# TRAINING METHODOLOGIES FOR ENERGY-EFFICIENT, LOW LATENCY SPIKING NEURAL NETWORKS

by

**Nitin Rathi**

**A Dissertation**

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

**Doctor of Philosophy**

School of Electrical and Computer Engineering

West Lafayette, Indiana

December 2021

# THE PURDUE UNIVERSITY GRADUATE SCHOOL
# STATEMENT OF COMMITTEE APPROVAL

*Dedicated to my mother, my late grandmother (Maa), and Animita - three women who have had the most impact on my life*

# ACKNOWLEDGMENTS

To start with, I want to express my heartfelt gratitude to my advisor, Prof. Kaushik Roy, for his invaluable advice and guidance to help me become a better researcher. The time spent at his lab, Nanoelectronics Research Laboratory (NRL), has shaped my professional career as well as had a positive impact on the way I look at problems in general. He gave me the time and freedom to tackle difficult research problems and provided constructive feedback to help me achieve my goals. I will forever be thankful to him.

Next, I would like to thank my committee members Prof. Anand Raghunathan, Prof. Sumeet Gupta, and Prof. Vijay Raghunathan for assisting me in designing my research plan and helping me achieve my research goals.

I would also like to thank Dr. Debjani Roy from Bose Institute, Kolkata, India, for introducing me to the fascinating world of research and helping me understand the impact of the work done in research labs. The time spent in her lab was instrumental in my decision to pursue the doctoral journey.

I would like to thank my first mentor at NRL, Dr. Gopalakrishnan Srinivasan, who helped me get started and was always available to answer any questions. He shared his deep insights about the subject and helped me understand the tiniest details. I would like to thank all the wonderful people I met at NRL - Amogh, Indranil, Deboleena, Utkarsh, Sangamesh, Deepika, Adarsh, Tanvi, Timur, Mustafa, Eunseon that made the entire doctoral journey fun and exciting. I would also like to thank all my collaborators and fellow lab mates at NRL for their support.

Last, but not least, I would like to dedicate this doctoral journey to my parents and Animita. My parents have always given me the freedom to choose my career path and were supportive of my decision to do a PhD. Animita has always pushed me to do my best and celebrated all my small wins - my first paper, conference award, internship, job offers, etc. I would like to thank her for her unconditional love and support.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABSTRACT

Deep learning models have become the de-facto solution in various fields like computer vision, natural language processing, robotics, drug discovery, and many others. The skyrocketing performance and success of multi-layer neural networks comes at a significant power and energy cost. Thus, there is a need to rethink the current trajectory and explore different computing frameworks. One such option is spiking neural networks (SNNs) that is inspired from the spike-based processing observed in biological brains. SNNs operating with binary signals (or spikes), can potentially be an energy-efficient alternative to the power-hungry analog neural networks (ANNs) that operate on real-valued analog signals. The binary all-or-nothing spike-based communication in SNNs implemented on event-driven hardware offers a low-power alternative to ANNs. A spike is a Delta function with magnitude 1. With all its appeal for low power, training SNNs efficiently for high accuracy remains an active area of research. The existing ANN training methodologies when applied to SNNs, results in networks that have very high latency. Supervised training of SNNs with spikes is challenging (due to discontinuous gradients) and resource-intensive (time, compute, and memory).Thus, we propose compression methods, training methodologies, learning rules

First, we propose compression techniques for SNNs based on unsupervised spike timing dependent plasticity (STDP) model. We present a sparse SNN topology where non-critical connections are pruned to reduce the network size and the remaining critical synapses are weight quantized to accommodate for limited conductance levels in emerging in-memory computing hardware . Pruning is based on the power law weight-dependent STDP model; synapses between pre- and post-neuron with high spike correlation are retained, whereas synapses with low correlation or uncorrelated spiking activity are pruned. The process of pruning non-critical connections and quantizing the weights of critical synapses is performed at regular intervals during training.

Second, we propose a multimodal SNN that combines two modalities (image and audio). The two unimodal ensembles are connected with cross-modal connections and the entire network is trained with unsupervised learning. The network receives inputs in both modalities for the same class and predicts the class label. The excitatory connections in the unimodal

ensemble and the cross-modal connections are trained with STDP. The cross-modal connections capture the correlation between neurons of different modalities. The multimodal network learns features of both modalities and improves the classification accuracy compared to unimodal topology, even when one of the modality is distorted by noise. The cross-modal connections are only excitatory and do not inhibit the normal activity of the unimodal ensembles.

Third, we explore supervised learning methods for SNNs.Many works have shown that an SNN for inference can be formed by copying the weights from a trained ANN and setting the firing threshold for each layer as the maximum input received in that layer. These type of converted SNNs require a large number of time steps to achieve competitive accuracy which diminishes the energy savings. The number of time steps can be reduced by training SNNs with spike-based backpropagation from scratch, but that is computationally expensive and slow. To address these challenges, we present a computationally-efficient training technique for deep SNNs. We propose a hybrid training methodology: 1) take a converted SNN and use its weights and thresholds as an initialization step for spike-based backpropagation, and 2) perform incremental spike-timing dependent backpropagation (STDB) on this carefully initialized network to obtain an SNN that converges within few epochs and requires fewer time steps for input processing. STDB is performed with a novel surrogate gradient function defined using neuron's spike time. The weight update is proportional to the difference in spike timing between the current time step and the most recent time step the neuron generated an output spike.

Fourth, we present techniques to further reduce the inference latency in SNNs. SNNs suffer from high inference latency, resulting from inefficient input encoding, and sub-optimal settings of the neuron parameters (firing threshold, and membrane leak). We propose DIET-SNN, a low-latency deep spiking network that is trained with gradient descent to optimize the membrane leak and the firing threshold along with other network parameters (weights). The membrane leak and threshold for each layer of the SNN are optimized with end-to-end backpropagation to achieve competitive accuracy at reduced latency. The analog pixel values of an image are directly applied to the input layer of DIET-SNN without the need to convert to spike-train. The first convolutional layer is trained to convert inputs into spikes where leaky-integrate-and-fire (LIF) neurons integrate the weighted inputs and generate an output spike when the membrane potential crosses the trained firing threshold. The trained membrane leak controls the flow of input information and attenuates irrelevant inputs to increase the activation sparsity in the convolutional and dense layers of the network. The reduced latency combined with high activation sparsity provides large improvements in computational efficiency.

Finally, we explore the application of SNNs in sequential learning tasks. We propose LITE-SNN, a lightweight SNN suitable for sequential learning tasks on data from dynamic vision sensors (DVS) and natural language processing (NLP). In general sequential data is processed with complex recurrent neural networks (like long short-term memory (LSTM), and gated recurrent unit (GRU)) with explicit feedback connections and internal states to handle the long-term dependencies. Whereas neuron models in SNNs - integrate-and-fire (IF) or leaky-integrate-and-fire (LIF) - have implicit feedback in their internal state (membrane potential) by design and can be leveraged for sequential tasks. The membrane potential in the IF/LIF neuron integrates the incoming current and outputs an event (or spike) when the potential crosses a threshold value. Since SNNs compute with highly sparse spike-based spatio-temporal data, the energy/inference is lower than LSTMs/GRUs. SNNs also have fewer parameters than LSTM/GRU resulting in smaller models and faster inference. We observe the problem of vanishing gradients in vanilla SNNs for longer sequences and implement a convolutional SNN with attention layers to perform sequence-to-sequence learning tasks. The inherent recurrence in SNNs, in addition to the fully parallelized convolutional operations, provides an additional mechanism to model sequential dependencies and leads to better accuracy than convolutional neural networks with ReLU activations.

# 1. INTRODUCTION

Building machines having brain-like capabilities has been a persistent dream of computer scientists. With recent advancements in artificial intelligence, especially deep artificial neural networks (ANNs), today's computers achieve superhuman performance in several cognitive tasks − for example *AlphaGo* beating the *Go* master in 2016. However, the unprecedented success of deep ANNs is accompanied by significant power and energy cost [1], [2]. While the human brain's energy budget is considered ∼20W, including simultaneous recognition, reasoning and control [3], conventional computing systems consume an order of magnitude higher power for classification alone. This remarkable ability of the human brain has lead researchers to seek inspiration from neuroscience for building neuromorphic computing systems [4] that can operate with brain-like efficiency, especially for battery-operated and resource-constrained edge devices. Among other things, the efficiency of biological systems is attributed to highly parallel, sparse, and event-driven computations. Spiking neural networks (SNNs) operating on sparse binary signals ('spikes') in an event-driven manner implemented on neuromorphic hardware can potentially address the energy problem in ANNs. The spike-based computation in SNNs replaces the energy intensive multiply-and-accumulate operation (required in ANNs) with simple additions. With all its appeal for energy efficiency, the success of SNNs was delayed due to the lack of accurate training methods. In recent years, SNNs trained with conversion and gradient-based learning algorithms have achieved similar inference accuracy as ANNs for complex image classification tasks [5], [6].

While the event-driven nature of SNNs offers a promising route for achieving lower energy and power consumption for intelligent hardware, it also poses a critical limitation on their learning capability. Integrating temporally-encoded statistics of spiking neurons/synapses with standard gradient-descent based learning algorithms (catered for ANNs which do not encode information in time) presents several challenges [7]. It is difficult to train the layers of a deep SNN architecture globally in an end-to-end manner [8]. Bio-plausible unsupervised [9] and supervised [10] learning have been explored as a viable solution which allows localized learning, and has proven to be computationally more efficient than backpropagation based algorithms. There has also been a significant thrust towards developing scalable gradient based algorithms which can be adapted to event-driven, sparse activity in SNNs [11], [12]. Overall, these algorithmic drives promises to scale up the performance of SNNs to levels currently offered by ANNs, while preserving the benefits of sparse event-based computations.

# 2. STDP BASED PRUNING OF CONNECTIONS AND WEIGHT QUANTIZATION IN SPIKING NEURAL NETWORKS FOR ENERGY-EFFICIENT RECOGNITION

## 2.1 Introduction

Human brain consisting of 20 billion neurons and 200 trillion synapses is by far the most energy-efficient neuromorphic system with cognitive intelligence. The human brain consumes only ∼20W of power which is nine orders of magnitude lower compared to a computer simulating human brain activity in real time [13]. This had led to the inspiration and development of Spiking Neural Networks (SNNs) which tries to mimic the behavior of human brain and process inputs in real time [14]. SNNs may provide an energy-efficient solution to perform neural computing. However, recent works have shown that to get reasonable accuracy compared to non-spiking Artificial Neural Networks (nANNs), the complexity and size of SNNs is enormous. In [15] to improve the classification accuracy for MNIST dataset by 12%, the number of neurons in 2-layer SNN had to be increased by $64X$. The authors in [16] achieved an average accuracy of 98.6% for MNIST dataset with two hidden layers consisting of 800 neurons in each layer. The quest of making SNNs larger and deeper for higher accuracy have compromised their energy efficiency and introduced challenges as mentioned below:

1. Large SNNs implemented on emerging memristive crossbar structures [17]–[19] are limited by the crossbar size. Large crossbars suffer from supply voltage degradation, noise generated from process variations, and sneak paths [20], [21].

2. SNNs with numerous synapses involve higher number of computations making them slower and energy inefficient.

SNNs are driven by the synaptic events and the total computation, memory, communication, power, area, and speed scale with the number of synapses [14]. We propose a pruned and weight quantized SNN topology with self-taught Spike Timing Dependent Plasticity (STDP) based learning. STDP, in turn, is also used to classify synapses as critical and non-critical. The non-critical synapses are pruned from the network, whereas the critical synapses are retained and weight quantized. Such pruning of connections and weight quantization can lead to their efficient implementations in emerging cross-bar arrays such as resistive random access memories (R-RAMs) [22], [23], magnetic tunnel junctions [24], or domain-wall motion based magnetic devices [25]. Such cross-bars, even though suitable for implementing efficient dot-products required for neural computing, are constrained in size, because of non-idealities such as sneak paths, weight quantization, and parameter variations [20], [21]. The resulting sparse SNN can achieve $2-3X$ improvement in energy, $2-4X$ in area and $2-3X$ in testing speed.

Synaptic pruning is commonly observed during the development of human brain. The elimination of synapses begins at the age of two and continues till adulthood, when synaptic density stabilizes and is maintained until old age [26]. From hardware implementation of neural networks, synapses are a costly resource and needs to be efficiently utilized for energy efficient learning. If synapses or connections are properly pruned, the performance decrease due to synaptic deletion is small compared to the energy savings [27]. This has motivated researchers to apply the technique of pruning [28], and weight quantization [29] to compress nANNs. Pruning and quantization performed on state-of-the-art network AlexNet trained for ImageNet dataset provided $7X$ benefit in energy efficiency along with $35X$ reduction in synaptic weight storage without any loss of accuracy [29]. The authors in [28] prune the connections of an nANN trained using backpropagation based on the Hessian of the loss function. The number of parameters were reduced by a factor of two while maintaining the same test accuracy. The supervised learning algorithm in [30] pruned the hidden layer neurons with low dominance to reduce network size. The network achieved similar performance with $4X$ less parameters for Fisher Iris problem compared to other spiking networks. The idea of pruning is based on identifying parameters with small saliency, whose deletion will have minimal effect on the error. These networks were trained with supervised learning algorithms, but in SNNs with unsupervised training it is difficult to calculate such parameters since there is no such defined error function. In real world, obtaining unlabeled images for unsupervised learning is much easier than gathering labeled images for supervised learning. The authors in [31] designed a SNN with synapses characterized by activation levels. The activation level of a synapse changed according to the timing of the pre- and post-synaptic activity. The synapses with the lowest activation level after the network had stabilized were

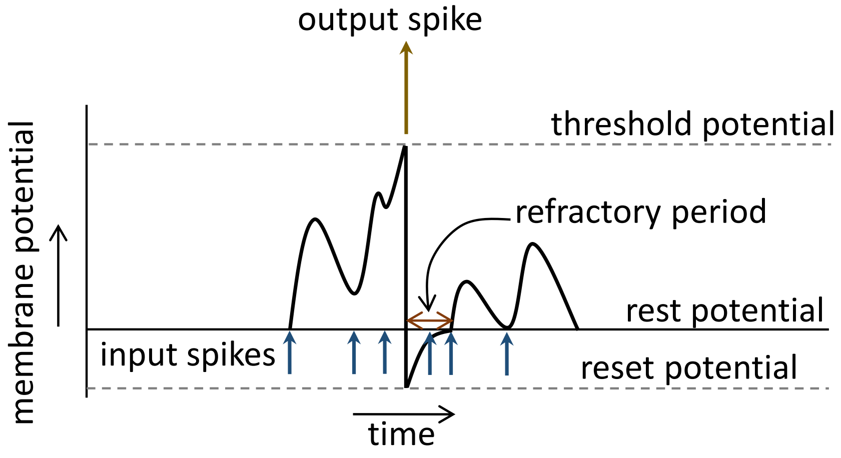


**Figure 2.1.** Leaky-Integrate-and-Fire (LIF) model of a single neuron's membrane potential dynamics in response to input spikes in SNN.

pruned to reduce network size. The process of pruning is applied only at the end and the remaining active synapses represent a small percentage of the overall connections. The novelty of our approach lies in self-taught STDP based weight pruning where the connections to be pruned are decided based on their weights learned by the unsupervised STDP algorithm. Connections having STDP weights above a threshold are considered critical while others are temporarily pruned. The threshold is fixed before training and is referred as pruning threshold. The critical connections are weight quantized to further reduce network complexity. The resulting compressed topology is energy-efficient while maintaining accuracy and alleviates the issues that constrain the scalability of crossbar structure, leading to robust design of neuromorphic systems. The main contributions of the work are mentioned below:

1. Online pruning based on the implicit correlation in neuronal activity resulting in a structured pruning methodology compared to simple thresholding.

2. Pruning the connections at regular intervals during training instead of just at the end to improve the training time.

3. The connections are pruned temporarily until the end of training to adapt to new training data. The low dominance connections are classified as non-critical and they become critical if new training data is introduced, therefore making the network scalable.

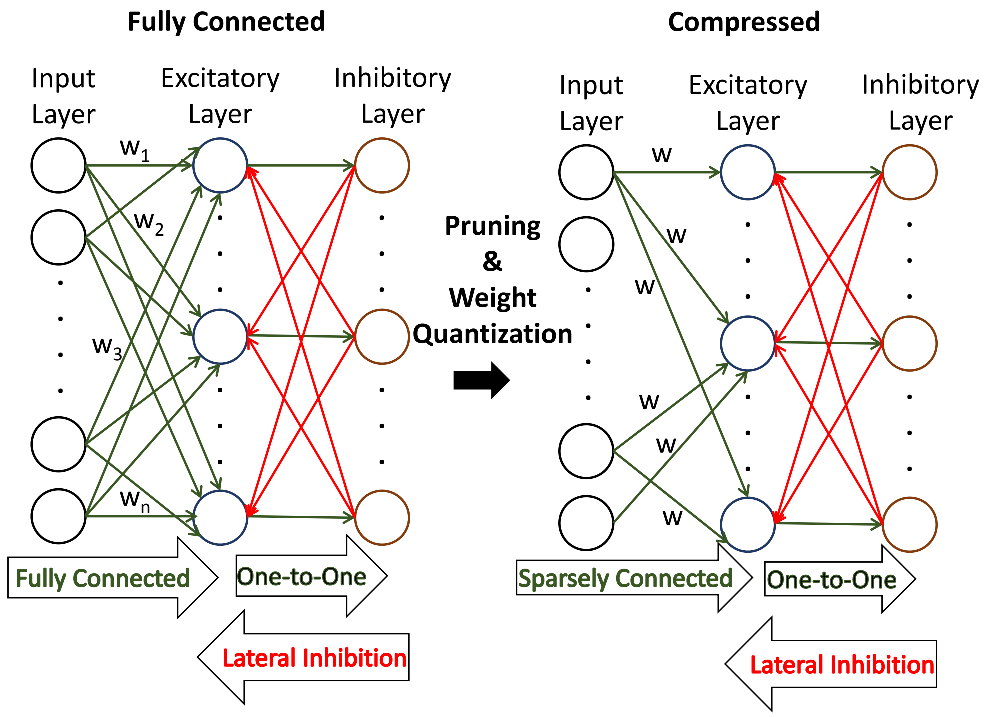


**Figure 2.2.** SNN topology with lateral inhibition. Input to excitatory is fully connected which is later pruned. Excitatory to inhibitory is one-to-one connected, whereas inhibitory is backward connected to all the excitatory except the one it receives the connection from. Pruning is performed only on the input to excitatory connections.

4. The quantization process is controlled by the underlying device technology implementing the synapse. The number of quantization levels depends on the available conductance states in the cross-bar arrays.

The rest of the chapter is organized as follows. Section 2.2 provides background information on the neuron and the synapse models and the STDP learning algorithm employed in this work. The network topology and the training and testing schemes are also briefly discussed. Section 2.3 presents the proposed compression techniques; STDP based pruning and weight quantization and sharing. The experiments on the proposed topology are presented in Section 2.4. The results of the experiments are analyzed in Section 2.5. Conclusions are drawn in Section 2.6. Section 2.7 discusses the implication of pruning threshold on the trade-off between accuracy and network size.

## 2.2 Background

### 2.2.1 Neuron & Synapse Model and STDP Learning

We employ the Leaky-Integrate-and-Fire (LIF) model [15] to simulate the membrane potential dynamics of a neuron in our spiking network model. Fig. 2.1 shows the change in membrane potential of a single post-neuron in response to input spikes (blue arrows) from pre-neurons. The membrane potential increases at the onset of a spike and exponentially decays towards rest potential in the absence of spiking activity. The post-neuron fires or emits a spike when its potential crosses the threshold and immediately its potential is set to a reset value. After firing the post-neuron goes into a period of inactivity known as refractory period during which it is abstained from spiking, irrespective of input activity as shown in Fig. 2.1.

The connection between two neurons is termed a synapse and is modeled by the conductance change which is modulated by the synaptic weight ($w$). The synaptic weight between a pair of neurons increases (decreases) if the post-neuron fires after (before) the pre-neuron has fired. This phenomenon of synaptic plasticity where the weight change is dependent on the inter spike timing of pre- and post-neuron is termed STDP. We adopt the power law weight-dependent STDP model, where the weight change is exponentially dependent on the spike timing difference of the pre- and post-neuron ($t_{pre} - t_{post}$) as well as the previous weight value [15].

### 2.2.2 Network Topology

The SNN topology for this work is shown in Fig. 2.2. It consists of input layer followed by excitatory and inhibitory layer. The input layer is fully connected to the excitatory layer, which in turn is one-to-one connected to the inhibitory layer. The number of neurons in the excitatory layer are varied to achieve better accuracy, whereas the number of neurons in the inhibitory layer is the same as the number in the excitatory layer. Each inhibitory neuron is backward connected to all the excitatory neurons except for the one from which it receives a connection from. Thus, the inhibitory layer provides lateral inhibition which discourages simultaneous firing of multiple excitatory neurons and promotes competition among them to learn different input features. The process of pruning and weight quantization is applied to the excitatory synapses to obtain the compressed topology as shown in Fig. 2.2. The fully connected topology serves as the baseline design and we compare the results of the compressed design with baseline.

To ensure similar firing rates for all neurons in the excitatory layer we employ an adaptive membrane threshold mechanism called homoeostasis [15]. The threshold potential is expressed as $V_{thresh} = V_t + \theta$, where $V_t$ is a constant and $\theta$ is changed dynamically. $\theta$ increases every time a neuron fires and decays exponentially. If a neuron fires more often, then its threshold potential increases and it requires more inputs to fire again. This ensures that all neurons in the excitatory layer learn unique features and avoids few neurons from dominating the response pattern.

### 2.2.3 Training & Testing

The connections from input to excitatory layer are trained using the STDP weight update rule to classify an input pattern. The training is unsupervised as we do not use any labels to update the weights. The weight update is given by the formula described in section 2.3.1. The input image is converted into a Poisson spike train based on individual pixel intensities. The excitatory neurons are assigned a class/label based on their average spiking activity over all the images. During testing, the class prediction is inferred by averaging the response of all excitatory neurons per input. The class represented by the neurons with the highest spiking rate is predicted as the image label. The prediction is correct if the actual label matches the one predicted by the SNN. This is similar to the approach followed in [15].

## 2.3 Compression Techniques

In this section, we describe the two compression techniques (pruning and weight quantization) employed in this work to convert the 2-layer fully connected SNN into a compact and sparse topology for digit and image recognition.

