n (Section 3.3.1) and η (Section 3.3.2), while timesteps (T) are determined by practical application according to e.g., required accuracy. Although some gradient-based optimization methods, such as Rathi et al. (2020), Rathi and Roy (2021), and Li et al. (2021), can improve the SNN to a fixed timestep, ECC allows SNN to be adaptive to different timesteps. In the future, we will consider hyper-parameters tuning methods, e.g., Parsa et al. (2020), to further improve ECC-SNN while maintaining its adaptability.

7. CONCLUSION

We develop a unifying theoretical framework to analyze the conversion from CNNs to SNNs and a new conversion method ECC to explicitly control the currents, so as to optimize accuracy loss, energy efficiency, and latency simultaneously. By comparing state-of-the-art methods, we confirm the superior performance of our method. Moreover, we study the impact of batch-normalization and show the robustness of ECC over quantization.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

DW designed the study, contributed to the source code, conducted the experiments, and evaluated the results. XY and XH provided the feedback and scientific advice throughout the process. All authors contributed to the final manuscript.

FUNDING

DW is supported by the University of Liverpool and China Scholarship Council Awards (Grant No. 201908320488). This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 956123. It is also supported by the U.K. EPSRC (through End-to-End Conceptual Guarding of Neural Architectures [EP/T026995/1]).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2022.759900/full#supplementary-material

REFERENCES

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Deng, S., and Gu, S. (2021). "Optimal conversion of conventional artificial neural networks to spiking neural networks," in *International Conference on Learning Representations*.

Diehl, P., Neil, D., Binas, J., Cook, M., Liu, S., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *International Joint Conference on Neural Networks*, 1–8.

Han, B., Srinivasan, G., and Roy, K. (2020). "RMP-SNN: residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13558–13567.

Ioffe, S., and Szegedy, C. (2015). "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *volume 37 of Proceedings of Machine Learning Research* (Lille: PMLR), 448–456.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., et al. (2018). "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT: IEEE), 2704–2713.

Ju, X., Fang, B., Yan, R., Xu, X., and Tang, H. (2019). An fpga implementation of deep spiking neural networks for low-power and fast classification. *Neural Comput*. 32, 182–204. doi: 10.1162/neco_a_01245

Kobayashi, R., Tsubo, Y., and Shinomoto, S. (2009). Made-to-order spiking neuron model equipped with a multi-timescale adaptive threshold. *Front. Comput. Neurosci*. 3, 9. doi: 10.3389/neuro.10.009.2009

Krizhevsky, A., and Hinton, G. (2009). "Learning multiple layers of features from tiny images," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*.

Kugele, A., Pfeil, T., Pfeiffer, M., and Chicca, E. (2020). Efficient processing of spatio-temporal data streams with spiking neural networks. *Front. Neurosci*. 14, 439. doi: 10.3389/fnins.2020.00439

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Comput*. 4, 541–551. doi: 10.1162/neco.1989.1.4.541

Lee, C., Sarwar, S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci*. 14, 119. doi: 10.3389/fnins.2020.00119

Lee, J., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci*. 10, 508. doi: 10.3389/fnins.2016.00508

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Front. Neurosci*. 11, 309. doi: 10.3389/fnins.2017.00309

Li, Y., Deng, S., Dong, X., Gong, R., and Gu, S. (2021). "A free lunch from ann: towards efficient, accurate spiking neural networks calibration," in *Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research*, eds M. Meila, and T. Zhang (PMLR), 6316–6325.

Lin, J., Gan, C., and Han, S. (2019). Defensive quantization: When efficiency meets robustness. *arXiv preprint* arXiv:1904.08444. doi: 10.48550/arXiv.1904.08444

Lu, S., and Sengupta, A. (2020). Exploring the connection between binary and spiking neural networks. *Front. Neurosci*. 14, 535. doi: 10.3389/fnins.2020.00535

Merolla, P., Arthur, J., Alvarez-Icaza, R., Cassidy, A., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Painkras, E., Plana, L., Garside, J., Temple, S., Davidson, S., Pepper, J., et al. (2012). "Spinnaker: a multi-core system-on-chip for massively-parallel neural net simulation," in *NaIn Proceedings of the IEEE 2012 Custom Integrated Circuits Conference* (San Jose, CA: IEEE), 1–4.

Parsa, M., Mitchell, J. P., Schuman, C. D., Patton, R. M., Potok, T. E., and Roy, K. (2020). Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Front. Neurosci*. 14, 667. doi: 10.3389/fnins.2020.00667

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci*. 12, 774. doi: 10.3389/fnins.2018.00774

Rathi, N., and Roy, K. (2021). Diet-snn: a low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Trans. Neural Netw. Learn. Syst*. 1–9. doi: 10.1109/TNNLS.2021.3111897

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in *International Conference on Learning Representations*.

Rueckauer, B., Lungu, I., Hu, Y.,and Pfeiffer, M., and Liu, S. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci*. 11, 682. doi: 10.3389/fnins.2017.00682

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). ImageNet large scale visual recognition challenge. *Int. J. Comput. Vis*. 115, 211–252. doi: 10.1007/s11263-015-0816-y

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). "How does batch normalization help optimization?" in *NeurIPS*, 2488–2498.

Schuman, C. D., Mitchell, J. P., Patton, R. M., Potok, T. E., and Plank, J. S. (2020). "Evolutionary optimization for neuromorphic systems," in *Proceedings of the Neuro-inspired Computational Elements Workshop*, 1–9.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci*. 13, 95. doi: 10.3389/fnins.2019.00095

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*. doi: 10.48550/arXiv.1409.1556

Soures, N., and Kudithipudi, D. (2019). Spiking reservoir networks: brain-inspired recurrent algorithms that use random, fixed synaptic strengths. *IEEE Signal Process. Mag*. 36, 78–87. doi: 10.1109/MSP.2019.2931479

Sze, V., Chen, Y., Yang, T., and Emer, J. (2019). Efficient processing of deep neural networks: a tutorial and survey. *Proc. IEEE* 105, 2295–2329. doi: 10.1109/JPROC.2017.2761740

Wu, H., Zhang, Y., Weng, W., Zhang, Y., Xiong, Z., Zha, Z.-J., et al. (2021). "Training spiking neural networks with accumulated spiking flow," in *Proceedings of the AAAI Conference on Artificial Intelligence*.

Yu, H., Wen, T., Cheng, G., Sun, J., Han, Q., and Shi, J. (2020). "Low-bit quantization needs good distribution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (Seattle, WA: IEEE), 680–681.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Wu, Yi and Huang. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Advantages of publishing in Frontiers

OPEN ACCESS
Articles are free to read for greatest visibility and readership

FAST PUBLICATION
Around 90 days from submission to decision

HIGH QUALITY PEER-REVIEW
Rigorous, collaborative, and constructive peer-review

TRANSPARENT PEER-REVIEW
Editors and reviewers acknowledged by name on published articles

Frontiers
Avenue du Tribunal-Fédéral 34
1005 Lausanne | Switzerland

Visit us: www.frontiersin.org
Contact us: frontiersin.org/about/contact

REPRODUCIBILITY OF RESEARCH
Support open data and methods to enhance research reproducibility

DIGITAL PUBLISHING
Articles designed for optimal readership across devices

FOLLOW US
@frontiersin

IMPACT METRICS
Advanced article metrics track visibility across digital media

EXTENSIVE PROMOTION
Marketing and promotion of impactful research

LOOP RESEARCH NETWORK
Our network increases your article's readership