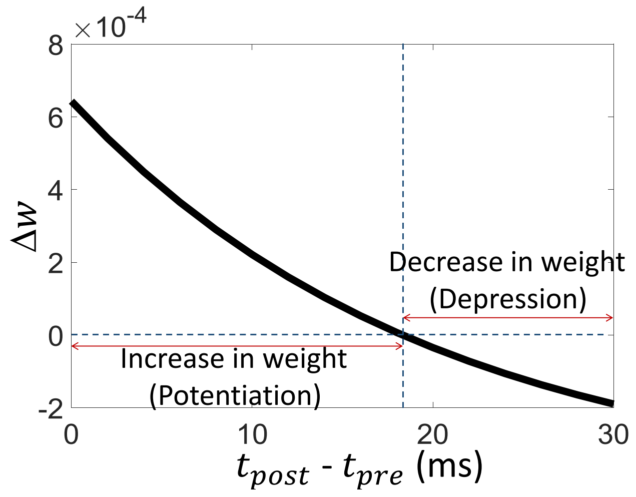


**Figure 2.3.** Change in synaptic weight based on temporal correlation in pre- and post-synaptic spikes. ($\eta = 0.002, \tau = 20ms, offset = 0.4, w_{max} = 1, w = 0.5, \mu = 0.9$)

### 2.3.1 STDP Based Pruning

Spike Timing Dependent Plasticity (STDP) is widely used as an unsupervised Hebbian training algorithm for SNNs. STDP postulates that the strength of the synapse is dependent on the spike timing difference of the pre- and post-neuron. The power law weight update for an individual synapse is calculated as

$$\Delta w = \eta \times [\mathrm{e}^{(\frac{t_{pre}-t_{post}}{\tau})} - offset] \times [w_{max} - w]^{\mu}$$

where $\Delta w$ is the change in weight, $\eta$ is the learning rate, $t_{pre}$ and $t_{post}$ are the time instant of pre- and post-synaptic spikes, $\tau$ is the time constant, $offset$ is a constant used for depression, $w_{max}$ is the maximum constrained imposed on the synaptic weight, $w$ is the previous weight value, $\mu$ is a constant which governs the exponential dependence on previous weight value. The weight update is positive (potentiation) if the post-neuron spikes immediately after the pre-neuron and negative (depression) if the spikes are far apart (Fig. 2.3). We employ STDP to train the excitatory synapses as well as to classify them as critical or non-critical. The synapses whose weights do not increase for a set of inputs are likely to have not contributed towards learning and thus can be potential candidates for deletion. On the

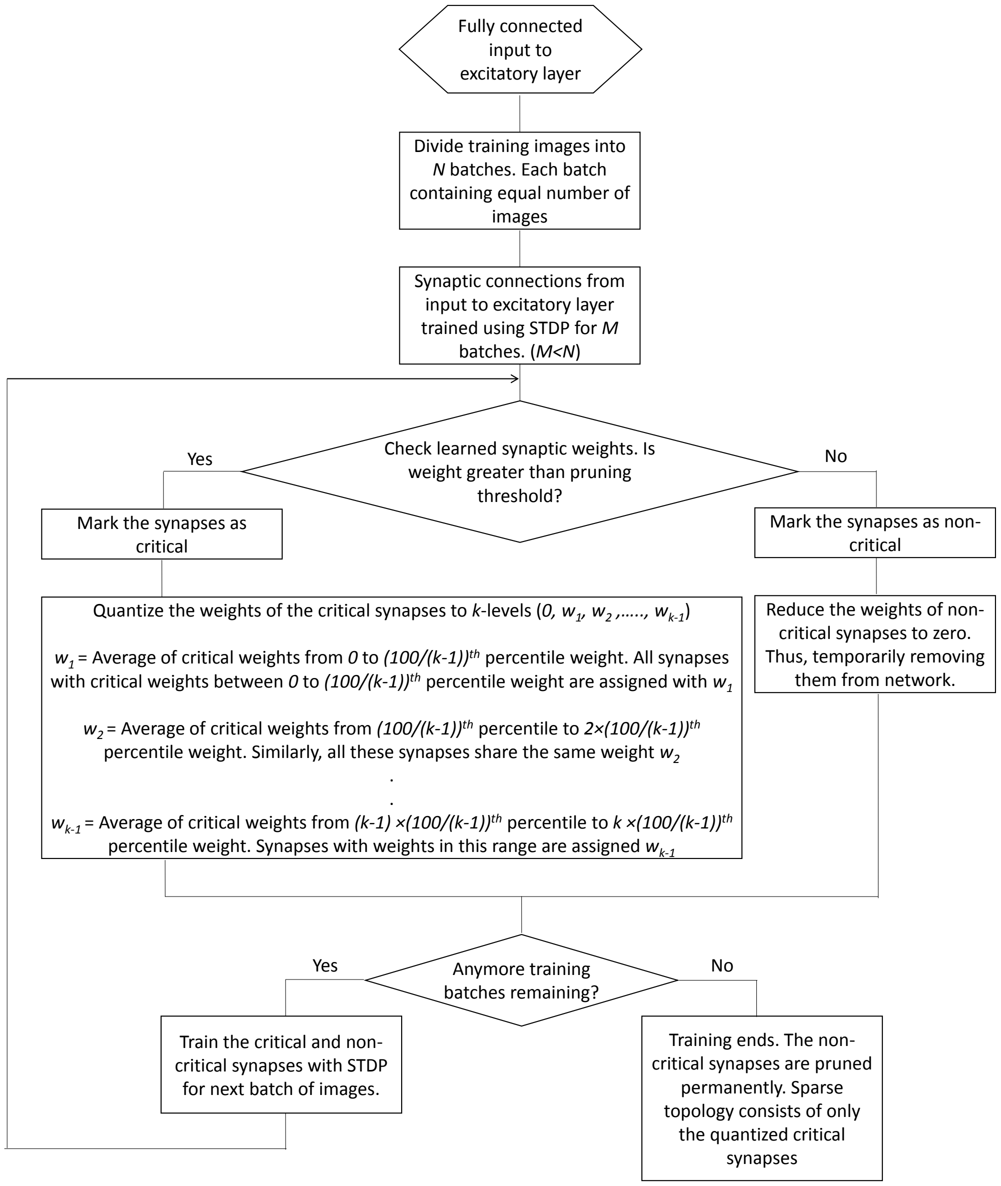

Fully connected input to excitatory layer
Divide training images into N batches. Each batch containing equal number of images
Synaptic connections from input to excitatory layer trained using STDP for M batches. (M<N)
Check learned synaptic weights. Is weight greater than pruning threshold?
Yes
No
Mark the synapses as critical
Mark the synapses as non-critical
Quantize the weights of the critical synapses to k-levels (0, w1, w2 ,....., wk-1)
w1 = Average of critical weights from 0 to (100/(k-1))th percentile weight. All synapses with critical weights between 0 to (100/(k-1))th percentile weight are assigned with w1
w2 = Average of critical weights from (100/(k-1))th percentile to 2×(100/(k-1))th percentile weight. Similarly, all these synapses share the same weight w2
wk-1 = Average of critical weights from (k-1) ×(100/(k-1))th percentile to k ×(100/(k-1))th percentile weight. Synapses with weights in this range are assigned wk-1
Reduce the weights of non-critical synapses to zero. Thus, temporarily removing them from network.
Anymore training batches remaining?
Yes
No
Train the critical and non-critical synapses with STDP for next batch of images.
Training ends. The non-critical synapses are pruned permanently. Sparse topology consists of only the quantized critical synapses

**Figure 2.4.** Flowchart of the proposed algorithm for compressing SNN using pruning and weight quantization.

other hand, synapses with higher weights have most likely learned the input pattern and can be classified as critical (provided they were initialized with small weights). The characteristic features of the input is captured in connections with higher weights and are critical for correct classification. Thus, synapses with STDP trained weights $(w + \Delta w)$ above pruning threshold are considered critical and all other synapses are marked as non-critical. The process of pruning and training is performed repeatedly by dividing the entire training set into

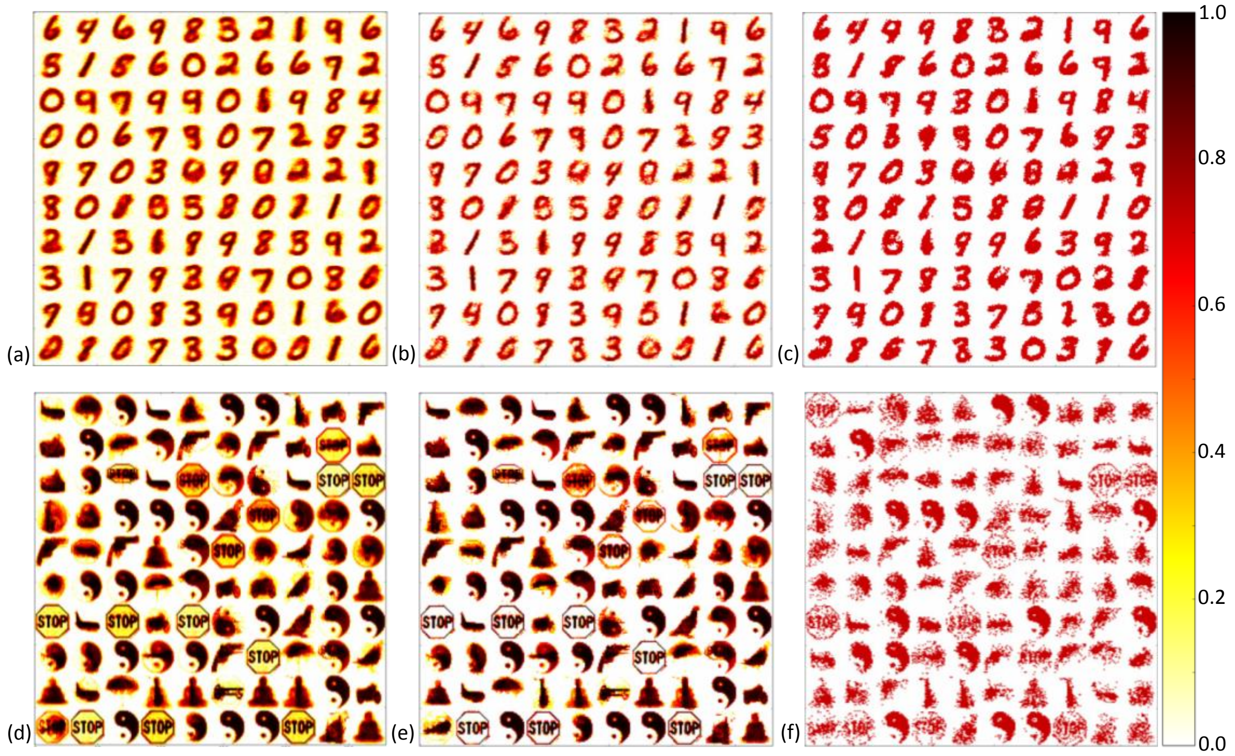


**Figure 2.5.** Rearranged weights of the connections from input to excitatory for (a) MNIST baseline; (b) MNIST pruning; (c) MNIST pruning and quantization; (d) Caltech 101 baseline; (e) Caltech 101 pruning; (f) Caltech 101 pruning and quantization.

multiple batches. After each batch the weights of all non-critical synapses are reduced to zero (they still remain in the network) and the network is trained with the next batch. The synapses with zero weight continue to participate in training, thus a non-critical synapse may become critical for different inputs. The process of reducing the weight to zero instead of eliminating the connection is essential for the network to learn the representation of inputs which appear in latter batches. The elimination of synapses will either make the network not learn the new representations or force the network to forget previous representations in order to learn new inputs. The process of retaining non-critical synapses with zero weight makes the network scalable. In the final training step when all the training images have been presented to the network any remaining non-critical connections are permanently removed from the network. The training starts with a fully connected network and the number of critical connections gradually decrease over time. At the end of training only the critical connections capturing the characteristic features of the inputs remain.

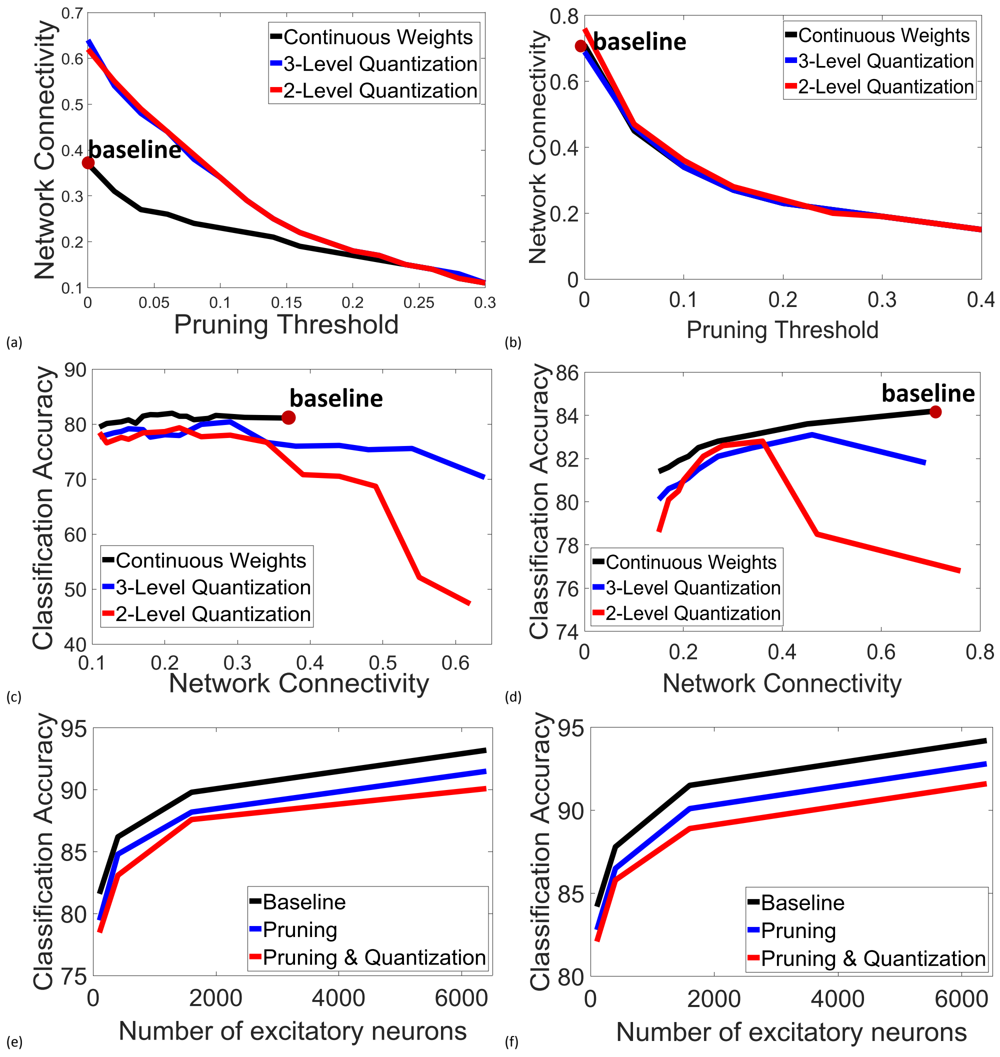


**Figure 2.6.** Variation in network connectivity with pruning threshold for (a) MNIST; (b) Caltech 101. Classification accuracy for different network connectivity for (c) MNIST; (d)Caltech 101. Classification accuracy for different number of excitatory neurons for (e) MNIST; and (f) Caltech 101.

### 2.3.2 Weight Sharing and Quantization

The process of pruning reduces the overall connectivity, but as mentioned in Section 2.1, SNNs with continuous weight values are difficult to implement in crossbar structures due

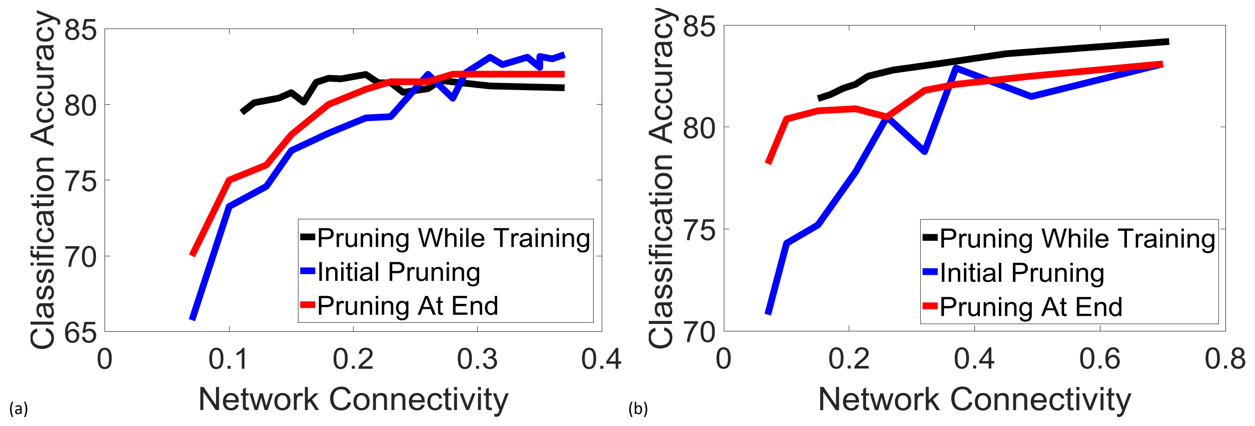


**Figure 2.7.** Classification accuracy for different network sparsity achieved by pruning the connections during training, before training and after training for (a) MNIST; and (b) Caltech 101.

to limitations on the number of available conductance states in devices implementing the synapse. Weight sharing and quantization discretizes the weights to the available number of conductance states. For example, network with 2-level weight quantization has only two values of weights: 0 (no connection) and $w$. All the synapses share the same weight ($w$) and the entire network can be represented as a sparse binary matrix. A 2-level weight quantized SNN can be implemented in crossbar architecture with a single fixed resistor [32], where $w$ is the conductance of the resistor. The value of $w$ is the average weight of all the critical connections trained using STDP. For example, we start with a network with $n$ number of synapses and after training (with pruning) $m$ critical synapses remain. The weights of the $m$ critical synapses ($w_1, w_2, \ldots, w_m$) are continuous and computed based on the STDP formula. The common weight value $w$ is the average of $w_1$ to $w_m$. The average is calculated after each pruning step and all the critical connections share the same average weight ($w_1$ to $w_m$ is replaced with $w$). Like pruning, the process of weight quantization and sharing is performed repeatedly after each training batch. The value of $w$ changes at every quantization step and the final value is obtained after training the network for all the input batches. Similarly, the weights can be quantized to 3-levels: $0, w_1, w_2$, where $w_1(w_2)$ is the low (high) conductance value. The conductance values are computed by calculating the $50^{th}$ percentile or the median weight of all the critical connections. The lower conductance value

$w_1$ is the average of all weights between 0 and the median weight, $w_2$ is the average of rest of the weights. The critical synapses with weights between 0 and the median weight are assigned $w_1$. The critical synapses with weights between median weight and the maximum weight share the quantized value of $w_2$. The accuracy of the network is directly proportional to the number of quantization levels. The performance of the system improves with more number of conductance levels. In a quantized SNN most of the connections share the same weight which reduces the implementation complexity.

Fig. 2.4 summarizes the proposed algorithm for achieving a pruned and weight quantized SNN. The 2-layer untrained network is initialized with full connectivity from input to excitatory layer. The weights are randomly assigned from a uniform distribution. The training images are divided into $N$ batches of equal number of images. The excitatory synapses are trained with STDP weight update rule for $M(M < N)$ training batches. The connections with current weights above the pruning threshold are classified as critical, rest of the connections are marked as non-critical. The non-critical connections are pruned by reducing their weights to zero. The weights of the critical synapses are quantized to the required number of conductance states. The pruned and quantized network is trained with STDP weight update rule for the next training batch. The process of pruning and quantization is performed at regular intervals for all the remaining training batches. The training ends when all the batches have been presented to the network. The first pruning and quantization step is delayed for $M$ batches to ensure proper detection of critical connections and to mitigate the bias due to random initialization of weights. The randomly initialized synapses require more training images to capture the input characteristic features. Once the input features have been captured the pruning can be performed more often (after every batch). Once the critical connections are identified, they more or less remain the same during training. So, the first pruning step is very crucial and more than one training batch is needed to identify the critical synapses. The baseline design is trained in a similar fashion with no pruning and quantization. All the training images are presented in one batch and the weights are trained using STDP.

## 2.4 Experimental Methodology

The proposed SNN topology is simulated in the open source spiking neuron simulator BRIAN implemented in Python [33]. BRIAN allows the modeling of biologically plausible neurons and synapses defined by differential equations. The parameters for the models are same as [15]. We tested our network for digit recognition on the MNIST dataset [34] and image recognition on a subset of images from the Caltech 101 dataset [35]. We propose two compression mechanisms: pruning and weight quantization. These mechanisms are applied on top of the baseline training algorithm. The baseline design is a fully connected network trained with STDP learning algorithm. We compare this design with a network trained in a similar fashion, but with compression techniques applied at appropriate intervals. Unlike nANNs which have pre-trained models available like AlexNet, VGG Net, GoogLeNet, etc., SNNs do not have such standard pre-trained models. Therefore, to compare our approach we train the baseline design in the absence of pruning and quantization.

### 2.4.1 MNIST Dataset

MNIST dataset contains $28 \times 28-$pixel sized grayscale images of digits 0-9. Thus, the input layer has 784 ($28 \times 28$) neurons fully connected with 100 excitatory neurons. The dataset is divided into 60,000 training and 10,000 testing images. We further divide the 60,000 training images into batches of 5,000 images ($N = 12$). The baseline design is trained with entire 60,000 images presented one after another. The compressed topology is initially trained for three training batches totaling 15,000 images ($M = 3$). STDP based critical connections are weight quantized and the weights of the non-critical connections is reduced to zero. The pruned and quantized network is trained with the next training batch. The process of pruning and quantization is performed after every batch henceforth. The rearranged input to excitatory synaptic weights of the trained baseline topology with 100 excitatory neurons is shown in Fig. 2.5(a). Fig. 2.5(b) shows the rearranged synaptic weights of the same network compressed with a pruning threshold of 0.3 and having continuous weight values. The rearranged synaptic weights of the pruned and 2-level weight quantized network is shown in Fig. 2.5(c).

### 2.4.2 Caltech 101 Dataset

Caltech 101 dataset is a collection of images of objects belonging to 101 different categories. Each category consists of 40 to 800 of around $300 \times 200-$pixel sized RGB images. The dataset also provides annotations for the object in the image which we use to separate the object from the background. Unlike MNIST images, we preprocess the Caltech 101 images to obtain $28 \times 28-$pixel sized grayscale images. Maintaining the same image size across datasets ensures that we do not need to change the network parameters. Out of 101, we selected 10 categories (yin yang, saxophone, stop sign, wrench, revolver, Buddha, airplanes, pigeon, motorbikes, umbrella) and randomly divided the total images in each category with 80% training and 20% testing images. Since each category has different number of images we create copies of images so that each category has similar number of training and testing images. This is necessary to avoid categories with more images to dominate the learning in the network. The preprocessing steps involved: converting the images to grayscale, averaging the pixels with Gaussian kernel of size $3 \times 3$ to suppress the noise and resizing the image to $28 \times 28$ pixels. All the preprocessing steps are performed using the OpenCV library [36] in python. The training set consists of 10,000 images with 1,000 images per category. The 10,000 images are further divided into batches of 500 images ($N = 20$). The baseline fully connected design is trained with entire 10,000 images. The compressed topology is initially trained with ten training batches totaling 5,000 images ($M = 10$). The critical connections are identified using STDP and weight quantized. The non-critical synapses are pruned. The pruned and quantized network is trained with all the remaining batches with pruning and quantization performed after every training batch. Figs. 2.5(d), (e) and (f) show the rearranged synaptic weights for the baseline, pruned and weight quantized topologies, respectively. Compression is performed with pruning threshold of 0.2 and 2-level weight quantization.

## 2.5 Results & Analyses

In this section, we analyze the results and compare the performance of compressed topology with the baseline design. The results are evaluated based on different parameters like pruning threshold and number of excitatory neurons. The removal of connections during training is compared with training a sparse network, both having similar connectivity.

### 2.5.1 Comparison with varying pruning threshold

The network connectivity is a strong function of the pruning threshold; higher the threshold, sparser is the network. The network connectivity is defined as the ratio of the actual number of connections to the total number of possible connections. The total number of possible connections with 100 excitatory neurons is 78400 ($784 \times 100$). The number of actual connections depend on the pruning steps. Figs. 2.6(a) and (b) shows the variation in final network connectivity with pruning threshold for MNIST and Caltech 101 datasets, respectively. The red dot in Figs. 2.6(a) and (b) with zero pruning threshold denotes the baseline design with no compression techniques applied. Ideally, the connectivity should be 1 since the connections are not pruned during training. The reduction in connectivity results from the inherent depression in the STDP learning rule. The further reduction in connectivity is achieved by increasing the pruning threshold. The compressed topologies are less sparse for low pruning threshold compared to baseline. This is due to weight quantization and sharing in early training stages. The shared weight is the average of all critical weights which is higher than almost half the critical weights. Thus, the average weight replaces half the STDP learned weights which were supposed to be much lower. This reduces the effect of inherent STDP depression on these synapses and reduces the probability of their removal. Figs. 2.6(c) and (d) show the test accuracy for different network connectivity for MNIST and Caltech 101 datasets, respectively. The baseline topology has an accuracy of 81.6% (MNIST) and 84.2% (Caltech 101) which is consistent with the results shown in [15]. The highest classification accuracy achieved for the compressed topology is 79.5% (MNIST) and 82.8% (Caltech 101). The accuracy degrades slightly compared to baseline but at the same time there is immense drop in network connectivity. The compressed topology is 75% (36%) sparser than the baseline topology for MNIST (Caltech 101) dataset. The accuracy drops for high network connectivity for the weight quantized networks. In quantized networks most of the connections share the same weight values and as the connectivity increases the synapses

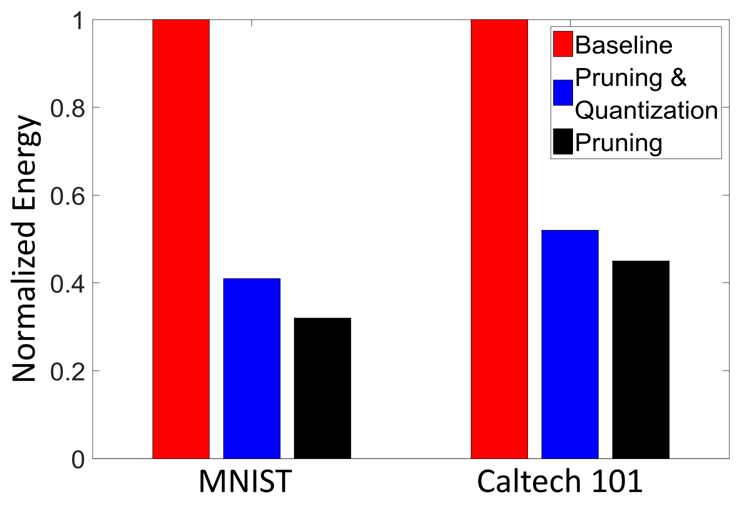


**Figure 2.8.** Normalized improvement in energy with pruning and weight quantization compared to baseline topology.

become more alike. This introduces confusion in the network as the spiking activity for different classes become similar. This is not observed in networks with continuous weights because the individual weight values are different which makes the spiking activity of various classes differ from one another. Therefore, the quantized networks have to be sparse to achieve higher classification accuracy.

### 2.5.2 Comparison with varying number of neurons

The change in classification accuracy with the number of excitatory neurons for MNIST and Caltech 101 datasets is shown in Figs. 2.6(e) and (f), respectively. The network is trained with a pruning threshold of 0.15(0.10) for MNIST (Caltech-101) dataset and the weights are quantized to 3-levels. These parameters correspond to the optimal trade-off between accuracy and energy as discussed in section 2.7. The baseline design with 6400 neurons achieved an accuracy of 93.2% for MNIST and 94.2% for Caltech 101 datasets. The pruned topology with similar number of neurons achieved an accuracy of 91.5% with 8% connectivity and 92.8% with 12% connectivity for MNIST and Caltech 101 datasets, respectively.

### 2.5.3 Pruning while training

The objective of pruning is to increase the sparsity in the network. This can be achieved in multiple ways: removing connections during training of a fully connected network, training a sparse network or removing connections at the end of training. In first case the connections are removed systematically based on some parameters. The second approach is performed by randomly removing connections from a fully connected topology to produce a sparse network. The removal of connections at the end of training identifies low weight connections and prunes them from the network. This results in a network with similar sparseness but the network is no longer trained after the pruning step. Nevertheless, in all the cases the final trained network connectivity is same. Figs. 2.7(a) and (b) shows the classification accuracy with varying network connectivity for all three approaches for MNIST and Caltech 101 datasets, respectively. The networks with initial pruning and pruning at the end are trained similar to baseline with no compression techniques. The pruning while training is the approach followed in rest of the work, where pruning is performed at regular intervals during training. The results for all the networks are shown for continuous weight distribution. Pruning the connections during training performs better since only the non-critical connections are removed. The network with initial sparsity is constructed by randomly removing connections. This shows that STDP successfully identifies the non-critical connections. Though pruning at the end of training removes some of the non-critical connections, the network's accuracy is lower compared to the proposed approach for highly sparse network. The absence of training after pruning and a single pruning step may be attributed for the reduction in classification accuracy.

### 2.5.4 Reduction in spike count or energy

The decrease in connectivity due to pruning leads to reduced spiking activity in the excitatory layer. The active power of a SNN is proportional to the firing activity in the network [14]. Thus, the energy can be quantified as the reduction in spike count of excitatory neurons during testing. Fig. 2.8 shows the normalized reduction in spiking activity or energy for compressed topology with respect to baseline. The pruned topology shows $3.1X$ and

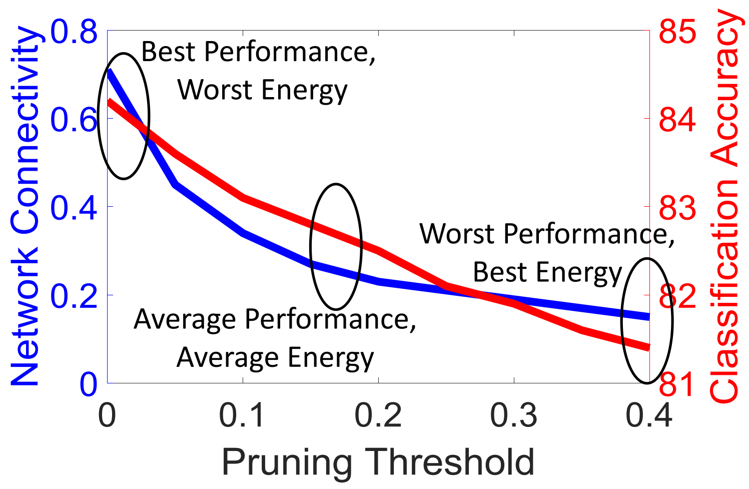


**Figure 2.9.** Network connectivity and corresponding classification accuracy achieved with varying pruning threshold for a 100-neuron network with continuous weights and trained on a subset of categories from CALTECH-101 dataset.

$2.2X$ improvement in energy whereas the 2-level weight quantized network achieves $2.4X$ and $1.92X$ improvement for MNIST and Caltech101 datasets, respectively. The compressed topology may achieve additional energy benefits from implementation in crossbar structures with low power devices. The emerging post-CMOS devices like MTJ, R-RAM and domain wall motion based devices consume very low power in idle state due to elimination of leakage. But these devices have limited number of programmable conductance states. The compressed topology quantized to the available number of conductance states can reap the energy benefits provided by these devices. The baseline design with continuous weight distribution is difficult to implement with these devices. The introduction of sparseness also reduces the area of the cross-bar arrays. The number of devices in the cross-bar is proportional to the number of connections. Pruning threshold of 0.15 (0.10) for MNIST (Caltech-101) dataset results in $4X$ ($2.6X$) reduction in number of connections with 0.6% (1.1%) drop in accuracy. The reduction in number of connections can be directly proportional to the area benefits if the cross-bar arrays are arranged efficiently. Therefore, our proposed approach results in $4X$ ($2.6X$) area reduction for MNIST (Caltech-101) dataset with minimal drop in accuracy.

## 2.6 Conclusions

In this work, we propose two compression techniques, pruning and weight quantization to compress SNNs. Compressed SNNs not only provide energy benefits but also mitigate the issue of limited programmable conductance states of post-CMOS devices for neuromorphic implementation. The novelty of our approach lies in fact that STDP learning rule is used to decide the network pruning and the weights of the critical connections are quantized to specific levels depending on device and technology requirements. The compressed topology is compared with the 2-layer fully connected topology for digit recognition with MNIST dataset and image recognition with Caltech 101 dataset. The proposed topology achieves $3.1X$ and $2.2X$ improvement in energy for MNIST and Caltech 101 datasets, respectively, compared to baseline fully connected SNN. The optimal compression parameters like pruning threshold and weight quantization levels are decided by performing multiple experiments with different images. Additionally, it is worth mentioning that the proposed topology reduced the training time by $3X$ and $2X$ for MNIST and Caltech 101 datasets, respectively, by achieving faster training convergence.

## 2.7 Discussion

Our results show that pruning and quantization can effectively reduce the number of connections during training in a SNN with minimal loss in accuracy. The process of pruning is controlled by the critical parameter ‘pruning threshold’ and the weight quantization step requires to make a proper judgement on the number of quantization levels. The number of quantization levels depend on the number of programmable conductance states available in the device technology implementing the synapse. Modern memristive cross-bars have shown 16 robust conductance states [37]. The accuracy of the network increases with more quantization levels and the best performance is achieved with continuous weights as shown in Figs. 2.6(a)-(f). To simplify our experiments, we considered only two and three level weight quantization along with continuous weights. The choice of pruning threshold is not as straightforward as it needs to consider the trade-off between accuracy and reducing connections. Fig. 2.9 shows this trade-off with varying pruning threshold. To the left, the pruning threshold is low resulting in dense network connectivity with high accuracy. To the right, the pruning threshold is high providing more area and energy benefits at the cost of accuracy degradation. Thus, the choice of pruning threshold depends on the application's tolerance on accuracy loss and energy budget. The bio-inspired STDP pruning mechanism allows to perform low power classification tasks but we still have a long way to go in order to match the accuracy and power efficiency of the human visual system. In the future we would like to include other mechanisms on top of the compression techniques to further improve accuracy and energy.

# 3. STDP BASED UNSUPERVISED MULTIMODAL LEARNING WITH CROSS-MODAL PROCESSING IN SPIKING NEURAL NETWORK

## 3.1 Introduction

Humans interact and exchange information with their environment through multiple channels. The speech is accompanied by lip movements, face gesture and body language. Most of the information in real world comes through multiple input channels. Images not only contain colors but also depth measurements, videos contain visual and audio signals, neurophysiological process of sensory perception receives stimuli from vision, taste, hearing, smell and so on. In fact, human brain inherently integrates audio-visual information in order to understand speech. This was demonstrated where a visual /ga/ with a voiced /ba/ is perceived as /da/ by most subjects [38]. The human brain's ability to integrate inputs from multiple modalities has been studied extensively [39]–[41]. The integration occurs in specific brain areas and cross-modal coupling facilitates the communication of one modality to areas that intrinsically belong to other modalities. The goal of this work is to learn the cross-modal connections between areas of single modality in Spiking Neural Networks (SNNs) to improve the recognition accuracy and make the system robust to noisy inputs.

The idea of combining two modalities is interesting because the strengths and weaknesses of each modality can be complementary. The image may be affected under low lighting conditions but the audio is not hampered, whereas a background noise may attenuate the audio, the image remains unadulterated. If the non-idealities in the datasets are independent, then the probability of misclassification is the product of the misclassification probability of each modality. The product of two probabilities is always lower than each probability, thus each modality helps to overcome and compensate for the weaknesses of other modality. In this work, we present a SNN topology that receives inputs from two modalities (audio and image) and classifies the input in one of the ten categories. Each input modality is presented to a 2-layer SNN (referred as ensemble in rest of the chapter) and cross-modal connections are developed for fusion of the ensembles. The entire training is unsupervised, i.e., input labels are not used to update the synaptic weights. In the real world unlabeled data is more readily available than labeled data. The two unimodal ensembles are 2-layer SNN with lateral inhibition similar to [15]. The connections in the unimodal network are trained using Spike Timing Dependent Plasticity (STDP). STDP is a form of Hebbian learning where correlated activity between two neurons evokes modifications in the synapse connecting the two neurons. The cross-modal connections are also learned using STDP by observing the activity in the two unimodal ensembles when both are presented with the same input label in different modalities (audio and image). This is different from the previously reported models since the fusion is entirely unsupervised. It enables online and incremental learning, making the system more versatile. The unimodal network learns the characteristic features of individual modality. The characteristic features of image and audio are very different and thus the two unimodal ensembles learn different representations of the same label. The cross-modal connections fuse the different representations in a way to capture correlations across different modalities. Later, in the chapter we show that well learned cross-modal connections invoke activity in the network even if the input is missing that modality. This is similar to be able to recall a picture of 'apple' when we hear the word 'apple' even though we do not see any image of it. The cross-modal connections also help in suppressing the noise in the input; they invoke additional activity in the neurons with correct label, thereby reducing the effect of noise on overall spiking activity. The proposed network performs the task of recognizing digits (0-9) when presented with both image (MNIST dataset) and audio (TI46 dataset) modalities.

The learning of cross-modal connections involves the propagation of spikes through three layers of the SNN (modality-1 input, modality-1 excitatory, modality-2 excitatory). This can only be achieved when there are enough spikes in the modality-1 excitatory layer that can propagate to the other ensemble via cross-modal connections. In this work, we achieve this by controlling the input spikes so that there are enough spikes in the excitatory layers. Both the ensembles have to be presented the same input label samples at a time in order to capture the correlation between the two modalities. The cross-modal connections are formed between neurons which learn the same input label in different modalities. This ensures a higher spike count for the neurons of the correct label compared to other neurons. The two

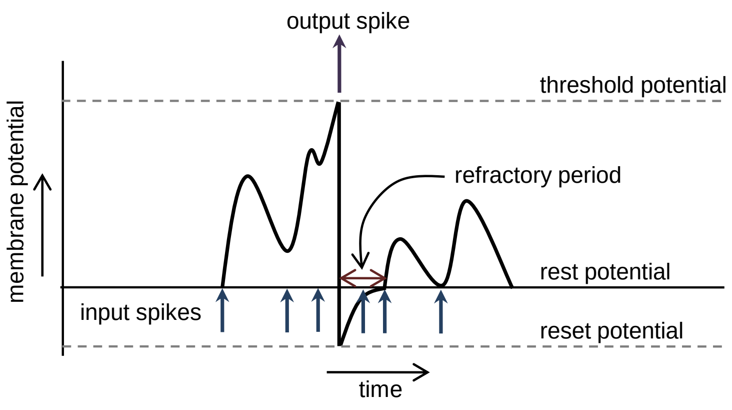


**Figure 3.1.** Membrane potential dynamics of a Leaky-Integrate-and-Fire (LIF) neuron model.

regions which intrinsically belong to individual modalities communicate via the cross-modal connections. These connections do not inhibit the spiking activity in the ensembles, thereby not affecting the learning of the unimodal connections.

## 3.2 Related Work

SNNs are primarily trained in three ways: supervised learning, unsupervised learning and converting a trained second generation non-spiking Artificial Neural Network (ANN) into SNN. The supervised learning [42], [43] and converted SNNs [44] achieve high accuracy but unsupervised local learning [15] is attractive because the system can self-learn from the unlabeled data which is more readily available. The two-layer SNN trained with STDP achieved an accuracy of 95% on the MNIST dataset [15]. The network trained with unsupervised learning and temporal coding achieved an accuracy of 81.9% on a subset of MNIST dataset [20]. Deep SNNs with multiple convolutional and pooling layers trained with STDP and linear Support Vector Machine (SVM) classifier achieved an accuracy of 98.4% on the MNIST dataset [45]. The authors in [46] combine STDP with Bienenstock-Cooper-Munro (BCM) theory to implement a learning rule. They test their network on a subset of TI46 speech corpus and achieve an accuracy of 95.25%. The digital liquid state machine (LSM) trained with spike-based online learning for speech recognition performs reasonably well on the subset of TI46 speech corpus [47]. Previous models of multimodal sensing and processing can be broadly divided into two categories: models that use statistical methods of computation (Maximum Likelihood [48], Hidden Markov Models [23] and Gaussian Mixture Models [48]) and models based on deep learning (Deep Belief Networks [49], Deep Boltzmann Machines [50] and Deep Autoencoders [51] ). The task of audiovisual human authentication is performed using principal component analysis on visual information and hidden Markov model for speech recognition. It uses fuzzy logic to make the final decision [23]. RGB-D (Red Green Blue - Depth) object recognition with color and depth as two modalities is performed by constructing separate convolution layers for color and depth and later merged using a multi-modal layer [52], [53]. Most of these systems process modalities separately and the fusion is made with AND and OR gates [48], fuzzy logic [23] and support vector machine [54]. In this work we tackle the problem of integrating multiple modalities with bio-inspired SNN. The authors in [55] applied the ensemble approach to recognize digits from the MNIST dataset. The input image is divided into multiple parts and applied to different ensembles. The ensembles exchange information among themselves via predictive connections and make a collective decision to recognize the input image. Although the predictive connections are similar to the cross-modal connections defined in this work, the formation of these connections are completely different. In this work, we learn the connections with unsupervised STDP learning rule. The network is randomly initialized and the STDP algorithm forms the cross-modal connections between ensembles, whereas the predictive connections in [55] have user-defined connectivity and fixed weight values. The ensembles in this work receive input in multiple modalities and the cross-modal connections develop the correlation between neurons that process different modalities.

*To the best of our knowledge, our work is the first to demonstrate unsupervised training of multimodal SNN for digit recognition.* The image dataset MNIST and its audio counterpart TI46 have been trained individually in previous works, but in this work we present a systematic methodology to train the network with multimodal inputs simultaneously. The main contributions of the work are mentioned below:

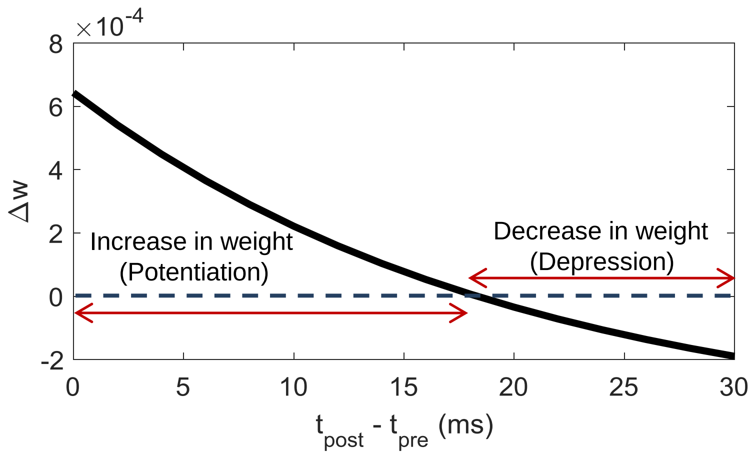


**Figure 3.2.** Weight change of a synapse with varying spike timing difference. $(\eta = 0.002, \tau = 20ms, offset = 0.4, w_{max} = 1, w = 0.5, \mu = 0.9)$

1. Synergistic online-learning framework for multimodal SNN. The individual modalities are trained as an ensemble and connected via cross-modal connections to perform a collective decision.

2. The multiple modalities are combined to take advantage of each other's strengths and suppress the effect of non-idealities and noise present in individual modalities.

3. The cross-modal connections learn the correlation in different modalities. The superior performance is achieved by invoking additional activity in the correlated neurons from the other ensemble.

## 3.3 Background

In this section we present the dynamics of a single neuron and synapse model employed in this work. The STDP learning mechanism used to train the proposed SNN is also discussed.

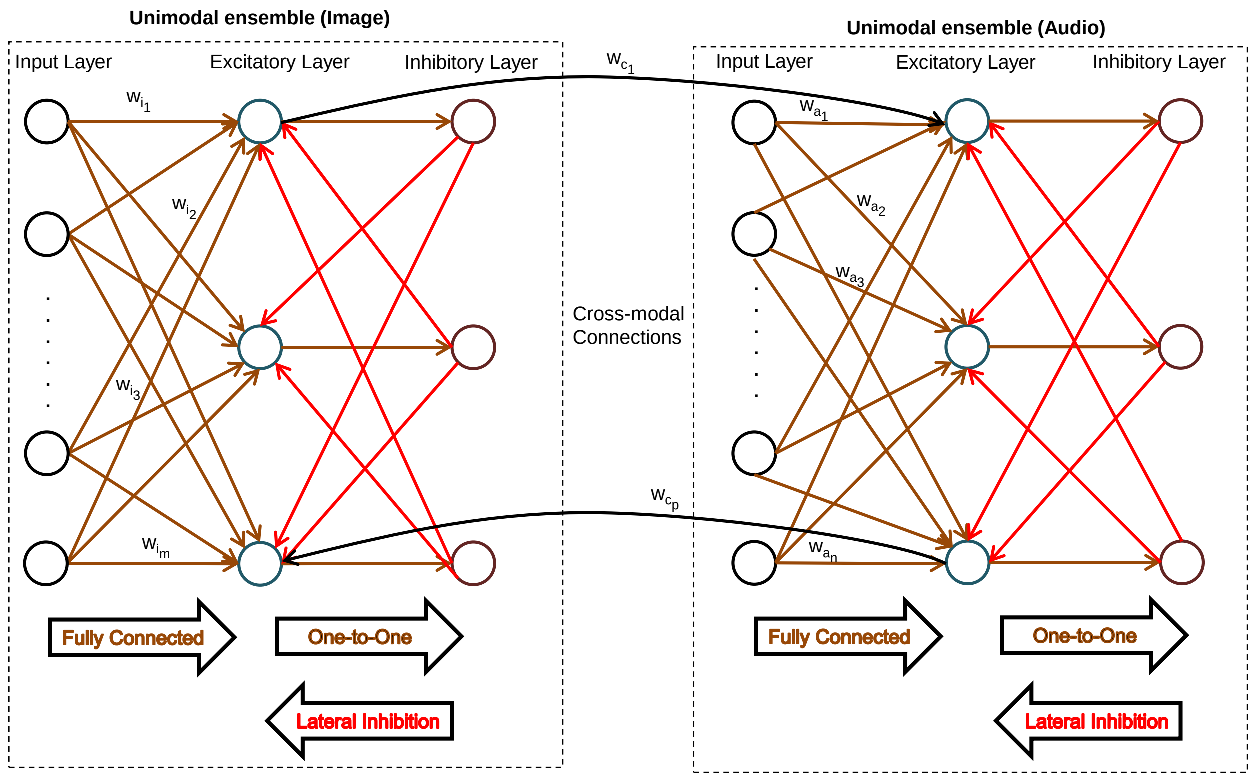


**Figure 3.3.** Proposed multimodal network topology. The input to excitatory layer is fully-connected in both the ensembles. The cross-modal connections are sparsely connected and randomly initialized. The integration of modalities is facilitated by the cross-modal connections.

### 3.3.1 Neuron & Synapse Model

The membrane potential V of a single neuron is described by the following equation and represented in Fig. 3.1

$$\tau \frac{dV}{dt} = (V_{rest} - V) + g_{\mathrm{e}}(V_{exc} - V) + g_{\mathrm{i}}(V_{inh} - V) \tag{3.1}$$

where $\tau$ is the time constant, $V_{rest}$ is the resting membrane potential, $V_{exc}$ and $V_{inh}$ are the equilibrium potential of the excitatory and inhibitory synapse, respectively, $g_{\mathrm{e}}$ and $g_{\mathrm{i}}$ are the conductance of the excitatory and inhibitory synapse, respectively. The membrane potential of the post-neuron is modulated in response to input spikes as shown in Fig. 3.1. The post-neuron emits a spike when its membrane potential crosses the threshold potential.

Immediately, after spiking the membrane potential falls to reset potential and the neuron enters into a period of inactivity referred as refractory period. The neuron is abstained from spiking during the refractory period irrespective of input activity.

The synapse connecting the two neurons is represented by a conductor whose conductance is described by

$$\begin{aligned} \tau_{g_{\mathrm{e}}}\frac{dg_{\mathrm{e}}}{dt} &= -g_{\mathrm{e}} \\ \tau_{g_{\mathrm{i}}}\frac{dg_{\mathrm{i}}}{dt} &= -g_{\mathrm{i}} \end{aligned} \tag{3.2}$$

where $g_{\mathrm{e}}(\tau_{g_{\mathrm{e}}})$ and $g_{\mathrm{i}}(\tau_{g_{\mathrm{i}}})$ are the conductance (time constant) for excitatory and inhibitory synapse, respectively. The conductance of a synapse is changed by $w$ when the pre-neuron fires. The value of $w$ is computed based on the learning mechanism discussed in the next sub-section.

### 3.3.2 Power-Law Weight-Dependent STDP

Spike Timing Dependent Plasticity (STDP) is an unsupervised Hebbian learning algorithm which modulates the conductance of the synapse based on the spike time of the pre- and post-neuron. In this work we employ the power-law weight update STDP. The weight change of a synapse is given by

$$\Delta w = \eta \times [\mathrm{e}^{\frac{t_{pre}-t_{post}}{\tau}} - offset] \times [w_{max} - w]^{\mu} \tag{3.3}$$

where $\eta$ is the learning rate, $t_{pre}$ and $t_{post}$ are the time instant of pre- and post-synaptic spike, respectively, $\tau$ is the time constant, $offset$ is a constant used for depression, $w_{max}$ is the maximum constrained imposed on the synaptic weight, $w$ is the previous weight value, $\mu$ is a constant which governs the exponential dependence on previous weight value. The weight is increased (potentiation) if the post-neuron fires immediately after the pre-neuron and reduced (depression) if the firing time difference ($t_{post} - t_{pre}$) is high (Fig. 3.2). As the learning rule is based on the precise timing of the spikes and also the dynamics of the LIF neuron is dependent on the spike-time of the pre-neuron, we employ spike-time

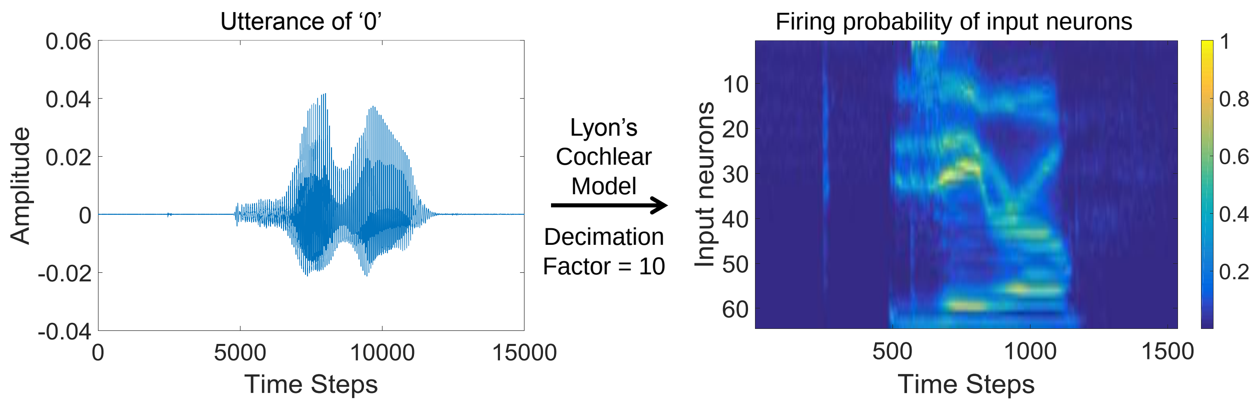


**Figure 3.4.** The utterance of '0' from the TI46 speech corpus sampled at 12.5 kHz is converted to a neural representation based on Lyon's cochlear model. The plot shows the firing probability for different input neurons/frequency channels processed with a decimation factor of 10.

based temporal coding. This learning rule is used to train both unimodal and cross-modal connections. The unimodal connections learn the discriminative features of each modality, whereas cross-modal connections behave as the interlink between the unimodal ensembles. The cross-modal connections allow communication between the unimodal networks to assist each other in classifying the image/audio.

## 3.4 Multimodal Spiking Neural Network

### 3.4.1 Unimodal ensemble

The unimodal ensemble topology for image and audio is shown in Fig. 3.3. It consists of input layer followed by excitatory and inhibitory layer. The input layer is fully connected to the excitatory layer, which in turn is one-to-one connected to the inhibitory layer. The number of neurons in the inhibitory layer is same as the number in the excitatory layer. Each inhibitory neuron is backward connected to all the excitatory neurons except for the one from which it receives a connection from. Thus, the inhibitory layer provides lateral inhibition which discourages simultaneous firing of multiple excitatory neurons and promotes competition among them to learn different input features. To ensure similar firing rates for all neurons in the excitatory layer we employ an adaptive membrane threshold mechanism called homoeostasis [15]. The threshold potential is expressed as $V_{thresh} = V_t + \theta$, where $V_t$ is a constant and $\theta$ is described by

$$\tau_\theta \frac{d\theta}{dt} = -\theta \tag{3.4}$$

$\theta$ increases every time a neuron fires and decays exponentially. If a neuron fires more often, then its threshold potential increases and it requires more inputs to fire again. This ensures that all neurons in the excitatory layer learn unique features and avoids few neurons from dominating the response pattern.

The input layer neurons in the image ensemble generate spikes based on the image pixel intensity. Each input neuron corresponds to a single pixel on the image. The pixel intensity (0-255) is converted into a Poisson spike-train with an average rate of 0-64 Hz. The spike train is feed to the next layer via the fully connected excitatory synapses. The connections from input to excitatory layer are trained with the STDP learning rule explained in section 3.3.2. The forward as well as the backward connections from excitatory to inhibitory layer is fixed before training and is not altered.

The unimodal ensemble for audio has a similar topology like the image ensemble. The recordings are pre-processed according to the Lyon's cochlear model [56] implemented in Slaney's Matlab auditory toolbox [57]. The model maps the mechanical vibration in the cochlea into neural representation. Fig. 3.4 shows the firing probability of different frequency channels for an utterance of '0' based on the cochlea model. The input neurons correspond to the frequency channels and the input spikes are forwarded to the excitatory layer via the fully connected layer. The connections from input to excitatory layer are trained using the STDP learning rule. The connections between excitatory and inhibitory layer have fixed weights, similar to the image ensemble. The connections from input to excitatory layer in both ensembles are referred as unimodal connections, whereas the connections between the two ensembles are referred as cross-modal connections.

### 3.4.2 Multimodal topology

The two unimodal ensembles described in the previous section can work independently to classify image and audio separately. The unimodal networks have drawbacks like limited accuracy, exponential increase in size for linear accuracy increase and the accuracy degrades with noise in inputs. The multimodal approach mitigates some of the issues by each modality assisting the other in making the decision. Also, in many real-world scenarios information is generated with multiple modalities. A talking person generates audio, facial gesture, eye movement along with the image of the person. These multiple modalities can be combined to not only recognize the person but at the same time give some insights into his/her mood and the connotation of his/her speech. The multimodal approach enables the network to utilize the available information to improve the performance instead of discarding or separating it due to network limitations. Fig. 3.3 shows the proposed multimodal topology. The multimodal network is formed by combining the unimodal ensembles and connecting them with cross-modal connections. The cross-modal connections are learned using the STDP algorithm. The connections are formed between neurons of the two excitatory layers which have learned the same input label from their respective modalities. For example, an image and audio of digit '0' is presented to both the networks. Neurons in respective excitatory layers learn the discriminative features of digit '0'. The cross-modal connections are formed between these neurons where both have learned the same label '0' but in different modalities. The STDP algorithm learns these connections as both the networks are presented with same label at same time.

The cross-modal connections are only excitatory, i.e., they invoke activity in neurons with the same label but do not inhibit neurons with different label. This ensures that the normal spiking activity in each ensemble is not inhibited by the cross-modal connections. The cross-modal connections between neurons of different class have low weight values and thus invoke less activity compared to connections between neurons that have learned the same label. The unsupervised training of cross-modal connections differentiates our approach from previous multimodal networks. Instead of learning the connections, the neurons that have learned the same label can be connected manually with a fixed weight. This eliminates the possibility of

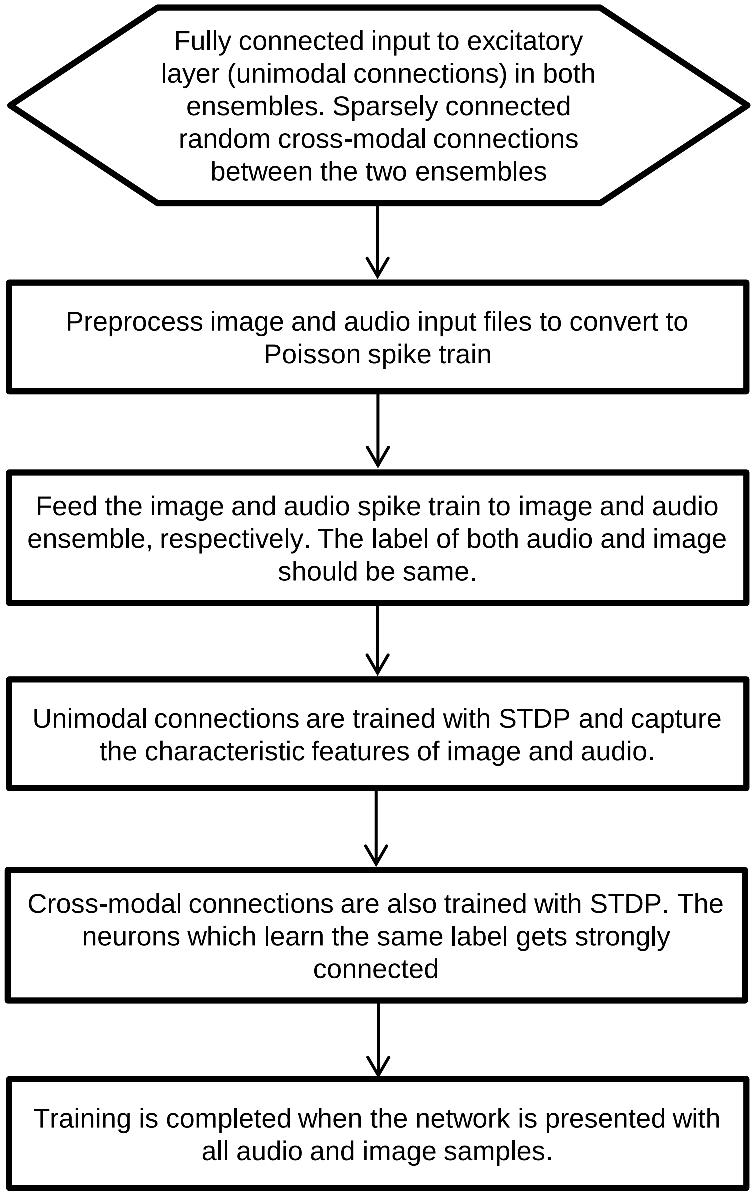

Fully connected input to excitatory layer (unimodal connections) in both ensembles. Sparsely connected random cross-modal connections between the two ensembles
Preprocess image and audio input files to convert to Poisson spike train
Feed the image and audio spike train to image and audio ensemble, respectively. The label of both audio and image should be same.
Unimodal connections are trained with STDP and capture the characteristic features of image and audio.
Cross-modal connections are also trained with STDP. The neurons which learn the same label gets strongly connected
Training is completed when the network is presented with all audio and image samples.

**Figure 3.5.** Flowchart of the overall learning algorithm for multimodal learning.

having any connections between neurons of different label. But self-learning eliminates the overhead of identifying neurons that have learned the same label and connect them using some algorithm. It enables the network to perform online learning and adapt to new input examples. The cross-modal connections carry spikes from image-audio as well as from audio-image ensemble but they are not bi-directional. Half the connections carry spikes from image to audio ensemble and the other half carries spikes from audio to image ensemble (Fig. 3.3).

The neurons in the excitatory layer of both image and audio ensemble can be involved in only one of the three cross-modal operations. The neuron can either send a spike to the other ensemble, or receive a spike from the other ensemble or not participate in cross-modal training. The same neuron cannot receive and send spike because it will introduce a positive feedback and make the system unstable.

Fig. 3.5 shows the flowchart of the overall learning algorithm for both unimodal and cross-modal connections. The unimodal connections are fully connected and initialized with random weights. The cross-modal connections are sparsely connected and also initialized with random weights. The input image and audio samples are preprocessed and converted to Poisson spike train. The unimodal excitatory connections in both the ensembles are trained with STDP. The characteristic features of the input samples are captured in the trained unimodal connections. The spiking activity in one excitatory layer is transferred to another through cross-modal connections. The STDP algorithm potentiates the cross-modal connections between neurons which learn the same label. It is necessary to present the same input label to both the ensembles in order to facilitate the correct cross-modal learning. The cross-modal connections invoke additional activity in neurons with the correct label. The training is completed when the network is presented with all training samples.

## 3.5 Experiments

The multimodal network is simulated in the open source spiking neuron simulator BRIAN implemented in Python [33]. The neurons and synapses are defined by differential equations discussed in section 3.3. The image ensemble is trained with MNIST dataset [28] and the audio ensemble is trained with TI46 speech corpus [58]. The cross-modal connections between the two ensembles are trained along with the unimodal connections.

### 3.5.1 Training & Testing

The excitatory connections in the unimodal ensembles are trained to learn the characteristic features of the input. The cross-modal connections capture the correlation in the spiking activity of the two ensembles and assist each other during testing. The training

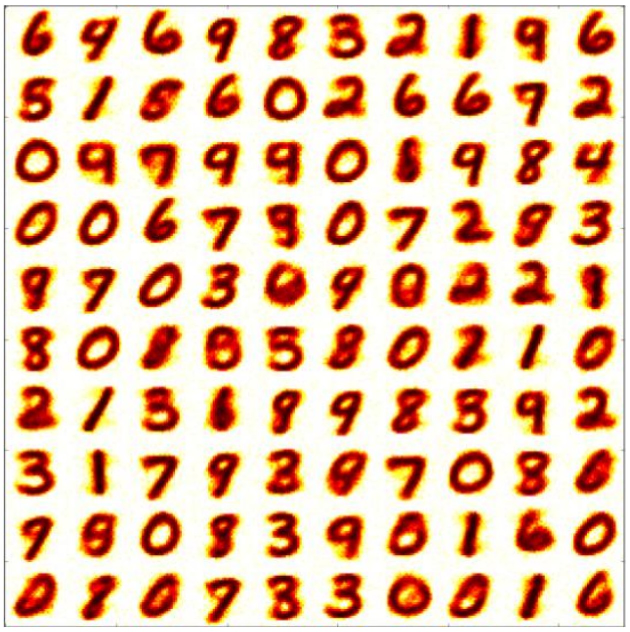


**Figure 3.6.** Rearranged trained weights of the unimodal connections in the image ensemble.

of both unimodal and cross-modal connections is unsupervised. The weight update is calculated based on the spike timing as discussed in section 3.3.2. The input image/audio is converted into a spike train and feed through the input layer as discussed in section 3.4.1. The spiking activity in the excitatory layers is induced by both the unimodal and cross-modal connections. The weights of the cross-modal and unimodal connections are initialized (before training) randomly from a uniform distribution. The unimodal connections are fully-connected, whereas the cross-modal connections are sparsely connected. The sparsity in the cross-modal connections is necessary to limit the spiking activity in the excitatory layers and only form strong connections between correlated neurons. The STDP learning rule decides the weight of the cross-modal connections and forms a stronger connection between neurons that spike together for the same label. The connections between neurons that do not spike for the same label are inhibited. At the end of training, each neuron in the excitatory layer is assigned a label based on the average combined spiking activity over all training inputs.

During testing, the label prediction of the input is performed by observing the spiking activity in both the ensembles. The ensembles are presented with both the audio and image of the same label. The average spiking rate of all the excitatory neurons with the same label in both the ensembles is computed. The label represented by the highest spiking rate is predicted as the input label. The prediction is correct if the actual label matches the predicted one. This is similar to the approach followed in [15], but with single modality.

### 3.5.2 MNIST dataset

MNIST dataset consists of 60,000 training and 10,000 testing images. Each image is a $28 \times 28-$pixel sized grayscale image of digits '0'-'9'. The image ensemble is trained with the MNIST images. There are 784 ($28 \times 28$) input neurons and 100 neurons in the excitatory and inhibitory layer. The input neurons generate a Poisson spike train with an average rate equivalent to the pixel value. The rearranged trained excitatory weights are shown in Fig. 3.6.

### 3.5.3 TI46 speech corpus

TI46 speech corpus contains utterances from both male and female speakers [58]. In this work we only use the digit utterances from all 16 speakers (8 males, 8 females). The dataset is divided into ~1600 training and ~2500 testing utterances of digits '0'-'9'. The audio samples are recorded at a sample rate of 12.5 kHz. The raw sample file of utterance of '0' is shown in Fig. 3.4. The audio samples are processed according to the Lyon's cochlear model implemented in Slaney's Matlab auditory toolbox. The model describes the propagation of sound in the inner ear and the conversion of the acoustical energy into neural representations. The model combines a series of filters that model the traveling pressure waves. The energy in the signal is detected with half wave rectifiers and several stages of automatic gain control. An important characteristic of the cochlea is that energy in the acoustic wave is separated by frequency. Each point in the cochlea responds best to one frequency. The cochlea near its base (where the sound enters) is most sensitive to high frequency and lower frequencies are sensed as the wave travels down the cochlea. In this work, the audio input is processed to be divided into 64 frequency channels and the firing probability of each channel at every time step for an utterance of '0' is shown in Fig. 3.4. The original input sample is decimated

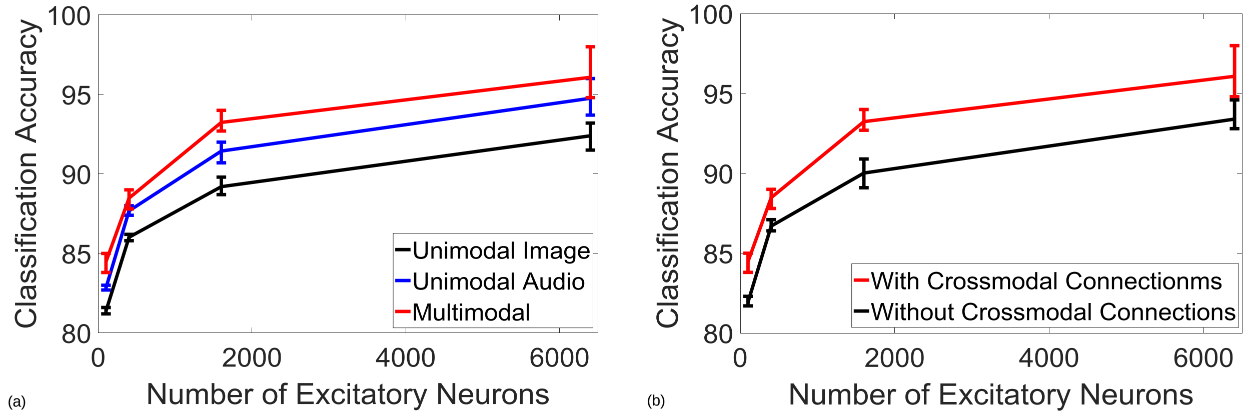


**Figure 3.7.** Classification accuracy with varying number of excitatory neurons for (a) unimodal and multimodal network; and (b) multimodal network with and without cross-modal connections.

(reducing the sampling rate) by a factor of 10 to improve training time. The preprocessed input is presented to the network for 1500 time steps (750 ms). Thus, the recognition speed of our model is 750 ms. The further compression of audio data deteriorates the performance as most of the signal data for the audio is lost. For the unimodal image classification, we could successfully perform the classification in 350 ms with no accuracy degradation. The firing probabilities are converted into a Poisson spike train and fed to the excitatory layer through the input neurons of the audio ensemble. The unsupervised STDP learning rule makes the excitatory connections capture the characteristic features of each label. Unlike image (Fig. 3.6), the learned weights of the audio ensemble do not represent a human readable pattern. The weights learn the frequency domain characteristics and are not shown here.

**Table 3.1.** STDP parameters for unimodal and cross-modal training.

| Parameter | Unimodal (Image) | Unimodal (Audio) | Crossmodal |
|---|---|---|---|
| Learning Rate ($\eta$) | 0.0005 | 0.00003 | 0.001 |
| Time Constant ($\tau$) | 20 ms | 15 ms | 30 ms |
| offset | 0.4 | | |
| $w_{max}$ | 1 | | |
| Exponential dependence ($\mu$) | 0.9 | | |

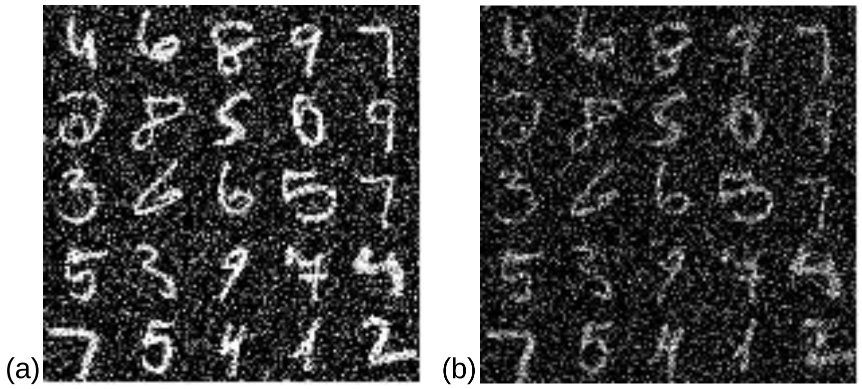
(a)  (b)

**Figure 3.8.** n-MNIST images with (a) AWGN; and (b) reduced contrast and AWGN.

The two-layer topology defined for audio and image ensemble can be trained separately to recognize audio and image, respectively. The introduction of cross-modal connections fuses the two ensembles to make a decision with higher confidence. The cross-modal connections are also trained in an unsupervised manner, therefore, enabling online learning for the multimodal network. The training is performed for 40,000 pair of inputs (1600 audio samples are repeated).

## 3.6 Results & Analyses

In this section we analyze the results of our multimodal network on various parameters. The multimodal approach is compared with the unimodal topology. The effect of noise and subsequent noise suppression by multimodal network is presented. The network is tested for compressed input sizes as well as for inputs with missing modalities. The STDP parameters for both unimodal and cross-modal connections are mentioned in Table 3.1.

### 3.6.1 Multimodal versus Unimodal

The accuracy of the multimodal network is compared with the unimodal topology for varying number of excitatory neurons. The multimodal network has similar total number of neurons compared to unimodal topology. For example, a 100-neuron unimodal image/audio network is compared with a multimodal network with two 50-neuron ensembles. Fig. 3.7(a) shows the classification accuracy with varying number of excitatory neurons for the three topologies. The image (audio) network achieves a highest accuracy of 93.2% (96.0%) with 6400 neurons. The multimodal network with similar size achieves a highest accuracy of 98% for both MNIST and TI46 dataset combined. We performed simulation with 5 different random seed values which are used for weight initialization and cross-modal connectivity. The network achieved an accuracy 95.4%, 94.8%, 98.0%, 97.3%, 94.9% for the five different initializations as shown by the error bars in Fig. 3.7(a). The variations increase for larger networks as more parameters are randomly initialized. The decision of the multimodal network is computed by observing the spiking activity in both ensembles. The average spike rates of similarly tagged neurons in both ensembles are computed. The neurons with the highest spike rate is predicted as the output. The multimodal network performs better than the unimodal topologies due to the cross-modal connections. The cross-modal connections introduce additional spiking activity in neurons of same class and spiking activity in neurons belonging to a different class remain unchanged. This improves the overall spiking activity of the correct class and performs better than unimodal designs. The cross-modal connections assist the network in making the right decision by increasing the spikes for the correct label/class.

### 3.6.2 Multimodal network without cross-modal connections

In section 3.6.1 we mentioned that the superior performance of multimodal network is due to cross-modal connections. To strengthen this point, we compare the multimodal network with and without cross-modal connections. The network with cross-modal connections is trained and tested in the similar way discussed in section 3.6.1. The multimodal network without cross-modal connections is trained as two separate unimodal networks. There is no information exchange between the two ensembles during training and testing. Since the two ensembles are trained independently, the order of input labels in both ensembles is random during training. The testing is performed by computing the spiking activity of both ensembles when presented with inputs of same class. The spike count of neurons in both ensembles with same label is added. The neurons with the highest average spike rate is

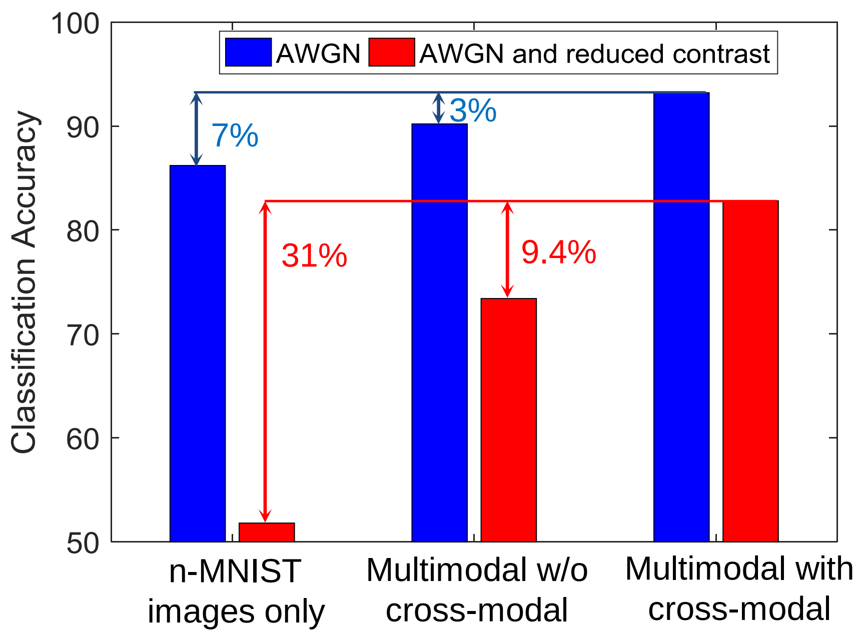


**Figure 3.9.** Classification accuracy of unimodal and multimodal network for noisy MNIST (n-MNIST) dataset.

predicted as the output. Since there are no cross-modal connections, the spiking activity in each ensemble is only due to its own inputs. Fig. 3.7(b) shows the classification accuracy of the networks with and without cross-modal connections for varying number of excitatory neurons. The network with (without) cross-modal connections achieves an accuracy of 98.0% (94.6%). The network with cross-modal connections performs better and the difference in accuracy increases for large number of neurons. This proves that STDP learning rule can correctly identify the correlation between neurons of different modality. The network may perform better with more ensembles receiving inputs in multiple modalities.

### 3.6.3 Testing with noisy data

In this section we compare the multimodal network with and without cross-modal connections for noisy input. The cross-modal connections assist in suppressing the effect of noise on accuracy. The network is evaluated with noisy MNIST (n-MNIST) dataset [59] and audio samples from TI46 speech corpus. The n-MNIST dataset contains images of the handwritten digits from the MNIST dataset with (a) additive white Gaussian noise (AWGN)

(Fig. 3.8(a)) and (b) a combination of AWGN and reduced contrast (Fig. 3.8(b)). The networks are trained with images from n-MNIST and audio samples from TI46 speech corpus. The training and testing are performed in the same way as discussed in section 3.6.1 and 3.6.2. The unimodal network trained with only n-MNIST images achieved an accuracy of 86.2% (51.8%) for images with AWGN (AWGN and reduced contrast) (Fig. 3.9). This is consistent with the results shown in [60]. The multimodal topology without cross-modal assistance improved the accuracy to 90.2% (73.4%) for images with AWGN (AWGN and reduced contrast). The network with cross-modal connections further improved the accuracy to 93.2% (82.8%) for images with AWGN (AWGN and reduced contrast). The cross-modal connections boost the accuracy by more than 30% as shown in Fig. 3.9. The multimodal topology has an added advantage of denoising by assisting the network with another modality to make a decision. This is similar to how humans recognize a person standing at a distance and talking. The voice adds confidence in recognizing the person even though the image is not clearly visible.

### 3.6.4 Testing with missing modality

In this section we evaluate the performance of our proposed multimodal network with cross-modal connections when tested with only one modality. The network is trained with both modalities, but while testing the network is presented with only one modality and the input for another modality is removed. This is similar to how infants respond to sound symbolism by associating a shape with sound [61]. For example, learning to recognize 'apple'. During learning both the image and audio of 'apple' is presented and humans learn to associate the image and audio together. Later, if we hear the utterance of 'apple' we are able to recollect the image in our brains, even though we do not see the image. Presumably, this is possible due to the cross-modal connections in our brains which enables communication between areas of different modalities.

Fig. 3.10 shows the classification accuracy with varying number of excitatory neurons when tested with only one modality. The network achieved an accuracy of 96.8% when tested with only audio input (image missing) and 93.9% when tested with only image input

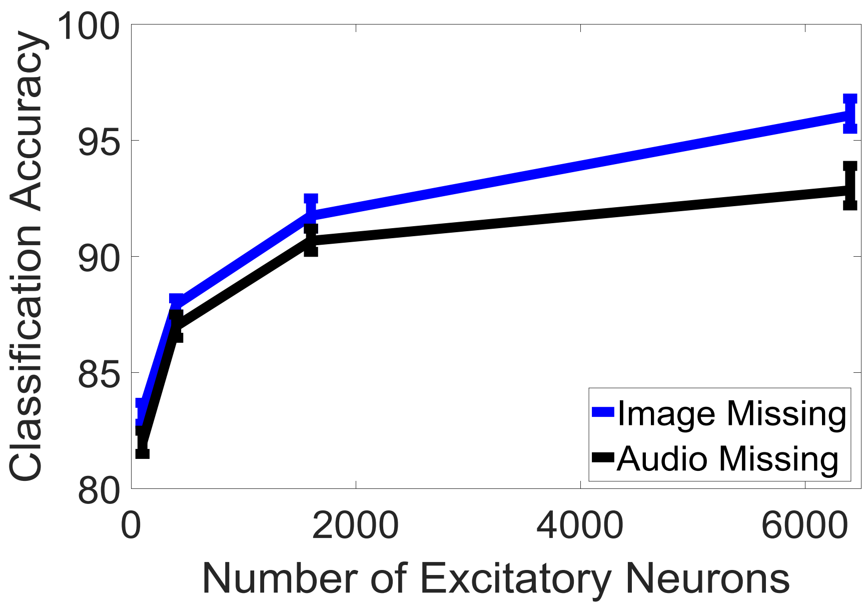


**Figure 3.10.** Classification accuracy of multimodal network with cross-modal connections when tested with only one modality.

(audio missing). The results are slightly better when compared with unimodal networks. The increase in accuracy is a result of well learned cross-modal connections. The cross-modal connections introduce activity in the excitatory layer of the missing modality resulting in higher overall spikes for the correct label. Ideally, there should be no spiking activity in the ensemble of the missing input, but the well learned cross-modal connections transfer spikes from one ensemble to another. This shows that cross-modal connections can activate areas of the network which do not receive the inputs directly.

### 3.6.5 Effect of Lateral Inhibition and Homoeostasis

As described in section 3.4.1, we employ lateral inhibition and homoeostasis while training our unimodal and multimodal networks. To test the necessity of these techniques we train an image unimodal network without lateral inhibition and homoeostasis and compare the testing accuracy. The unimodal network of 100 excitatory neurons with lateral inhibition and homoeostasis achieved an accuracy of 81.6% (Fig. 3.7 (a)). The same network when trained with only homoeostasis achieved an accuracy of 77.2% and the accuracy drops to 68.3% when both lateral inhibition and homoeostasis is not applied during training. Thus

Table 3.2. Comparison of our work with other unimodal and multimodal networks.

| Model | Type | # Layers | Learning Type | Modality | Dataset | Accuracy |
|---|---|---|---|---|---|---|
| [15] | Spiking | 2 | Unsupervised | Unimodal | MNIST | 95.00% |
| [20] | Spiking | 3 | Unsupervised | Unimodal | MNIST (subset) | 81.90% |
| [43] | Spiking | 3 | Supervised | Unimodal | MNIST | 98.06% |
| [45] | Spiking | 6 | Supervised | Unimodal | MNIST | 98.40% |
| [46] | Spiking | 3 | Supervised | Unimodal | TI46 (subset) | 95.25% |
| [47] | Spiking | Reservoir Network | Supervised | Unimodal | TI46 (subset) | 99.79% |
| [53] | Non-Spiking | 8 | Supervised | Multimodal | Washington RGB-D Object | 91.30% |
| [52] | Non-Spiking | 10+ | Supervised | Multimodal | Washington RGB-D Object | 86.90% |
| **This work** | **Spiking** | **2** | **Unsupervised** | **Unimodal** | **MNIST** | **93.20%** |
| **This work** | **Spiking** | **2** | **Unsupervised** | **Unimodal** | **TI46** | **96.00%** |
| **This work** | **Spiking** | **2** | **Unsupervised** | **Multimodal** | **MNIST & TI46** | **98.00%** |

homoeostasis is necessary during training and lateral inhibition further improves the learning process.

### 3.6.6 Comparison with other models

The performance of our proposed network is compared with other unimodal and multimodal networks (Table 3.2). The multimodal networks are non-spiking ANN trained with supervised learning. Our proposed network performs reasonably well and outperforms many previous unimodal approaches. It is difficult to directly compare the accuracies with other multimodal networks since they are ANN and tested on different datasets. This work is the first to train a SNN with multimodal inputs and evaluate it on the digit recognition task.

## 3.7 Conclusions

In this work, we present a synergistic learning framework for SNN that can be trained with multiple modalities and unsupervised learning. Our method combines the unimodal ensembles with cross-modal connections and makes a collective decision to recognize the inputs. The unimodal and cross-modal connections are trained simultaneously. The correlation between neurons of different modalities is captured in the cross-modal connections. The multimodal approach not only improves the accuracy but also makes the network noise tolerant. The ability to train the entire network with unsupervised learning can be an advantage in many applications since unlabeled data is much readily available. The multimodal topology is compared with unimodal topology for image and audio datasets. The collaborative learning results in an accuracy improvement of up to 4.8% for normal inputs and 31% for noisy inputs. The multimodal approach also reduced the training time by $2.2X$ compared to unimodal topology for similar classification accuracy.

## 3.8 Discussion

The results in this chapter show that the cross-modal connections improve the accuracy of the multimodal network. The cross-modal connections enable communication between two ensembles by connecting neurons that learn the same label. The number of cross-modal connections is a critical parameter and affects the classification accuracy. Too few connections are not able to introduce the required activity, and too many connections may introduce activity in neurons of different label. The cross-modal connections can be formed in two ways: learned with STDP or manually connecting the neurons from the two ensembles that have learned the same label. The connections learned with STDP require no user interference and can easily adapt to new inputs. On the other hand, connecting neurons manually avoids the problem of forming a connection between neurons that have learned different labels. Fig. 3.11 shows the effect of number of cross-modal connections on classification accuracy for both STDP learned cross-modal connections and manual connections. The plot is shown for a 100-neuron multimodal network with cross-modal connections. For STDP based learning the cross-modal connections are formed randomly and initialized with random weights. The STDP algorithm potentiates the connections between neurons that have learned the same label in both ensembles and depresses other connections. The learning is limited to connections that have been randomly decided. The network achieves a highest classification

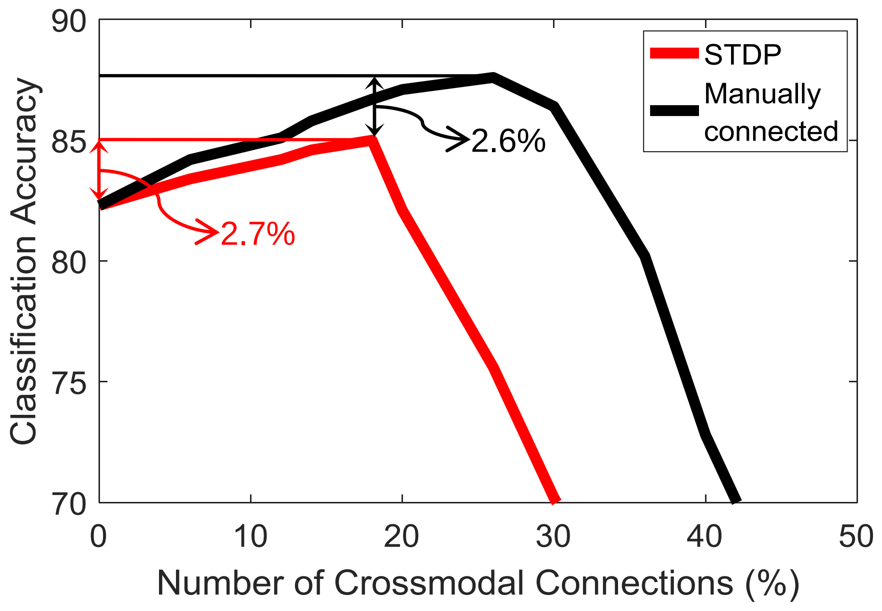


**Figure 3.11.** Classification accuracy of multimodal network (100 neurons) for varying number of cross-modal connections. The cross-modal connections are represented as the % of total number of connections. The highest accuracy is achieved with 18% cross-modal connections.

accuracy of 85% with 18% cross-modal connections compared to 82.3% without cross-modal connections. As the number of random cross-modal connections is increased, the neurons that have learned different label gets connected. This reduces the classification accuracy as the spiking activity of the entire network is increased rather than just the neurons with the correct label. The cross-modal connections which are manually connected are formed after the network has been trained for 5000 examples.

The training starts with no cross-modal coupling and the two ensembles are trained independently. After 5000 examples, the cross-modal connections are formed between neurons that have learned the same input label in both the ensembles. Even though the connections are manually formed, the training is still unsupervised as we do not use the correct input labels. The network achieves a classification accuracy of 87.6% which is 2.6% higher than STDP based connections. The increase in accuracy is due to the elimination of connections between neurons that have learned different labels. The accuracy should keep increasing as more neurons with similar labels get connected. But, the accuracy goes down beyond 26% connections (Fig. 3.11). The neurons are tagged with a label for which it has the highest

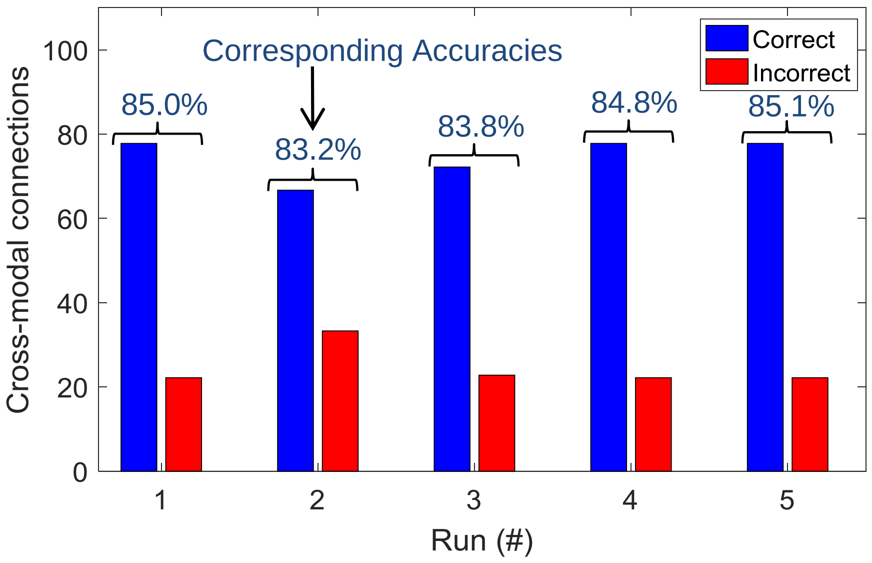


**Figure 3.12.** Number of correct and incorrect cross-modal connections for randomly initialized network of 100-neurons with 18% cross-modal connectivity for five different runs. Correct connections are the ones between neurons with the same label in both ensembles.

spiking activity among all input labels. A single neuron may capture characteristic features of multiple input labels but gets tagged for which it had the highest spikes. Therefore, the neurons in the ensemble do not exclusively spike for the label they are tagged with but also spike for other input labels. So, the accuracy drops for high number of connections as neurons show higher spiking activity for incorrect labels.

As mentioned before, the STDP based cross-modal connections are initialized randomly and different set of random connections should give varying results. Fig. 3.12 shows the % of correct and incorrect cross-modal connections along with classification accuracy for five runs of a 100-neuron network with 18% random cross-modal connectivity. The correct connections are the ones where the connected neurons have learned the same label in both ensembles, whereas incorrect connections are between neurons that have learned different labels. In each run the network was initialized with different set of cross-modal connections by adjusting the seed of the random number generator. The chart shows that STDP algorithm is able to connect more than 70% of the connections correctly resulting in a classification accuracy of ∼85%. This also explains the increase in accuracy compared to a network with no cross-modal connections. The correctly connected cross-modal connections increase the spiking activity for the correct label resulting in higher accuracy. The amount of information exchanged between the two ensembles is very critical and controlled by the number of cross-modal connections. We have tried to make an effort to emulate the multimodal learning in human brain into artificial SNNs. In the future we would like to include more modalities and identify more features like connotation and mood of the speaker.

# 4. ENABLING DEEP SPIKING NEURAL NETWORKS WITH HYBRID CONVERSION AND SPIKE TIMING DEPENDENT BACKPROPAGATION

## 4.1 Introduction

In recent years, Spiking Neural Networks (SNNs) have shown promise towards enabling low-power machine intelligence with event-driven neuromorphic hardware. Founded on bio-plausibility, the neurons in an SNN compute and communicate information through discrete binary events (or ‘spikes’) a significant shift from the standard artificial neural networks (ANNs), which process data in a real-valued (or analog) manner. The binary all-or-nothing spike-based communication combined with sparse temporal processing precisely make SNNs a low-power alternative to conventional ANNs. With all its appeal for power efficiency, training SNNs still remains a challenge. The discontinuous and non-differentiable nature of a spiking neuron (generally, modeled as leaky-integrate-and-fire (LIF), or integrate-and-fire (IF)) poses difficulty to conduct gradient descent based backpropagation. Practically, SNNs still lag behind ANNs, in terms of performance or accuracy, in traditional learning tasks. Consequently, there has been several works over the past few years that propose different learning algorithms or learning rules for implementing deep convolutional SNNs for complex visual recognition tasks [62]–[64]. Of all the techniques, conversion from ANN-to-SNN [6], [44], [63], [65] has yielded state-of-the-art accuracies matching deep ANN performance for Imagenet dataset on complex architectures (such as, VGG [66] and ResNet [67] ). In conversion, we train an ANN with ReLU neurons using gradient descent and then convert the ANN to an SNN with IF neurons by using suitable threshold balancing [6]. But, SNNs obtained through conversion incur large latency of $2000-2500$ time steps (measured as total number of time steps required to process a given input image[1]). The term ‘time step’ defines an unit of time required to process a single input spike across all layers and represents the network latency. The large latency translates to higher energy consumption during inference, thereby, diminishing the efficiency improvements of SNNs over ANNs. To reduce

[1]↑SNNs process Poisson rate-coded input spike trains, wherein, each pixel in an image is converted to a Poisson-distribution based spike train with the spiking frequency proportional to the pixel value

the latency, spike-based backpropagation rules have been proposed that perform end-to-end gradient descent training on spike data. In spike-based backpropagation methods, the non-differentiability of the spiking neuron is handled by either approximating the spiking neuron model as continuous and differentiable [68] or by defining a surrogate gradient as a continuous approximation of the real gradient [11], [12], [69]. Spike-based SNN training reduces the overall latency by $\sim 10\times$ (for instance, $200-250$ time steps required to process an input [70]) but requires more training effort (in terms of total training iterations) than conversion approaches. A single feed-forward pass in ANN corresponds to multiple forward passes in SNN which is proportional to the number of time steps. In spike-based backpropagation, the backward pass requires the gradients to be integrated over the total number of time steps that increases the computation and memory complexity. The multiple-iteration training effort with exploding memory requirement (for backward pass computations) has limited the applicability of spike-based backpropagation methods to small datasets (like CIFAR10) on simple few-layered convolutional architectures.

In this work, we propose a hybrid training technique which combines ANN-SNN conversion and spike-based backpropagation that reduces the overall latency as well as decreases the training effort for convergence. We use ANN-SNN conversion as an initialization step followed by spike-based backpropagation incremental training (that converges to optimal accuracy with few epochs due to the precursory initialization). Essentially, our hybrid approach of taking a converted SNN and incrementally training it using backpropagation yields improved energy-efficiency as well as higher accuracy than a model trained from scratch with only conversion or only spike-based backpropagation.

In summary, this work makes the following contributions:

- We introduce a hybrid computationally-efficient training methodology for deep SNNs. We use the weights and firing thresholds of an SNN converted from an ANN as the initialization step for spike-based backpropagation. We then train this initialized network with spike-based backpropagation for few epochs to perform inference at a reduced latency or time steps.

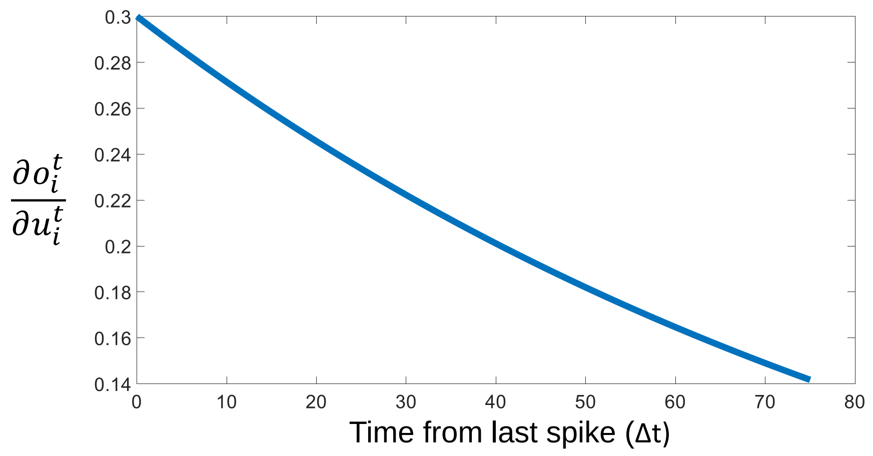


**Figure 4.1.** Surrogate gradient of the spiking neuron activation function (Eq. 4.11). $\alpha = 0.3, \beta = 0.01$. The gradient is computed for each neuron and $\Delta t$ defines the time difference between current simulation time and the last spike time of the neuron. For example, if a neuron spikes at $t_s = 12$ its gradient will be maximum at $t = 12(\Delta t = 0)$ and gradually decrease for later time steps. If the same neuron spikes later at $t_s = 24$ its previous spike history will be overwritten and the gradient computation for $t = 24$ onward will only consider the most recent spike. This avoids the overhead of storing all the spike history in memory.

- We propose a novel spike time-dependent backpropagation (STDB, a variant of standard spike-based backpropagation) that computes surrogate gradient using neuron's spike time. The parameter update is triggered by the occurrence of spike and the gradient is computed based on the time difference between the current time step and the most recent time step the neuron generated an output spike. This is motivated from the Hebb's principle which states that the plasticity of a synapse is dependent on the spiking activity of the neurons connected to the synapse.

- Our hybrid approach with the novel surrogate gradient descent allows training of large-scale SNNs without exploding memory required during spike-based backpropagation. We evaluate our hybrid approach on large SNNs (VGG, ResNet-like architectures) on Imagenet, CIFAR datasets and show near iso-accuracy compared to similar ANNs and converted SNNs at lower compute cost and energy.

## 4.2 Spike Timing Dependent Backpropagation (STDB)

In this section, we describe the spiking neuron model, derive the equations for the proposed surrogate gradient based learning, present the weight initialization method for SNN, discuss the constraints applied for ANN-SNN conversion, and summarize the overall training methodology.

### 4.2.1 Leaky Integrate and Fire (LIF) Neuron Model

The neuron model defines the dynamics of the neuron's internal state and the trigger for it to generate a spike. The differential equation

$$\tau \frac{dU}{dt} = -(U - U_{rest}) + RI \tag{4.1}$$

is widely used to characterize the leaky-integrate-and-fire (LIF) neuron model where, $U$ is the internal state of the neuron referred as the membrane potential, $U_{rest}$ is the resting potential, $R$ and $I$ are the input resistance and the current, respectively. The above equation is valid when the membrane potential is below the threshold value ($V$). The neuron generates an output spike when $U \geqslant V$ and $U$ is reduced to the reset potential. This representation is described in continuous domain and more suitable for biological simulations. We modify the equation to be evaluated in a discrete manner in the Pytorch framework [69]. The iterative model for a single post-neuron is described by

$$u_{\mathrm{i}}^{t} = \lambda u_{\mathrm{i}}^{t-1} + \sum_{\mathrm{j}} w_{\mathrm{ij}} o_{\mathrm{j}}^{t} - v o_{\mathrm{i}}^{t-1} \tag{4.2}$$

$$o_{\mathrm{i}}^{t-1} = \begin{cases} 1, & \text{if } u_{\mathrm{i}}^{t-1} > v \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

where $u$ is the membrane potential, subscript i and j represent the post- and pre-neuron, respectively, superscript $t$ is the time step, $\lambda$ is a constant ($< 1$) responsible for the leak in membrane potential, $w$ is the weight connecting the pre- and post-neuron, $o$ is the binary

**Algorithm 1** ANN-SNN conversion: initialization of weights and threshold voltages

```
Input: Trained ANN model ($A$), SNN model ($N$), Input ($X$)
// Copy ann weights to snn
for $l$=1 to $L$ do
    $N_l.W \leftarrow A_l.W$
// Initialize threshold voltage to 0
$V \leftarrow [0, \cdots, 0]_{L-1}$
for $l$=1 to $L$-1 do
    $v \leftarrow 0$
    for $t$=1 to $T$ do
        $O_0^t \leftarrow PoissonGenerator(X)$
        for $k$=1 to $l$ do
            if $k < l$ then
                // Forward (Algorithm 4)
            else
                // Pre-nonlinearity ($A$)
                $A \leftarrow N_l(O_{k-1}^t)$
                if $max(A) > v$ then
                    $v \leftarrow max(A)$
    $V[l] \leftarrow v$
```

output spike, and $v$ is the firing threshold potential. The right hand side of Equation 4.2 has three terms: the first term calculates the leak in the membrane potential from the previous time step, the second term integrates the input from the previous layer and adds it to the membrane potential, and the third term which is outside the summation reduces the membrane potential by the threshold value if a spike is generated. This is known as soft reset as the membrane potential is lowered by $v$ compared to hard reset where the membrane potential is reduced to the reset value. Soft reset enables the spiking neuron to carry forward the excess potential above the firing threshold to the following time step, thereby minimizing information loss.

### 4.2.2 Spike Timing Dependent Backpropagation (STDB) Learning Rule

The neuron dynamics (Equation 4.2) show that the neuron's state at a particular time step recurrently depends on its state in previous time steps. This introduces implicit recurrent connections in the network [12]. Therefore, the learning rule has to perform the temporal

**Algorithm 2** Initialize the neuron parameters. Membrane potential ($U$), last spike time ($S$), dropout mask ($M$). The initialization is performed once for every mini-batch.

```
Input: Input($X$), network model($N$)
$b\_size = X.b\_size$
$h = X.height$
$w = X.width$
for $l$=1 to $L$ do
    if $isintance(N_l, Conv)$ then
        $U_l = zeros(b\_size, N_l.out, h, w)$
        $S_l = ones(b\_size, N_l.out, h, w) * (-1000)$
    else if $isintance(N_l, Linear)$ then
        $U_l = zeros(b\_size, N_l.out)$
        $S_l = ones(b\_size, N_l.out) * (-1000)$
    else if $isintance(N_l, Dropout)$ then
        // Generate the dropout map that will be fixed for all time steps
        $M_l = N_l(ones(U_{l-1}.shape))$
    else if $isintance(N_l, AvgPool)$ then
        // Reduce the width and height after average pooling layer
        $h = h//kernel\_size$
        $w = w//kernel\_size$
```

credit assignment along with the spatial credit assignment. Credit assignment refers to the process of assigning credit or blame to the network parameters according to their contribution to the loss function. Spatial credit assignment identifies structural network parameters (like weights), whereas temporal credit assignment determines which past network activities contributed to the loss function. Gradient-descent learning solves both credit assignment problem: spatial credit assignment is performed by distributing error spatially across all layers using the chain rule of derivatives, and temporal credit assignment is done by unrolling the network in time and performing backpropagation through time (BPTT) using the same chain rule of derivatives [71]. In BPTT, the network is unrolled for all time steps and the final output is computed as the sum of outputs from each time step. The loss function is defined on the summed output.

The dynamics of the neuron in the output layer is described by Equation (4.4), where the leak part is removed ($\lambda = 1$) and the neuron only integrates the input without firing. This eliminates the difficulty of defining the loss function on spike count [70].

$$u_{\mathrm{i}}^{t} = u_{\mathrm{i}}^{t-1} + \sum_{\mathrm{j}} w_{\mathrm{ij}} o_{\mathrm{j}} \tag{4.4}$$

The number of neurons in the output layer is the same as the number of categories in the classification task. The output of the network is passed through a softmax layer that outputs a probability distribution. The loss function is defined as the cross-entropy between the true output and the network's predicted distribution.

$$L = -\sum_{\mathrm{i}} y_{\mathrm{i}} log(p_{\mathrm{i}}) \tag{4.5}$$

$$p_{\mathrm{i}} = \frac{\mathrm{e}^{u_{\mathrm{i}}^{T}}}{\sum_{k=1}^{N} \mathrm{e}^{u_{k}^{T}}} \tag{4.6}$$

$L$ is the loss function, $y$ the true output, $p$ the prediction, $T$ the total number of time steps, $u^T$ the accumulated membrane potential of the neuron in the output layer from all time steps, and $N$ the number of categories in the task. For deeper networks and large number of time steps the truncated version of the BPTT algorithm is used to avoid memory issues. In the truncated version the loss is computed at some time step $t'$ before T based on the potential accumulated till $t'$. The loss is backpropagated to all layers and the loss gradients are computed and stored. At this point, the history of the computational graph is cleaned to save memory. The subsequent computation of loss gradients at later time steps $(2t', 3t', ...T)$ are summed together with the gradient at $t'$ to get the final gradient. The optimizer updates the parameters at $T$ based on the sum of the gradients. Gradient descent learning has the objective of minimizing the loss function. This is achieved by backpropagating the error and updating the parameters opposite to the direction of the derivative. The derivative of the loss function w.r.t. to the membrane potential of the neuron in the final layer is described by,

$$\frac{\partial L}{\partial u_{\mathrm{i}}^{T}} = p_{\mathrm{i}} - y_{\mathrm{i}} \tag{4.7}$$

**Algorithm 3** Training an SNN with surrogate gradient computed with spike timing. The network is composed of $L$ layers. The training proceeds with mini-batch size ($batch\_size$)

```
Input: Mini-batch of input ($X$) - target ($Y$) pairs, network model ($N$), initial weights ($W$), threshold voltage ($V$)
$U, S, M = InitializeNeuronParameters(X)$ [Algorithm 2]
 // Forward propagation
 for $t$=1 to $T$ do
     $O_0^t = PoissonGenerator(X)$
     for $l$=1 to $L$-1 do
         if $isintance(N_l, [Conv, Linear])$ then
             // accumulate the output of previous layer in $U$, soft reset when spike occurs
             $U_l^t = \lambda U_l^{t-1} + W_l O_{l-1}^t - V_l * O_l^{t-1}$
             // generate the output (+1) if $U$ exceeds $V$
             $O_l^t = STDB(U_l^t, V_l, t)$
             // store the latest spike times for each neuron
             $S_l^t[O_l^t == 1] = t$
         else if $isintance(N_l, AvgPool)$ then
             $O_l^t = N_l(O_{l-1}^t)$
         else if $isintance(N_l, Dropout)$ then
             $O_l^t = O_{l-1}^t * M_l$
     $U_L^t = \lambda U_L^{t-1} + W_L O_{L-1}^t$
// Backward Propagation
 Compute $\frac{\partial L}{\partial U_L}$ from the cross-entropy loss function using BPTT
 for $t$=$T$ to 1 do
     for $l$=$L$-1 to 1 do
         Compute $\frac{\partial L}{\partial O_l^t}$ based on if $N_l$ is linear, conv, pooling, etc.
         $\frac{\partial L}{\partial U_l^t} = \frac{\partial L}{\partial O_l^t}\frac{\partial O_l^t}{\partial U_l^t} = \frac{\partial L}{\partial O_l^t} * \alpha e^{-\beta S_l^t}$
```

To compute the gradient at current time step, the membrane potential at last time step ($u_{\mathrm{i}}^{t-1}$ in Equation 4.4) is considered as an input quantity. Therefore, gradient descent updates the network parameters $W_{\mathrm{ij}}$ of the output layer as,

$$W_{\mathrm{ij}} = W_{\mathrm{ij}} - \eta \Delta W_{\mathrm{ij}} \tag{4.8}$$

$$\Delta W_{\mathrm{ij}} = \sum_t \frac{\partial L}{\partial W_{\mathrm{ij}}^t} = \sum_t \frac{\partial L}{\partial u_{\mathrm{i}}^T} \frac{\partial u_{\mathrm{i}}^T}{\partial W_{\mathrm{ij}}^t} = \frac{\partial L}{\partial u_{\mathrm{i}}^T} \sum_t \frac{\partial u_{\mathrm{i}}^T}{\partial W_{\mathrm{ij}}^t} \tag{4.9}$$

where $\eta$ is the learning rate, and $W_{\mathrm{ij}}^t$ represents the copy of the weight used for computation at time step $t$. In the output layer the neurons do not generate a spike, and hence, the issue of non-differentiability is not encountered. The update of the hidden layer parameters is described by,

$$\Delta W_{\mathrm{ij}} = \sum_t \frac{\partial L}{\partial W_{\mathrm{ij}}^t} = \sum_t \frac{\partial L}{\partial o_{\mathrm{i}}^t} \frac{\partial o_{\mathrm{i}}^t}{\partial u_{\mathrm{i}}^t} \frac{\partial u_{\mathrm{i}}^t}{\partial W_{\mathrm{ij}}^t} \tag{4.10}$$

where $o_{\mathrm{i}}^t$ is the thresholding function (Equation 4.3) whose derivative w.r.t to $u_{\mathrm{i}}^t$ is zero everywhere and not defined at the time of spike. The challenge of discontinuous spiking nonlinearity is resolved by introducing a surrogate gradient which is the continuous approximation of the real gradient.

$$\frac{\partial o_{\mathrm{i}}^t}{\partial u_{\mathrm{i}}^t} = \alpha \mathrm{e}^{-\beta \Delta t} \tag{4.11}$$

where $\alpha$ and $\beta$ are constants, $\Delta t$ is the time difference between the current time step ($t$) and the last time step the post-neuron generated a spike ($t_s$). It is an integer value whose range is from zero to the total number of time steps ($T$).

$$\Delta t = (t - t_s), 0 < \ \Delta t < T \ , \ \Delta t \ \epsilon \ \mathbb{Z} \tag{4.12}$$

The values of $\alpha$ and $\beta$ are selected depending on the value of $T$. If $T$ is large $\beta$ is lowered to reduce the exponential decay so a spike can contribute towards gradients for later time steps. The value of $\alpha$ is also reduced for large $T$ because the gradient can propagate through many time steps. The gradient is summed at each time step and thus a large $\alpha$ may lead to exploding gradient. The surrogate gradient can be pre-computed for all values of $\Delta t$ and

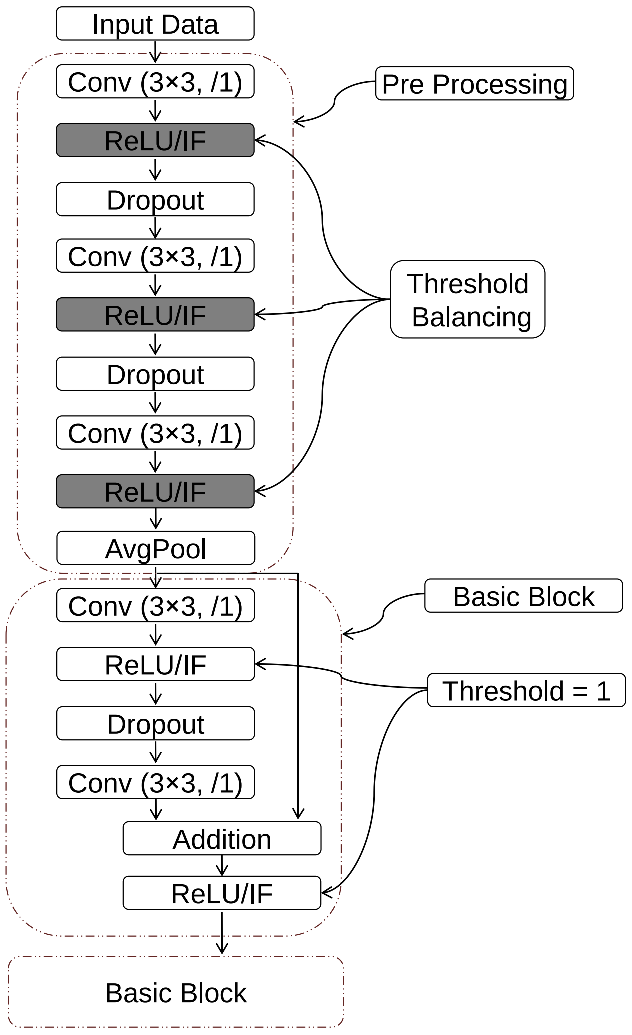


**Figure 4.2.** Residual architecture for SNN

stored in a look-up table for faster computation. The parameter updates are triggered by the spiking activity but the error gradients are still non-zero for time steps following the spike time. This enables the algorithm to avoid the 'dead neuron' problem, where no learning happens when there is no spike. Fig. 4.1 shows the activation gradient for different values of $\Delta t$, the gradient decreases exponentially for neurons that have not been active for a long time. In Hebbian models of biological learning, the parameter update is activity dependent. This is experimentally observed in spike-timing-dependent plasticity (STDP) learning rule which modulates the weights for pair of neurons that spike within a time window [72].

## 4.3 SNN Weight Initialization

A prevalent method of constructing SNNs for inference is ANN-SNN conversion [6], [44]. Since the network is trained with analog activations it does not suffer from the non-differentiablity issue and can leverage the training techniques of ANNs. The conversion process has a major drawback: it suffers from long inference latency (∼2500 time steps) as mentioned in Section 4.1. As there is no provision to optimize the parameters after conversion based on spiking activity, the network can not leverage the temporal information of the spikes. In this work, we propose to use the conversion process as an initialization technique for STDB. The converted weights and thresholds serve as a good initialization for the optimizer and the STDB learning rule is applied for temporal and spatial credit assignment.

Algorithm 1 explains the ANN-SNN conversion process. The threshold voltages in SNN needs to be adjusted based on the ANN weights. In [6], the authors showed two ways to achieve this: weight-normalization and threshold-balancing. In weight-normalization the weights are scaled by a normalization factor and threshold is set to 1, whereas in threshold-balancing the weights are unchanged and the threshold is set to the normalization factor. Both have a similar effect and either can be used to set the threshold. We employ the threshold-balancing method and the normalization factor is calculated as the maximum output of the corresponding convolution/linear layer in SNN. The maximum is calculated over a mini-batch of input for all time steps.

There are several constraints imposed on training the ANN for the conversion process [6], [44]. The neurons are trained without the bias term because the bias term in SNN has an indirect effect on the threshold voltage which increases the difficulty of threshold balancing and the process becomes more prone to conversion loss. The absence of bias term eliminates the use of Batch Normalization [73] as a regularizer in ANN since it biases the input of each layer to have zero mean. As an alternative, Dropout [74] is used as a regularizer for both ANN and SNN training. The implementation of Dropout in SNN is further discussed in Section 4.5. The pooling operation is widely used in ANN to reduce the convolution map size. There are two popular variants: max pooling and average pooling [75]. Max (Average) pooling outputs the maximum (average) value in the kernel space of the neuron's activations. In SNN, the activations are binary and performing max pooling will result in significant information loss for the next layer, so we adopt the average pooling for both ANN and SNN [44].

## 4.4 Network Architectures

In this section, we describe the changes made to the VGG [66] and residual architecture [67] for hybrid learning and discuss the process of threshold computation for both the architectures.

### 4.4.1 VGG Architecture

The threshold balancing is performed for all layers except the input and output layer in a VGG architecture. For every hidden convolution/linear layer the maximum input[2] to the neuron is computed over all time steps and set as threshold for that layer. The threshold assignment is done sequentially as described in Algorithm 1. The threshold computation for all layers can not be performed in parallel (in one forward pass) because in the forward method (Algorithm 4) we need the threshold at each time step to decide if the neuron should spike or not.

### 4.4.2 Residual Architecture

Residual architectures introduce shortcut connections between layers that are not next to each other. In order to minimize the ANN-SNN conversion loss various considerations were made by [6]. The original residual architecture proposed by [67] uses an initial convolution layer with wide kernel (7×7, stride 2). For conversion, this is replaced by a pre-processing block consisting of a series of three convolution layer (3×3, stride 1) with dropout layer in between (Fig. 4.2). The threshold balancing mechanism is applied to only these three layers and the layers in the basic block have unity threshold.

[2]↑input to a neuron is the weighted sum of spkies from pre-neurons $\sum_j w_{ij} o_j$

**Table 4.1.** Classification results (Top-1) for CIFAR10, CIFAR100 and ImageNet data sets. Column-1 shows the network architecture. Column-2 shows the ANN accuracy when trained under the constraints as described in Section 4.3. Column-3 shows the SNN accuracy for $T = 2500$ when converted from a ANN with threshold balancing. Column-4 shows the performance of the same converted SNN with lower time steps and adjusted thresholds. Column-5 shows the performance after training the Column-4 network with STDB for less than 20 epochs.

| Architecture | ANN | ANN-SNN Conversion ($T = 2500$) | ANN-SNN Conversion (reduced time steps) | Hybrid Training (ANN-SNN Conversion + STDB) |
|---|---|---|---|---|
| CIFAR10 | | | | |
| VGG5 | 87.88% | 87.64% | 84.56% ($T = 75$) | 86.91% ($T = 75$) |
| VGG9 | 91.45% | 90.98% | 87.31% ($T = 100$) | 90.54% ($T = 100$) |
| VGG16 | 92.81% | 92.48% | 90.2% ($T = 100$) | 91.13% ($T = 100$) |
| ResNet8 | 91.35% | 91.12% | 89.5% ($T = 200$) | 91.35% ($T = 200$) |
| ResNet20 | 93.15% | 92.94% | 91.12% ($T = 250$) | 92.22% ($T = 250$) |
| CIFAR100 | | | | |
| VGG11 | 71.21% | 70.94% | 65.52% ($T = 125$) | 67.87% ($T = 125$) |
| ImageNet | | | | |
| ResNet34 | 70.2% | 65.1% | 56.87% ($T = 250$) | 61.48% ($T = 250$) |
| VGG16 | 69.35% | 68.12% | 62.73% ($T = 250$) | 65.19% ($T = 250$) |

## 4.5 Overall Training Algorithm

Algorithm 1 defines the process to initialize the parameters (weights, thresholds) of SNN based on ANN-SNN conversion. Algorithm 2 and 4 show the mechanism of training the SNN with STDB. Algorithm 2 initializes the neuron parameters for every mini-batch, whereas Algorithm 4 performs the forward and backward propagation and computes the credit assignment. The threshold voltage for all neurons in a layer is same and is not altered in the training process. For each dropout layer we initialize a mask ($M$) for every mini-batch of inputs. The function of dropout is to randomly drop a certain number of inputs in order to avoid overfitting. In case of SNN, inputs are represented as a spike train and we want to keep the dropout units same for the entire duration of the input. Thus, a random mask ($M$) is initialized (Algorithm 2) for every mini-batch and the input is element-wise multiplied with the mask to generate the output of the dropout layer [70]. The Poisson generator function outputs a Poisson spike train with rate proportional to the pixel value in the input. A random number is generated at every time step for each pixel in the input image. The random number is compared with the normalized pixel value and if the random number is less than the pixel value an output spike is generated. This results in a Poisson spike train with rate equivalent to the pixel value if averaged over a long time. The weighted sum of the input is accumulated in the membrane potential of the first convolution layer. The STDB function compares the membrane potential and the threshold of that layer to generate an output spike. The neurons that output a spike their corresponding entry in $S$ is updated with current time step ($t$). The last spike time is initialized with a large negative number (Algorithm 2) to denote that at the beginning the last spike happened at negative infinity time. This is repeated for all layers until the last layer. For last layer the inputs are accumulated over all time steps and passed through a softmax layer to compute the multi-class probability. The cross-entropy loss function is defined on the output of the softmax and the weights are updated by performing the temporal and spatial credit assignment according to the STDB rule.

## 4.6 Experiments

We tested the proposed training mechanism on image classification tasks from CIFAR [76] and ImageNet [77] datasets. The results are summarized in Table 4.1. CIFAR10: The dataset consists of labeled $60,000$ images of 10 categories divided into training ($50,000$) and testing ($10,000$) set. The images are of size $32\times32$ with RGB channels.

CIFAR100: The dataset is similar to CIFAR10 except that it has 100 categories.

ImageNet: The dataset comprises of labeled high-resolution 1.2 million training images and $50,000$ validation images with 1000 categories.

## 4.7 Energy-Delay Product Analysis of SNNs

A single spike in an SNN consumes a constant amount of energy [64]. The first order analysis of energy-delay product of an SNN is dependent on the number of spikes and the

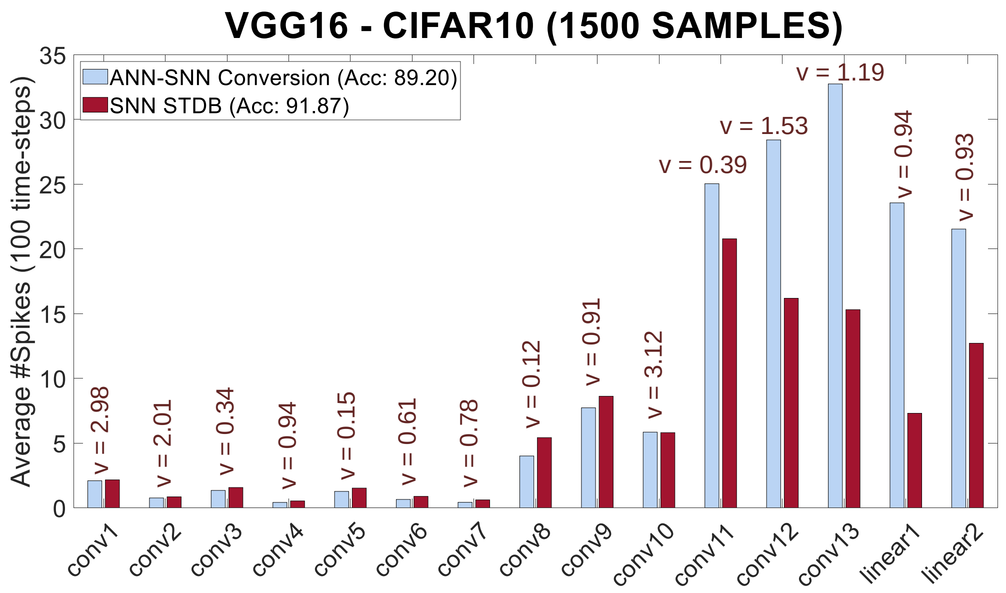


**Figure 4.3.** Average number of spikes for each layer in a VGG16 architecture for purely converted SNN and SNN trained with hybrid technique. The converted SNN and SNN trained with hybrid technique achieve an accuracy of 89.20% and 91.87%, respectively, for the randomly selected 1500 samples from the test set. Both the networks were inferred for 100 time steps and 'v' represents the threshold voltage for each layer obtained during the conversion process (Algorithm 1).

total number of time steps. Fig. 4.3 shows the average number of spikes in each layer when evaluated for 1500 samples from CIFAR10 testset for VGG16 architecture. The average is computed by summing all the spikes in a layer over 100 time steps and dividing by the number of neurons in that layer. For example, the average number of spikes in the $10^{th}$ layer is 5.8 for both the networks, which implies that over a 100 time step period each neuron in that layer spikes 5.8 times on average over all input samples. Higher spiking activity corresponds to lower energy-efficiency. The average number of spikes is compared for a converted SNN and SNN trained with conversion-and-STDB. The SNN trained with conversion-and-STDB has 1.5× less number of average spikes over all layers under iso conditions (time steps, threshold voltages, inputs, etc.) and achieves higher accuracy compared to the converted SNN. The converted SNNs when simulated for larger time steps further degrade the energy-delay product with minimal increase in accuracy [6].

Table 4.2. Comparion of our work with other SNN models on CIFAR10 and ImageNet datasets

| Model | Dataset | Training Method | Architecture | Accuracy | Time-steps |
|---|---|---|---|---|---|
| [63] | CIFAR10 | ANN-SNN Conversion | 2Conv, 2Linear | 82.95% | 6000 |
| [64] | CIFAR10 | ANN-SNN Conversion | 3Conv, 2Linear | 77.43% | 400 |
| [6] | CIFAR10 | ANN-SNN Conversion | VGG16 | 91.55% | 2500 |
| [70] | CIFAR10 | Spiking BP | VGG9 | 90.45% | 100 |
| [62] | CIFAR10 | Surrogate Gradient | 5Conv, 2Linear | 90.53% | 12 |
| **This work** | **CIFAR10** | **Hybrid Training** | **VGG16** | **91.13%** **92.02%** | **100** **200** |
| [6] | ImageNet | ANN-SNN Conversion | VGG16 | 69.96% | 2500 |
| **This work** | **ImageNet** | **Hybrid Training** | **VGG16** | **65.19%** | **250** |

## 4.8 Related Work

In [78], the authors proposed a method to directly train on SNN by keeping track of the membrane potential of spiking neurons only at spike times and backpropagating the error at spike times based on only the membrane potential. This method is not suitable for networks with sparse activity due to the 'dead neuron' problem: no learning happens when the neurons do not spike. In our work, we need one spike for the learning to start but gradient contribution continues in later time steps as shown in Fig. 4.1. In [79], the authors derived a surrogate gradient based method on the membrane potential of a spiking neuron at a single time step only. The error was backpropagated at only one time step and only the input at that time step contributed to the gradient. This method neglects the effect of earlier spike inputs. In our approach, the error is backpropagated for every time step and the weight update is performed on the gradients summed over all time steps. The authors in [80] proposed a gradient function similar to the one proposed in this work. They used the difference between the membrane potential and the threshold to compute the gradient compared to the difference in spike timing used in this work. The membrane potential is a continuous value whereas the spike time is an integer value bounded by the number of time steps. Therefore, gradients that depend on spike time can be pre-computed and stored in a look-up table for faster computation. They evaluated their approach on shallow architectures with two convolution layer for MNIST dataset. In this work, we trained deep SNNs with multiple stacked layers for complex calssification tasks. In [69], the authors performed backpropagation through time on SNN with a surrogate gradient defined on the membrane potential. The surrogate gradient was defined as piece-wise linear or exponential function of the membrane potential. The other surrogate gradients proposed in the literature are all computed on the membrane potential [12]. The authors in [70] approximated the neuron output as continuous low-pass filtered spike train. They used this approximated continuous value to perform backpropagation. Most of the works in the literature on direct training of SNN or conversion based methods have been evaluated on shallow architectures for simple classification problems. In Table 4.2 we compare our model with the models that reported accuracy on CIFAR10 and ImageNet dataset. In [62], the authors achieved convergence in 12 time steps by using a dedicated encoding layer to capture the input precision. It is beyond the scope of this work to compute the hardware and energy implications of such encoding layer. Our model performs better than all other models at far fewer number of time steps.

## 4.9 Conclusions

The direct training of SNN with backpropagation is computationally expensive and slow, whereas ANN-SNN conversion suffers from high latency. To address this issue we proposed a hybrid training technique for deep SNNs. We took an SNN converted from ANN and used its weights and thresholds as initialization for spike-based backpropagation of SNN. We then performed spike-based backpropagation on this initialized network to obtain an SNN that can perform with fewer number of time steps. The number of epochs required to train SNN was also reduced by having a good initial starting point. The resultant trained SNN had higher accuracy and lower number of spikes/inference compared to purely converted SNNs at reduced number of time steps. The backpropagation through time was performed with surrogate gradient defined using neuron's spike time that captured the temporal information and helped in reducing the number of time steps. We tested our algorithm on CIFAR and ImageNet datasets and achieved state-of-the-art performance with fewer number of time steps.

# 5. DIET-SNN: A LOW-LATENCY SPIKING NEURAL NETWORK WITH DIRECT INPUT ENCODING AND LEAKAGE AND THRESHOLD OPTIMIZATION

## 5.1 Introduction

In recent years, a class of neural networks inspired by the event-driven form of computations in the brain has gained popularity for their promise of low-power computing [81], [82]. Spiking neural networks (SNNs) first emerged in computational neuroscience as an attempt to model the behavior of biological neurons [83]. They were pursued for low-complexity tasks implemented on bio-plausible neuromorphic platforms. At the same time in standard deep learning, the analog-valued artificial neural networks (ANNs) became the de-facto model for training various computer vision and natural language processing tasks [84], [85]. The skyrocketing performance and success of multi-layer ANNs came at a significant power and energy cost [2]. Recently, major chip maker Nvidia estimated that $80 - 90\%$ of the energy cost of neural networks at data centers lies in inference processing [86]. The tremendous energy costs and the demand for edge intelligence on battery-powered devices have shifted the focus on exploring lightweight energy-efficient inference models for machine intelligence. To that effect, various techniques such as weight pruning [87], model compression [88], and quantization methods [89] are proposed to reduce the size and computations in ANNs. Nonetheless, the inherent one-shot analog computation in ANNs requires the expensive operation of multiplying two real numbers (except when both weights and activations are 1-bit [90]). In contrast, SNNs inherently compute and transmit information with binary signals distributed over time, providing a promising alternative for power-efficient machine intelligence.

For a long time, SNNs' success was delayed due to the unavailability of good learning algorithms. In recent years, the advent of supervised learning algorithms for SNNs has overcome many of the roadblocks surrounding the discontinuous derivative of the spike activation function [12]. Since SNNs receive and transmit information through spikes, analog values need to be encoded into spikes. There are a plethora of input encoding methods like rate coding [6], [44], temporal coding [91], rank-order coding [92], and other special coding schemes [93]. Among these, rate-coding has shown competitive performance on complex tasks [6], [44], [70] while others are limited to simple tasks like learning the XOR function and classifying digits from the MNIST dataset. Also, dynamic vision sensors (DVS) record the change in image pixel intensities and directly convert it to spikes that can estimate optical flow [94] and classify hand gestures [80]. In rate coding, the analog value is represented by the rate of firing of the neuron. In each timestep, the neuron either fires (output '1') or stays inactive (output '0'). Therefore, an analog value of 0.5 is represented by a neuron that fires during 50% of the total number of timesteps. The number of timesteps[1] determines the discretization error in the representation of the analog value by spike-train. This leads to adopting a large number of timesteps for high accuracy at the expense of high inference latency [6]. The two other parameters that are crucial for SNNs are firing *threshold* of the neuron and membrane potential *leak*. The neuron fires when the membrane potential exceeds the firing threshold and the potential is reset after each firing. Such neurons are usually referred to as integrate-and-fire (IF) neurons. The threshold value is very significant for the correct operation of SNNs because a high threshold will prevent the neuron from firing ('dead-neuron' problem), and a lower threshold will lead to excessive firing, affecting the ability of the neuron to differentiate between two input patterns. Another neuron model, leaky-integrate-and-fire (LIF), introduces a leak factor that allows the membrane potential to keep shrinking over time [96]. Most of the recent work on supervised learning in SNNs has either employed the IF or the LIF neuron model [6], [44], [70], [97], [98]. Some proposals adopt kernel-based spike response models [68], [78], [99], but for the most part, these approaches show limited performance on simple datasets and do not scale for deep networks. The leak provides an additional knob that can potentially be used to tune SNNs for better energy-efficiency. However, there has not been any exploration of the full design space of optimizing the leak and the threshold to achieve better latency (or energy) and accuracy tradeoff. Research, so far, has been mainly focused on using fixed leak for the entire network that can limit the capabilities of SNNs [70], [97]. The firing thresholds are also fixed [70]

[1]↑Wall-clock time for 1 'timestep' is dependent on the number of computations performed and the underlying hardware [95]. In the simulation, 1 timestep is the time taken to perform 1 forward pass

or selected based on some heuristics [5], [6]. In [6], the threshold was selected as the maximum pre-activation of each layer, whereas in [5] the authors selected a certain percentile of the pre-activation distribution as the threshold. Some recent works employ leak/threshold optimization, but their application is limited to simple datasets [100], [101]. The current challenges in SNN models are high inference latency and energy, long training time, and substantial training costs in terms of memory and computation. Most of these challenges arise due to in-efficient input encoding, and improper methods of selecting the membrane leak and the threshold.

To address these challenges, this paper makes the following contributions:

- We propose a gradient descent based training method that learns the correct membrane leak and firing threshold for each layer of a deep spiking network via error-backpropagation. The goal is to jointly optimize the neuron parameters (membrane leak and threshold) and the network parameters (weights) to achieve high accuracy at low inference latency. The tailored membrane leak and threshold for each layer leads to large improvements in activation sparsity and energy-efficiency.

- We train the first convolutional layer to act as the spike-generator, whose spike-rate is a function of the weights, membrane leak, and threshold. This also eliminates the need for a generator function (and associated overheads) used in other coding schemes[2].

- To evaluate the effectiveness of the proposed algorithm, we train SNNs on both VGG [66] and ResNet [67] architectures for CIFAR [76] and ImageNet [102] datasets. DIET-SNN achieves similar accuracy as ANN with $6-18\times$ less compute energy. The performance is achieved at inference latency of 5 timesteps compared to $100-2000$ timesteps for state-of-the-art SNN models.

## 5.2 Background and Related Work

The development of efficient learning algorithms for deep SNNs is an on-going research challenge. There has been a significant amount of success with recent supervised learning

[2]↑For rate-coding, a Poisson generator is used to convert the analog values to spike-train [44]. The encoder generates random numbers every timestep and compares it with the analog values to produce the spikes.

algorithms [6], [11], [62], [69], [98] that can be broadly classified as conversion algorithms [5], [6], [64] and spike-based backpropagation algorithms [11], [70]. Additionally, there are bio-plausible algorithms that employ spike timing dependent plasticity learning rule [103], or feedback alignment to update the weights. Some of these algorithms apply random weights [104] or fixed weights [105] as the feedback weight during backpropagation[3]. The success of these algorithms is limited to simple tasks. Therefore, we focus our discussion on ANN-to-SNN conversion and backpropagation algorithms that are more suitable for complex tasks and are scalable to deep networks.

### 5.2.1 ANN-to-SNN Conversion

ANN-to-SNN conversion is the most successful method of training rate-coded deep SNNs [5], [6], [44], [64], [98]. An ANN with ReLU neurons is trained with standard backpropagation with some restrictions (no bias, average pooling, no batch normalization). Although some works show that some of the restrictions can be relaxed [5]. Next, SNN (iso-architecture as ANN) with IF neurons is initialized with the weights of the trained ANN. The underlying principle is that the ReLU neuron can be mapped to the IF neuron with minimum loss. The mapping is possible for SNNs that operate on rate-coded inputs [6], [44] or on direct input encoding [5]. The major bottleneck of this method is to determine the firing threshold of the IF neurons that can balance the accuracy-latency tradeoff. Generally, the threshold is computed as the maximum pre-activation of the IF neuron resulting in high inference accuracy at the cost of high inference latency ($2000 - 2500$ timesteps) [6]. In recent work, the authors showed that instead of using the maximum pre-activation, a certain percentile of the pre-activation distribution reduces the inference latency ($100 - 200$ timesteps) with minimal accuracy drop [106]. These heuristic techniques of determining the firing threshold lead to a sub-optimal accuracy-latency tradeoff. Additionally, ANN-to-SNN conversion has a major drawback: the absence of the timing information. The quintessential parameter 'time' is not utilized in the conversion process which leads to higher inference latency. The backprop-

[3]↑In standard backpropagation, the feedback weight is $W^T$, where $W$ is the weight used in the forward pass

agation based algorithms, described next, employ the timing information to calculate the gradients and have lower inference latency compared to conversion algorithms.

### 5.2.2 Error Backpropagation in SNN

ANNs have achieved success with gradient-based training that backpropagates the error signal from the output layer to the input layer. It requires computing a gradient of each operation performed in the forward pass. Unfortunately, the IF and the LIF neuron does not have a continuous derivative. The derivative of the spike function (Dirac delta) is undefined at the time of spike and '0' otherwise. This has hindered the application of standard backpropagation in SNN. There have been many proposals to perform gradient-based training in SNNs [68], [70] – among them, the most successful is surrogate-gradient based optimization [12]. The discontinuous derivative of the IF neuron is approximated by a continuous function that serves as the surrogate for the real gradient. SNNs trained with surrogate-gradient perform backpropagation through time (BPTT) to achieve high accuracy and low latency (100 timesteps), but the training is very compute and memory intensive in terms of total training iterations compared to conversion techniques. The multiple-iteration training effort with exploding memory requirement for backpropagation has limited the application of this method to simpler tasks on shallow architectures [70].

### 5.2.3 Hybrid SNN Training

In recent work, the authors proposed a hybrid mechanism to circumvent the high training costs of backpropagation as well as maintain low inference latency ($100-250$ timesteps) [97]. The method involves both ANN-to-SNN conversion and error-backpropagation. A trained ANN is converted to an SNN as described earlier and the weights of the converted SNN are further fine-tuned with surrogate gradient and BPTT. The authors showed a faster convergence ($< 20$ epochs) in SNN training due to the precursory initialization from the ANN-to-SNN conversion. This presents a practically feasible method to train deep SNNs with limited resources, which is otherwise challenging with only backpropagation from random initialization [70]. Hybrid training tries to achieve the best of both worlds: high accuracy

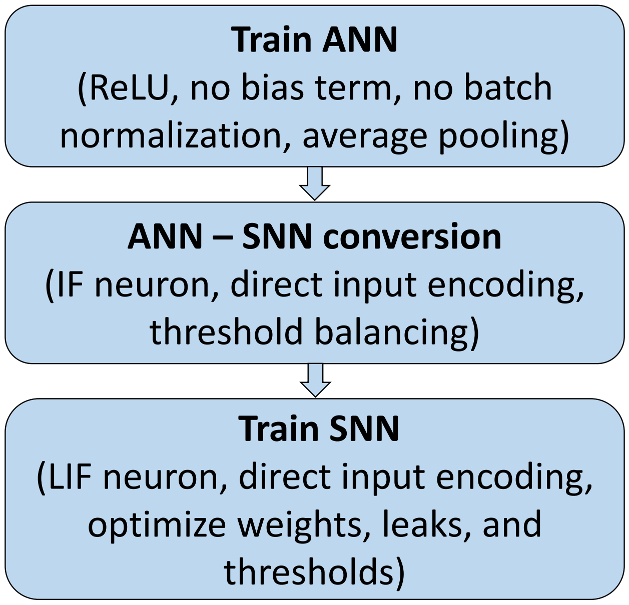

Train ANN
(ReLU, no bias term, no batch normalization, average pooling)
ANN – SNN conversion
(IF neuron, direct input encoding, threshold balancing)
Train SNN
(LIF neuron, direct input encoding, optimize weights, leaks, and thresholds)

**Figure 5.1.** Training pipeline

and low latency. But it still employs rate coding, fixed membrane leak, and fixed threshold, and therefore, the latency-accuracy tradeoff can be improved further.

In this work, we adopt the hybrid training method to train the SNNs. We start with ANN-to-SNN conversion and select the threshold for each layer as $95^{th}$ percentile of the pre-activation distribution. The pixel intensities are directly applied in the input layer during the threshold computation. This serves as the initial model that is further trained to optimize the membrane leak and threshold.

## 5.3 Algorithm for training DIET-SNN

In this section, we describe the input encoding, leaky-integrate-and-fire (LIF) neuron model, and derive the backpropagation equations to update the weights, the thresholds, and the leak factors.

### 5.3.1 Direct Input Encoding

The pixel intensities of an image are applied directly to the input layer of the SNN at each timestep [5], [106]. The first convolutional layer composed of LIF neurons acts as both

**Table 5.1.** Top-1 classification accuracy

| Architecture | ANN | ANN-to-SNN | Training only weights in SNN | **DIET-SNN** | Timesteps ($T$) |
|---|---|---|---|---|---|
| | | | CIFAR10 | | |
| VGG6 | 90.80% | 86.19% | 89.24% | **89.42%** | 5 |
| | | | | **90.05%** | 10 |
| VGG16 | 93.72% | 73.52% | 91.68% | **92.70%** | 5 |
| | | | | **93.44%** | 10 |
| ResNet20 | 92.79% | 47.26% | 90.29% | **91.78%** | 5 |
| | | | | **92.54%** | 10 |
| | | | CIFAR100 | | |
| VGG16 | 71.82% | 46.54% | 65.83% | **69.67%** | 5 |
| ResNet20 | 64.64% | 31.40% | 62.95% | **64.07%** | 5 |
| | | | ImageNet | | |
| VGG16 | 70.08% | 24.58% | 64.32% | **69.00%** | 5 |

the feature extractor and the spike-generator, which accumulates the weighted pixel values and generates output spikes. This is similar to rate-coding, but the spike-rate is a function of the weights, membrane leak, and threshold that are all learned by gradient-descent.

### 5.3.2 Neuron Model

We employ the LIF neuron model described by

$$u_{\mathrm{i}}^{t} = \lambda_{\mathrm{i}} u_{\mathrm{i}}^{t-1} + \sum_{\mathrm{j}} w_{\mathrm{ij}} o_{\mathrm{j}}^{t} - v_{\mathrm{i}} o_{\mathrm{i}}^{t-1} \tag{5.1}$$

$$z_{\mathrm{i}}^{t-1} = \frac{u_{\mathrm{i}}^{t-1}}{v_{\mathrm{i}}} - 1 \quad \text{and} \quad o_{\mathrm{i}}^{t-1} = \begin{cases} 1, & \text{if } z_{\mathrm{i}}^{t-1} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

where $u$ is the membrane potential, $\lambda$ is the leak factor with a value in $[0-1]$, $w$ is the weight connecting pre-neuron j and post-neuron i, $o$ is the binary spike output, $v$ is the firing threshold, and $t$ represents the timestep. The first term in Equation 5.1 denotes the leakage in the membrane potential, the second term integrates the weighted input received from pre-neuron, and the third term accounts for the reduction in potential when the neuron generates an output spike. After the spike, a soft reset is performed where the potential is reduced by threshold instead of resetting to zero [98]. The threshold governs the average integration time of input, and the leak regulates how much of the potential is retained from the previous timestep. Now, we derive the expressions to compute the gradients of the parameters at all layers. The spatial and temporal credit assignment is performed by unrolling the network in time and employing BPTT.

### 5.3.3 Output layer

The neuron model in the output layer only accumulates the incoming inputs without any leakage and does not generate an output spike and is described by

$$u_l^t = u_l^{t-1} + W_l o_{l-1}^t \tag{5.3}$$

where $u_l$ is a vector containing the membrane potential of $N$ output neurons, $N$ is the number of classes in the task, $W_l$ is the weight matrix connecting the output layer and the previous layer, and $o_{l-1}$ is a vector containing the spike signals from layer $(l-1)$. The loss function is defined on $u_l$ at the last timestep $T$. We employ the cross-entropy loss and the softmax is computed on $u_l^T$. The symbol $T$ is used for timestep and not to denote the transpose of a matrix.

$$s(u_l^T) : \begin{bmatrix} u_1^T \\ \dots \\ u_N^T \end{bmatrix} \rightarrow \begin{bmatrix} s_1 \\ \dots \\ s_N \end{bmatrix} s_{\mathrm{i}} = \frac{\mathrm{e}^{u_{\mathrm{i}}^T}}{\sum_{k=1}^{N} \mathrm{e}^{u_k^T}} \tag{5.4}$$

$$L = -\sum_{\mathrm{i}} y_{\mathrm{i}} log(s_{\mathrm{i}}), \quad \frac{\partial L}{\partial u_l^T} = s - y \tag{5.5}$$

where $s$ is the vector containing the softmax values, $L$ is the loss function, and $y$ is the one-hot encoded vector of the true label or target. The weight update is computed as

$$W_l = W_l - \eta \Delta W_l \tag{5.6}$$

**Table 5.2.** DIET-SNN compared with other SNN models

| Model | Method | Architecture | SNN Accuracy | Timesteps |
|---|---|---|---|---|
| | | CIFAR10 | | |
| [6] | ANN-to-SNN | VGG16 | 91.55% | 2500 |
| [5] | ANN-to-SNN | 4 Conv, 2 FC | 90.85% | 400 |
| [97] | Hybrid | VGG16 | 92.02% | 200 |
| [70] | Backprop | VGG9 | 90.45% | 100 |
| [62] | Backprop | CIFARNet | 90.53% | 12 |
| [107] | Backprop | CIFARNet | 90.98% | 8 |
| [108] | Backprop | CIFARNet | 91.41% | 5 |
| **This work** | **DIET-SNN** | **CIFARNet** | **91.59%** | **5** |
| **This work** | **DIET-SNN** | **VGG16** | **92.70%** | **5** |
| | | CIFAR100 | | |
| [98] | ANN-to-SNN | VGG16 | 70.09% | 768 |
| [97] | Hybrid | VGG11 | 67.87% | 125 |
| [106] | ANN-to-SNN | VGG15 | 63.20% | 62 |
| **This work** | **DIET-SNN** | **VGG16** | **69.67%** | **5** |
| | | ImageNet | | |
| [6] | ANN-to-SNN | VGG16 | 69.96% | 2500 |
| [98] | ANN-to-SNN | VGG16 | 71.34% | 768 |
| [5] | ANN-to-SNN | VGG16 | 49.61% | 400 |
| [97] | Hybrid | VGG16 | 65.19% | 250 |
| [106] | ANN-to-SNN | VGG15 | 66.56% | 64 |
| [107] | Backprop | AlexNet | 50.22% | 10 |
| **This work** | **DIET-SNN** | **VGG16** | **69.00%** | **5** |

$$\begin{aligned}\Delta W_l = \sum_t \frac{\partial L}{\partial W_l} = \sum_t \frac{\partial L}{\partial u_l^t}\frac{\partial u_l^t}{\partial W_l} = \frac{\partial L}{\partial u_l^T}\sum_t \frac{\partial u_l^t}{\partial W_l} \\ = (s-y)\sum_t o_{l-1}^t\end{aligned} \tag{5.7}$$

$$\frac{\partial L}{\partial o_{l-1}^t} = \frac{\partial L}{\partial u_l^T}\frac{\partial u_l^t}{\partial o_{l-1}^t} = (s-y)W_l \tag{5.8}$$

where $\eta$ is the learning rate.

### 5.3.4 Hidden layers

The neurons in the convolutional and fully-connected layers are defined by the LIF model as

$$u_l^t = \lambda_l u_l^{t-1} + W_l o_{l-1}^t - v_l o_l^{t-1} \tag{5.9}$$

$$z_l^t = \frac{u_l^t}{v_l} - 1 \quad \text{and} \quad o_l^t = \begin{cases} 1, & \text{if } z_l^t > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.10}$$

where $\lambda_l$ ($v_l$) is a real value representing leak (threshold) for all neurons in layer $l$. All neurons in a layer share the same leak and threshold value. This reduces the number of trainable parameters and we did not observe any significant improvement by assigning individual threshold/leak to each neuron. The weight update is calculated as

$$\begin{aligned} \Delta W_l = \sum_t \frac{\partial L}{\partial W_l} &= \sum_t \frac{\partial L}{\partial o_l^t} \frac{\partial o_l^t}{\partial z_l^t} \frac{\partial z_l^t}{\partial u_l^t} \frac{\partial u_l^t}{\partial W_l} \\ &= \sum_t \frac{\partial L}{\partial o_l^t} \frac{\partial o_l^t}{\partial z_l^t} \frac{1}{v_l} o_{l-1}^t \end{aligned} \tag{5.11}$$

$^{\partial o_l^t}/_{\partial z_l^t}$ is the discontinuous gradient and we approximate it with the surrogate gradient [11]

$$\frac{\partial o_l^t}{\partial z_l^t} = \gamma\ max\{0, 1 - |z_l^t|\} \tag{5.12}$$

$$\frac{\partial o_l^t}{\partial u_l^t} = \frac{\partial o_l^t}{\partial z_l^t} \frac{\partial z_l^t}{\partial u_l^t} = \frac{\partial o_l^t}{\partial z_l^t} \frac{1}{v_l} \tag{5.13}$$

where $\gamma$ is a constant denoting the maximum value of the gradient. The threshold update is then computed as

$$v_l = v_l - \eta \Delta v_l \tag{5.14}$$

**Table 5.3.** ANN vs DIET-SNN compute energy. Each operation in ANN (SNN) consumes $4.6pJ$ ($0.9pJ$). The input layer in DIET-SNN is non-spiking, so it's energy is same as ANN. Column-5 shows the ratio of #operations in input layer to the total #operations in the network.

| Architecture (timesteps) | Dataset | Normalized $\#OP_{ANN}(a)$ | Normalized $\#OP_{SNN}(b)$ | $\frac{\#OP \text{ layer 1}}{\text{Total } \#OP}(c)$ | ANN / DIET-SNN Energy $(\frac{a*4.6}{c*4.6+(1-c)*b*0.9})$ |
|---|---|---|---|---|---|
| VGG6 (T=5) | CIFAR10 | 1.0 | 0.14 | 0.029 | 18 |
| VGG16 (T=5) | CIFAR10 | 1.0 | 0.39 | 0.005 | 12.4 |
| VGG16 (T=5) | CIFAR100 | 1.0 | 0.40 | 0.005 | 12.1 |
| VGG16 (T=5) | ImageNet | 1.0 | 0.41 | 0.006 | 11.7 |
| ResNet20 (T=5) | CIFAR10 | 1.0 | 0.76 | 0.013 | 6.3 |
| ResNet20 (T=5) | CIFAR100 | 1.0 | 0.72 | 0.013 | 6.6 |

$$\begin{aligned}\Delta v_l = \sum_t \frac{\partial L}{\partial v_l} &= \sum_t \frac{\partial L}{\partial o_l^t}\frac{\partial o_l^t}{\partial z_l^t}\frac{\partial z_l^t}{\partial v_l} \\ &= \sum_t \frac{\partial L}{\partial o_l^t}\frac{\partial o_l^t}{\partial z_l^t}\left(\frac{-v_l o_l^{t-1} - u_l^t}{(v_l)^2}\right)\end{aligned} \tag{5.15}$$

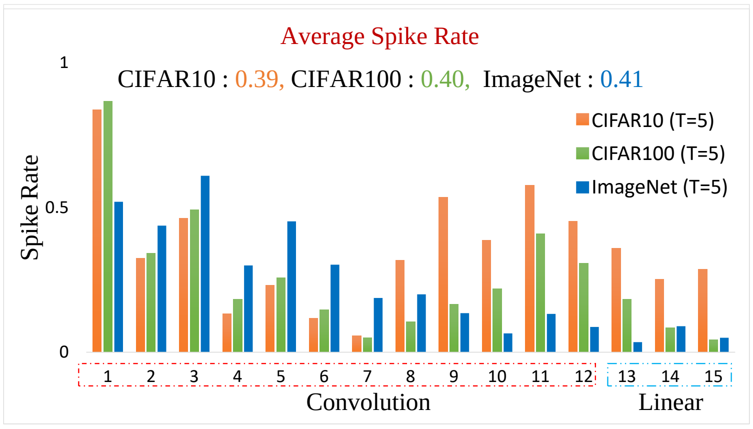
And finally the leak update is computed as

$$\lambda_l = \lambda_l - \eta\Delta\lambda_l \tag{5.16}$$

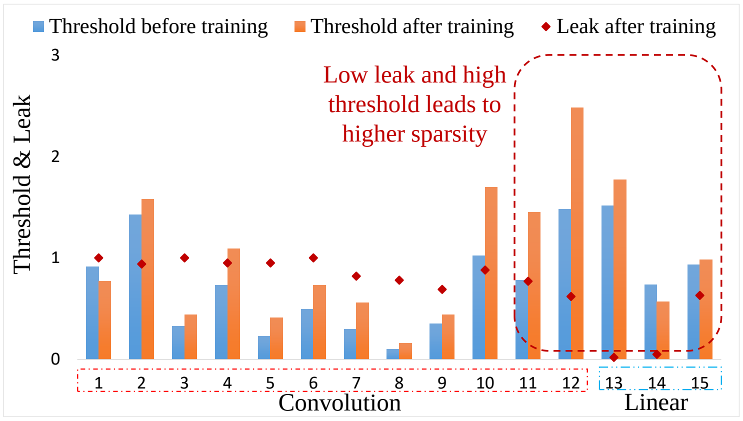
$$\begin{aligned}\Delta\lambda_l = \sum_t \frac{\partial L}{\partial \lambda_l} &= \sum_t \frac{\partial L}{\partial o_l^t}\frac{\partial o_l^t}{\partial u_l^t}\frac{\partial u_l^t}{\partial \lambda_l} \\ &= \sum_t \frac{\partial L}{\partial o_l^t}\frac{\partial o_l^t}{\partial u_l^t}u_l^{t-1}\end{aligned} \tag{5.17}$$

## 5.4 Experiments

The three-step DIET-SNN training pipeline (Fig. 5.1) begins with training an ANN without the bias term and batch-normalization to achieve minimal loss during ANN-to-SNN conversion [6], [44], [97]. Dropout [74] is used as the regularizer and the dropout mask is unchanged during all timesteps of an input sample [97]. Average-pooling is used to reduce the feature map size in VGG architectures, whereas for ResNets, a stride of 2 is employed to reduce the feature size. Next, the trained ANN is converted to SNN with IF neurons. The threshold is computed sequentially as $95^{th}$ percentile of the pre-activation distribution at each layer. The pre-activation for each neuron is the weighted sum of inputs $\sum_{\mathrm{j}} o_{\mathrm{j}} w_{\mathrm{ij}}$ received by the neuron. During threshold computation, the leak in the hidden layers is set to unity, and the input layer employs direct input encoding. Finally, the converted SNN is trained with error-backpropagation to optimize the weights, the membrane leak, and the firing thresholds of each layer as described by the equations in Section 5.3. Algorithm 4 describes the three processes employed in the training pipeline. We evaluate the performance of DIET-SNN on VGG and ResNet architectures for CIFAR and ImageNet datasets (Table 5.1). Column-2 in Table 5.1 shows the ANN accuracy; column-3 shows the accuracy after ANN-to-SNN conversion with 5 timesteps; column-4 shows the accuracy when only the weights in SNN are trained with spike-based backpropagation; column-5 shows the accuracy when the weights, threshold, and leak are jointly optimized (DIET-SNN). The performance of DIET-SNN compared to current state-of-the-art SNNs is shown in Table 5.2. DIET-SNN shows $5-100\times$ improvement in inference latency compared to other spiking networks. Although the authors in [62], [107], [108] achieved competitive accuracy on the CIFAR10 dataset with low inference latency, the accuracy degraded on more challenging tasks or the energy-efficiency of SNNs was compromised. The authors in [107] propose a method to train two networks (ANN and SNN) simultaneously and share the weights between them. The weights are trained in ANN and continuously copied to SNN; the activations of ANN are computed as the sum of spikes in SNN for that layer. As the training is not performed in the spiking domain, the temporal information is not utilized and the method fails to achieve competitive accuracy on the ImageNet dataset (Table 5.2). The normalization method, NeuNorm, computes a weighted summation of spike count and uses that quantity as the input to the convolutional layer instead of the raw spike signals [62]. Therefore, the convolution requires the multiply-and-accumulate (MAC) operation as both the input and the weight are real-valued quantities. In SNN, one of the major advantages is that the expensive MAC operation (needed in ANN) is reduced to simple additions due to binary inputs (more discussion in Section 5.5). Although the authors achieved competitive accuracy in a lower number of time-steps, the proposed



(a) Spike Rate



(b) Leak and Threshold after training

**Figure 5.2.** (a) Layerwise spike rate for VGG16 during inference over entire test-set. Average spike rate is calculated as total $^{\#spikes}/_{\#neurons}$. An average spike rate of 0.41 indicates that every neuron fired on average 0.41 times for each image over all timesteps. (b) Layerwise leak and threshold for VGG16 on CIFAR100 dataset. The threshold before training represents the values obtained from ANN-to-SNN conversion process. The leak before training is unity for all layers.

normalization method loses the energy benefits of SNNs and is similar to ANNs in terms of the type of computation. In contrast, DIET-SNN achieves state-of-the-art accuracy on CIFAR and ImageNet datasets with spike-based communication between layers (except for the first layer) that leads to better energy-efficiency.

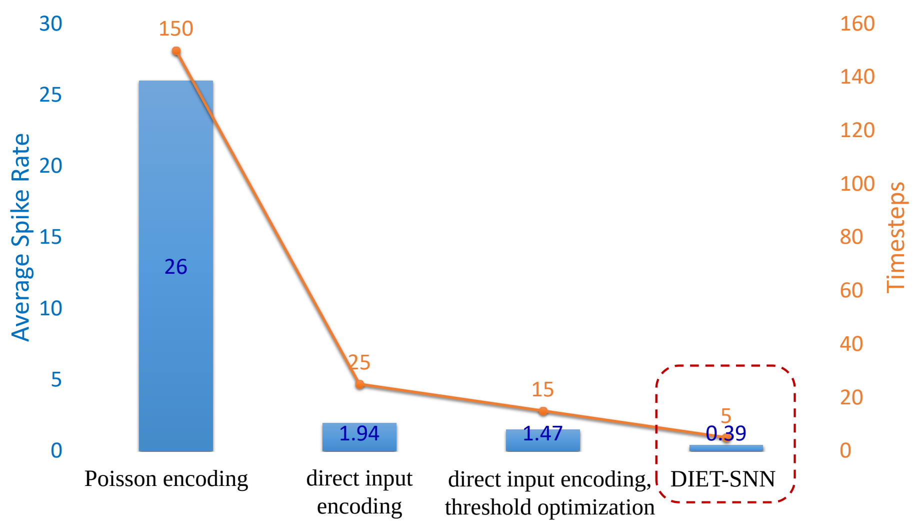


**Figure 5.3.** Effect of employing direct input encoding, threshold and leak optimization

## 5.5 Energy Efficiency

In this section, we delve in the compute energy comparison between ANN and SNN. In ANN, each operation computes a dot-product involving one floating-point (FP) multiplication and one FP addition (MAC), whereas, in SNN, each operation is only one FP addition due to binary spikes. The computations in SNN implemented on neuromorphic hardware are event-driven [82], [95]. Therefore, in the absence of spikes, there are no computations and no active energy is consumed. We computed the energy cost/operation for ANNs and SNNs in 45nm CMOS technology (Table 5.4). The energy cost for 32-bit ANN MAC operation ($4.6pJ$) is 5.1× more than SNN addition operation ($0.9pJ$) [109]. These numbers may vary for different technologies, but generally, in most technologies, the addition operation is much

**Table 5.4.** Energy costs of addition and multiplication in 45nm CMOS [109]

| | |
|---|---|
| FP ADD (32 bit) | $0.9pJ$ |
| FP MULT (32 bit) | $3.7pJ$ |
| FP MAC (32 bit) | $(0.9 + 3.7)$ |
| | $= 4.6pJ$ |

cheaper than the multiplication operation. In ANN, the number of operations in convolution layer is

$$\#OP_{ANN} = k_w \times k_h \times c_{\text{in}} \times h_{out} \times w_{out} \times c_{out}, \tag{5.18}$$

and for fully-connected layer is

$$\#OP_{ANN} = f_{\text{in}} \times f_{out}, \tag{5.19}$$

where $k_w(k_h)$ is kernel width (height), $c_{\text{in}}(c_{out})$ is the number of input (output) channels, $h_{out}(w_{out})$ is the height (width) of the output feature map, and $f_{\text{in}}(f_{out})$ is the number of input (output) features. The number of operations in iso-architecture SNN is specified by

$$\#OP_{SNN} = Spi\text{ke}Rate_l \times \#OP_{ANN} \tag{5.20}$$

$$Spi\text{ke}Rate_l = \frac{\#TotalSpikes_l \text{ over all inference timesteps}}{\#Neurons_l} \tag{5.21}$$

where $Spi\text{ke}Rate_l$ is the total spikes in layer $l$ over all timesteps divided by the number of neurons in layer $l$. A spike rate of 1 (every neuron fired once) implies that the number of operations for ANN and SNN are the same (though operations are MAC in ANN while addition in SNNs). Lower spike rates denote more sparsity in spike events and higher energy-efficiency. Table 5.3 shows the compute energy comparison of ANN and SNN. As the first layer in our proposed network is non-spiking, we compute its energy based on MAC operations (column-5 in Table 5.3). Overall SNNs achieve better energy-efficiency due to two reasons: low spike rate leading to less number of operations, and lower compute energy/operation (MAC vs ADD). The average spike rate for VGG16 during inference is 0.4 (Fig. 5.2(a)); therefore, the effective number of operations in SNN is lower than that in ANN. The low spike rate is facilitated by the low (high) leak (threshold) values in the deeper layers obtained from the gradient descent training (Fig. 5.2(b)). The energy for ResNet is more than VGG because more than 50% of the total operations in ResNet occurs in the first 3 layers where the spike rate is high. The standard ResNet architecture was modified with initial 3 plain convolutional layers to minimize the accuracy loss during ANN-to-SNN conversion [6]. The authors in [106] reported an average spike rate of 2.35 for VGG16 on CIFAR100 with 62% accuracy. The maximum spike rate of 20 was reported for VGG16 architecture on CIFAR10 dataset [97]. DIET-SNN performs considerably better in all metrics compared to these models and achieves better compute energy than ANN on complex tasks like CIFAR and ImageNet with similar accuracy. We did not consider the data movement cost in our evaluation as it is dependent on the system architecture and the underlying hardware implementation. Although we would like to mention that in SNN the membrane potentials have to be fetched at every timestep, in addition to the weights and activations. Many proposals reduce the memory cost by data buffering [110], trading computations for memory [111], and data reuse through efficient dataflows [112]. All such techniques can be extended to SNNs to address the memory cost. The training of SNNs is still a cause of concern for energy-efficiency because it requires several days, even on high-performance GPUs. The hybrid approach and DIET-SNN alleviate the issue to some extent by reducing the number of training epochs and the number of timesteps, but further innovations in both algorithms and accelerators for SNNs are required to reduce the training cost.

## 5.6 Effect of direct input encoding and threshold/leak optimization

Table 5.1 shows the effect on accuracy (under iso-timesteps) by training threshold/leak along with direct input encoding. In this section, we analyze the impact of input encoding, threshold and leak on the average spike rate and latency (under iso-accuracy). We train four different spiking networks: (a) SNN with IF neuron and Poisson rate encoding; (b) SNN with IF neuron and direct input encoding; (c) threshold optimization added to (b); (d) SNN with LIF neuron, analog encoding, and threshold/leak optimization (DIET-SNN). The SNNs are trained to achieve similar accuracy for VGG16 on CIFAR10. The networks (a)-(d) achieved an accuracy of 92.10%, 92.41%, 92.37%, and 92.70%, respectively. The network with Poisson input encoding required 150 timesteps with average spike rate of 26 (Fig. 5.3). By replacing Poisson encoding with direct input encoding (proposed work), the latency (spike-rate) improved to 25 timesteps (1.94). As mentioned in Section 5.1, the information in rate-coded SNNs is encoded in the firing rate of the neuron. In Poisson encoding, the firing rate is proportional to the pixel value, whereas in direct input encoding, the SNN learns the optimal firing rate by training the parameters of the first convolutional layer. . This reduces the number of timesteps required to encode the input. Next, the addition of threshold optimization reduces the latency (spike-rate) to 15 timesteps (1.47). In SNNs with IF neurons, a neuron's activity is dependent on the ratio of the weights and the neuron's threshold. Therefore, training only weights should be sufficient, however, training threshold along with weights provides additional parameter for optimization and leads to lower latency as shown in network (c) compared to (b). . Finally, the addition of the leak parameter reduces the latency (spike-rate) to 5 timesteps (0.39). The leak and threshold together eliminate the excess membrane potential and suppress unnecessary firing activities that improve the latency and the spike-rate. The compute energy compared to ANN (Table 5.3) for the networks (a)-(d) are 0.2, 2.6, 3.4, and 12.4, respectively. Therefore, the co-optimization of weights, leak, and threshold along with direct input encoding leads to improved latency and low energy-consumption.

## 5.7 Conclusions

SNNs that operate with asynchronous discrete events can potentially solve the energy issue in deep learning. To that effect, we presented DIET-SNN, an energy-efficient spiking network that is trained to operate with low inference latency and high activation sparsity. The membrane leak and the firing threshold of the LIF neurons are trained with error-backpropagation along with the weights of the network to optimize both accuracy and latency. We initialize the parameters of DIET-SNN, taken from a trained ANN, to speed-up the training with spike-based backpropagation. The image pixels are applied directly as input to the network, and the first convolutional layer is trained to perform the spike-generation operation. This leads to high activation sparsity in the convolutional and dense layers of the network. The high sparsity combined with low inference latency reduces the compute energy by $6-18\times$ compared to an equivalent ANN with similar accuracy. DIET-SNN achieves similar accuracy as other state-of-the-art SNN models with $20-500\times$ less number of timesteps.

**Algorithm 4** DIET-SNN training algorithm with ANN-to-SNN conversion followed by spike-based backpropagation to jointly optimize weights, thresholds, and membrane leaks.

```
*ANN training*
**Input:** Dataset ($D$), ANN model ($N_a$), initial weights ($W_a$)
**while** *stopping criterion not met* **do**
    sample mini-batch of input ($X$) - target ($Y$) pairs from $D$
    $\hat{Y} = N_a(X)$    //Forward propagation
    $Loss = CrossEntropy(Y, \hat{Y})$
    $W_a \leftarrow W_a - \epsilon \frac{dLoss}{dW_a}$ // Weight update

*ANN-to-SNN conversion*
**Input:** Trained ANN weights ($W_a$), mini-batch of input ($X$) - target ($Y$) pairs from $D$, SNN model ($N_s$), Timesteps ($T$)
// Initialize SNN weights with trained ANN weights
$W_s \leftarrow W_a$
$V$: threshold voltage
// compute the threshold for all layers sequentially
**for** $l$ ***in*** $N_s$ **do**
    **for** $t$=1 **to** $T$ **do**
        $A_l = N_l(X)$ // pre-nonlinearity activation of layer $l$
        **if** $95^{th}$ *percentile of* $A_l$ ¿ $V_l$ **then**
            $V_l = 95^{th}$ percentile of $A_l$

*Spike-based backpropagation on converted SNN*
**Input:** Dataset ($D$), Converted SNN ($N_s$), SNN weights ($W_s$)
**while** *stopping criterion not met* **do**
    sample mini-batch of input ($X$) - target ($Y$) pairs from $D$, $U$: membrane potential, $O$: spike output, $\lambda$: membrane leak
    **for** $t$=1 **to** $T$ **do**
        $O_0 = X$ // direct input encoding
        **for** $l$=1 **to** $L$-1 **do**
            // accumulate the output of previous layer in $U$, soft reset when spike occurs
            $U_l^t = \lambda_l U_l^{t-1} + W_{s_l} O_{l-1}^t - V_l O_l^{t-1}$
            // generate spike if $U$ exceeds $V$
            **if** $U_l > V_l$ **then**
                $O_l = 1$
        // only accumulation in the final layer
        $U_L^t = U_L^{t-1} + W_{s_L} O_{L-1}^t$
    $Loss = CrossEntropy(Y, U_L^T)$
    $W_a \leftarrow W_a - \epsilon \frac{dLoss}{dW_a}$ // Weight update
    $V \leftarrow V - \epsilon \frac{dLoss}{dV}$ // Threshold update
    $\lambda \leftarrow \lambda - \epsilon \frac{dLoss}{d\lambda}$ // Leak update
```

# 6. LITE-SNN: LEVERAGING INHERENT DYNAMICS TO TRAIN ENERGY-EFFICIENT SPIKING NEURAL NETWORKS FOR SEQUENTIAL LEARNING

## 6.1 Introduction

Deep learning is achieving significant milestones in various fields including computer vision [113], natural language processing (NLP) [114], drug discovery [115], autonomous driving [116], and many others. However, the success comes at a significant power and energy cost [1], and if unaddressed, it can hinder the deployment of ubiquitous intelligent edge devices, stagnate or slow down the progress of large model development, and affect various sustainability goals. Specifically, large language models (with trillions of parameters [117]) have recently received a lot of scrutiny for their high energy consumption combined with their popularity and high usage in everyday applications [118]. Researchers are exploring various alternatives at the hardware (custom chips [119]), algorithms (model compression [120], transfer learning [121]), and system-level [122] changes to circumvent the high energy and compute requirements of deep networks. Spiking neural networks (SNNs) running on low-power event-driven neuromorphic platforms [82] is one such solution to improve the energy efficiency of deep neural networks. SNNs are becoming popular for their promise of low power machine intelligence through the asynchronous event-driven computations with binary signals (spikes). SNNs have achieved state-of-the-art results on challenging image classification tasks with better energy efficiency compared to traditional (with ReLU like activations) artificial neural networks (ANNs) [107], [123]. However, their application in other areas like sequential learning tasks (speech recognition, language translation, sentiment analysis, etc.) is not well explored. In this work, we argue that SNNs are better suited for applications that involve sequential tasks because of their similarities to recurrent neural networks (RNNs). We focus on the inherent recurrence dynamics in membrane potential and how it can be leveraged to store past information in sequential inputs. Moreover, we demonstrate that SNNs perform at a lower energy and memory budget compared to different RNN models like long short-term memory (LSTM) [124] and gated recurrent unit (GRU) [125].

Recent improvements in SNN training mechanisms like direct input coding [5], [123], surrogate gradient-based backpropagation [12] have reduced the inference latency from thousands of time-steps to less than ten time-steps [108], [123]. The low inference latency combined with sparse spike-based communication makes SNNs the perfect candidate to solve sequential learning tasks on energy-constrained devices. Although transformer-based models achieve excellent translation quality on NLP tasks [126], they are too big to employ on edge devices to achieve an acceptable battery life. Therefore, the other alternative is to use LSTMs, GRUs, and more recently, convolutional neural networks (CNNs) [127]. In this work, we compare the performance of SNNs with these alternatives to find an energy-efficient lightweight solution for NLP tasks on edge devices. The code and trained models are available at https://github.com/nitin-rathi/LITE-SNN

## 6.2 Background

In this section, we briefly review the necessary background on SNNs. We explain the neuron models employed in all the SNNs developed in this work, discuss the recurrence dynamics in SNN and its similarities to RNNs, and mention the input coding and learning rules used to train the SNNs.

### 6.2.1 Neuron Model

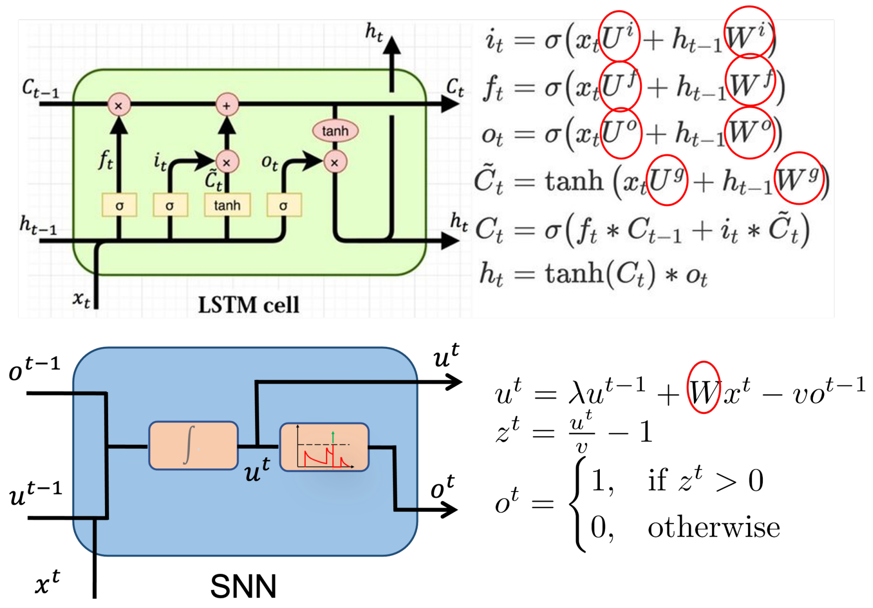
We employ the IF/LIF neuron model described by

$$u_{\mathrm{i}}^{t} = \lambda_{\mathrm{i}} u_{\mathrm{i}}^{t-1} + \sum_{\mathrm{j}} w_{\mathrm{ij}} o_{\mathrm{j}}^{t} - v_{\mathrm{i}} o_{\mathrm{i}}^{t-1} \tag{6.1}$$

$$z_{\mathrm{i}}^{t-1} = \frac{u_{\mathrm{i}}^{t-1}}{v_{\mathrm{i}}} - 1 \quad \text{and} \quad o_{\mathrm{i}}^{t-1} = \begin{cases} 1, & \text{if } z_{\mathrm{i}}^{t-1} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

where $u$ is the membrane potential, $\lambda$ is the leak factor with a value in $[0-1]$, $w$ is the weight connecting pre-neuron j and post-neuron i, $o$ is the binary spike output, $v$ is the firing threshold, and $t$ represents the timestep. For IF neurons, leak ($\lambda$) is equal to 1.

### 6.2.2 Inherent Recurrence Dynamics in SNNs



$h_t$
$C_{t-1}$
$C_t$
tanh
$f_t$
$i_t$
$\tilde{C}_t$
$o_t$
σ
tanh
$h_{t-1}$
$h_t$
$x_t$
LSTM cell
$i_t = \sigma(x_t U^i + h_{t-1} W^i)$
$f_t = \sigma(x_t U^f + h_{t-1} W^f)$
$o_t = \sigma(x_t U^o + h_{t-1} W^o)$
$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$
$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$
$h_t = \tanh(C_t) * o_t$
$o^{t-1}$
$u^t$
$u^{t-1}$
$u^t$
$o^t$
$x^t$
SNN
$u^t = \lambda u^{t-1} + W x^t - v o^{t-1}$
$z^t = \frac{u^t}{v} - 1$
$o^t = \begin{cases} 1, & \text{if } z^t > 0 \\ 0, & \text{otherwise} \end{cases}$

**Figure 6.1.** SNN has 8× less number of weight parameters compared to LSTM

In this section, we study the similarities between RNNs and SNNs [12]. The term RNN refers to networks that have an explicit feedback connection in their internal state to store the history of previous inputs like LSTMs, and GRUs as well as vanilla RNNs [114]. Fig. 6.1 compares the recurrent dynamical equations of LSTM and SNN. Each LSTM layer has 8 weight matrices (similarly GRU has 6, and vanilla RNN has 2) compared to 1 weight matrix for SNN. Therefore, in terms of parameters, SNNs have the least number of trainable weights and require comparatively less storage. Both SNN and RNN receive time-varying input, has an internal state that serves as memory, and generates time-varying output. However, they differ in the type of recurrence. In RNN, the recurrent connection in the hidden state ($h_t$) is explicitly defined by a set of weight matrices whereas, in SNN the recurrence in membrane potential is implicitly defined via leak ($\lambda$). The leak is a single float value between 0 and 1 (shared by all the neurons in the same layer) that controls how much of the previous information is carried forward. SNNs draw their strength from distributing computations over time. The spikes are distributed over T time-steps (we use T=5 for all SNNs in this work) and one forward pass in RNN results in T forward passes in SNNs. However, each forward pass performs a simple addition operation on sparse binary activations and overall, requires less energy (discussed in Sec. 6.6). We exploit this inherent membrane potential dynamics to design SNNs that can compete with RNNs and requires less storage and computational energy.

### 6.2.3 Input coding and Training Mechanism

SNNs compute and communicate information through binary signals (spikes). Therefore, analog inputs like image pixel, word embedding, etc., need to be converted to spike trains. There are various encoding methods like rate coding [6], temporal coding [128], direct coding [5], etc. In rate coding, the analog value is represented as the average firing rate of the neuron whereas, temporal coding encodes the analog value as the time difference between two successive spikes. Direct coding trains a neural network with LIF neurons that accepts analog values as input and generates an output spike train [123]. SNNs trained with direct coding have outperformed other coding methods in terms of latency and energy for image classification. Thus, we employ the direct coding method in all SNNs developed in this work.

The training mechanisms for SNNs can be broadly classified into two categories: ANN-to-SNN conversion [6], and surrogate gradient-based backpropagation [11]. In ANN-to-SNN conversion, a shadow ANN is trained with gradient-descent and the weights are transferred to SNN followed by threshold balancing to determine the firing threshold of each layer. It takes advantage of all the training mechanisms available for ANNs and the conversion to SNN is fast and efficient. However, the inference latency or the number of time-steps for SNNs trained with this method is very high [6]. The derivative of the LIF neuron is discontinuous [97] and standard gradient-descent methods can not be applied directly. Therefore, many surrogate gradients [12] are proposed to approximate the true gradient and enable spike-based backpropagation training in SNNs. Recently, authors in [123] extended the surrogate-gradient-based method to include optimization of threshold voltage and leak along with weights of the network. SNNs trained with this method achieved very low latency and high accuracy, and we employ this training method in all the SNNs developed in this work.

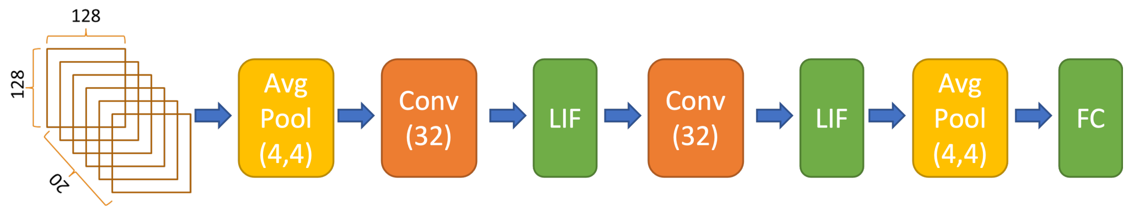


**Figure 6.2.** Gesture recognition with SNNs

In the following sections, we design and train SNNs that employ the input coding and training mechanisms discussed above, and exploit the inherent recurrent dynamics to solve various sequential tasks.

## 6.3 Gesture Recognition

**Table 6.1.** Gesture recognition on IBM DVS128 dataset

| Model | Accuracy | Timesteps |
|---|---|---|
| SLAYER [80] | 93.65% | 300 |
| DECOLLE [10] | 95.54% | 500 |
| Our model | 95.14% | 5 |

Event-based sensors [129] capture the relative motion between the object and the camera. The output of the camera is binary, representing the presence of relative motion. Therefore, these sensors directly generate spikes and are suitable to be processed by spiking networks. Due to the fundamentally different working principle compared to standard cameras, event cameras provide exceptionally high temporal resolution, high dynamic range, no motion blur, and low power consumption. Therefore, event cameras are employed in optical flow estimation [130], gesture recognition [131], object tracking [132], and many other applications. In gesture recognition, the task is to identify the hand gestures from a series of event camera outputs. Each event is represented as $(time,\ x,\ y,\ polarity)$, where *time* is the time of the event, $x, y$ is the location in the frame, and *polarity* is a binary value representing the change at the pixel location. A single gesture consists of thousands of such events. There are many ways [133] to represent a gesture so that it can be processed by a neural network. We select a certain set of events and evenly distribute the events into 20 blocks. All the events in one block are represented by a $128 \times 128$ frame where each location represents the presence of an event for that location in that block (Fig. 6.2). The 20 blocks are combined along the channel dimension, and the entire gesture is represented as $128 \times 128 \times 20$ binary image. The image is processed by an SNN with 2 conv layers, 2 pooling layers, and LIF neurons. We train the network on the IBM DVS128 dataset [133] and report the accuracy and inference latency in Table 6.1. Our model performs better in terms of latency than previous SNN models and the improvements are attributed to optimizing the threshold and leak with gradient-descent that allows the membrane potential to store the information more efficiently.

## 6.4 Sentiment Analysis

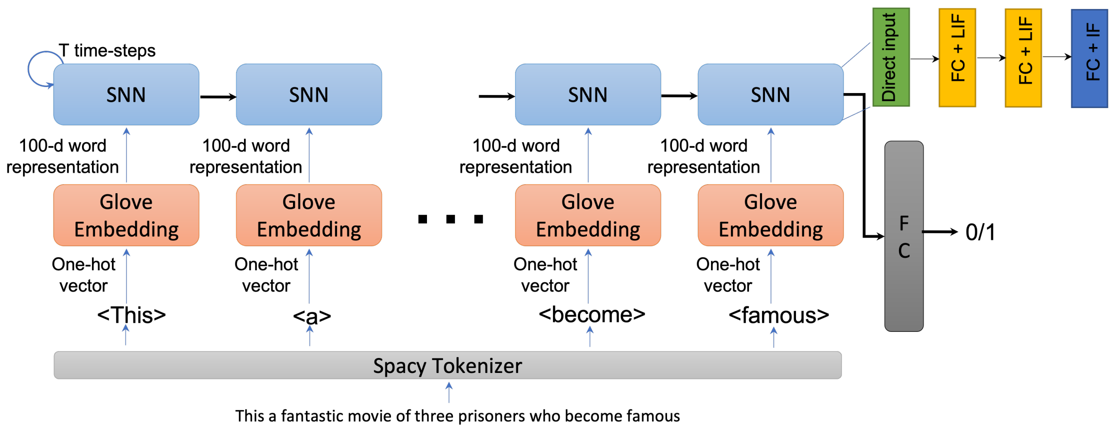


**Figure 6.3.** Network architecture for sentiment analysis with SNNs

**Table 6.2.** Binary classification on IMDB movie reviews

| Model | Accuracy | #Parameters (millions) |
|---|---|---|
| Vanilla RNN | 82.89% | 0.06 |
| LSTM | 89.26% | 0.25 |
| SNN | 88.54% | 0.03 |

We explore the application of SNNs for NLP tasks that have traditionally been solved using RNNs. Sentiment analysis is a binary classification problem where given an input

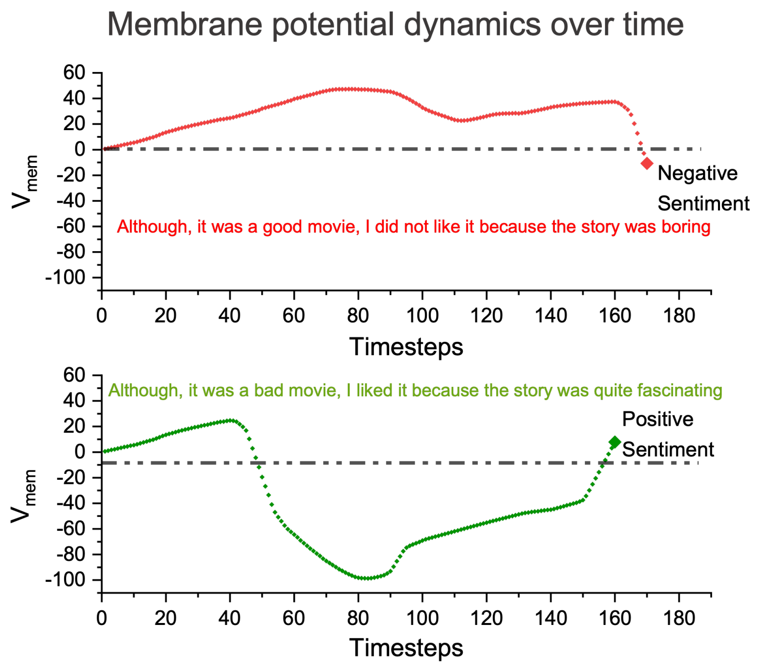


**Figure 6.4.** Change in membrane potential over time of output layer IF neuron for two different inputs. The membrane potential at any time represents the sentiment of all previous words processed till that time. In the top example, there are positive words (‘good movie’) in the beginning and therefore the membrane potential is high, however, the membrane potential goes down in the end to reflect the overall negative sentiment of the input and vice-versa for the bottom example

text, the task is to classify the text as having positive or negative sentiment. The IMDB dataset has $50,000$ labeled movie reviews with $25,000$ reviews used for training and $25,000$ for testing. Fig 6.3 shows the text pre-processing and unrolled SNN architecture employed for sentiment analysis. The spacy tokenizer[1] splits the input sentence into individual tokens that are one-hot encoded. Next, a pre-trained Glove embedding [134] is employed to generate a dense vector representation of the input token. The dense vector is processed by the SNN. SNN has two fully connected (FC) layers with LIF neurons and one FC layer with IF neurons. We employ IF/LIF neuron models because they are simple, require less number of computations to update the membrane potential, and the surrogate-gradient based learning algorithms are well-defined for these models. The state of the SNN is preserved after process-

[1]↑https://spacy.io/

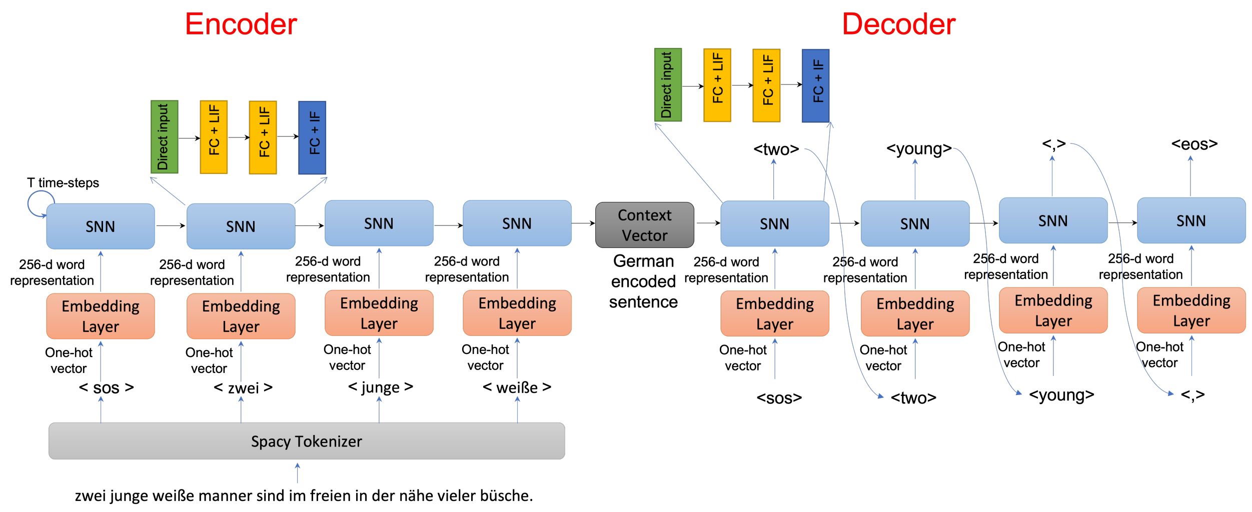


**Figure 6.5.** Sequence to sequence translation with vanilla SNNs

ing each token; in other words, the membrane potentials of LIF and IF neurons are not reset. The history of all the tokens is stored in the membrane potential of LIF and IF neurons. Table 6.2 compares the performance of vanilla RNN, LSTM, and SNN. SNN performs better than vanilla RNN with $2\times$ less parameters and performs within 1% accuracy of LSTM with $8\times$ fewer parameters. The results clearly suggest that SNNs can provide a better alternative than LSTMs to solve NLP tasks on energy-constrained edge devices. We plot the change in membrane potential over time for different input sentences to better understand the SNN dynamics (Fig. 6.4). The membrane potential reflects the sentiment change as the inputs are processed one word at a time.

## 6.5 Sequence to Sequence Learning

In sequence-to-sequence learning tasks, a given sequence of arbitrary length is transformed into another sequence of arbitrary length [135]. For example, summarizing a long paragraph into few sentences, translating text from one language to another, etc.

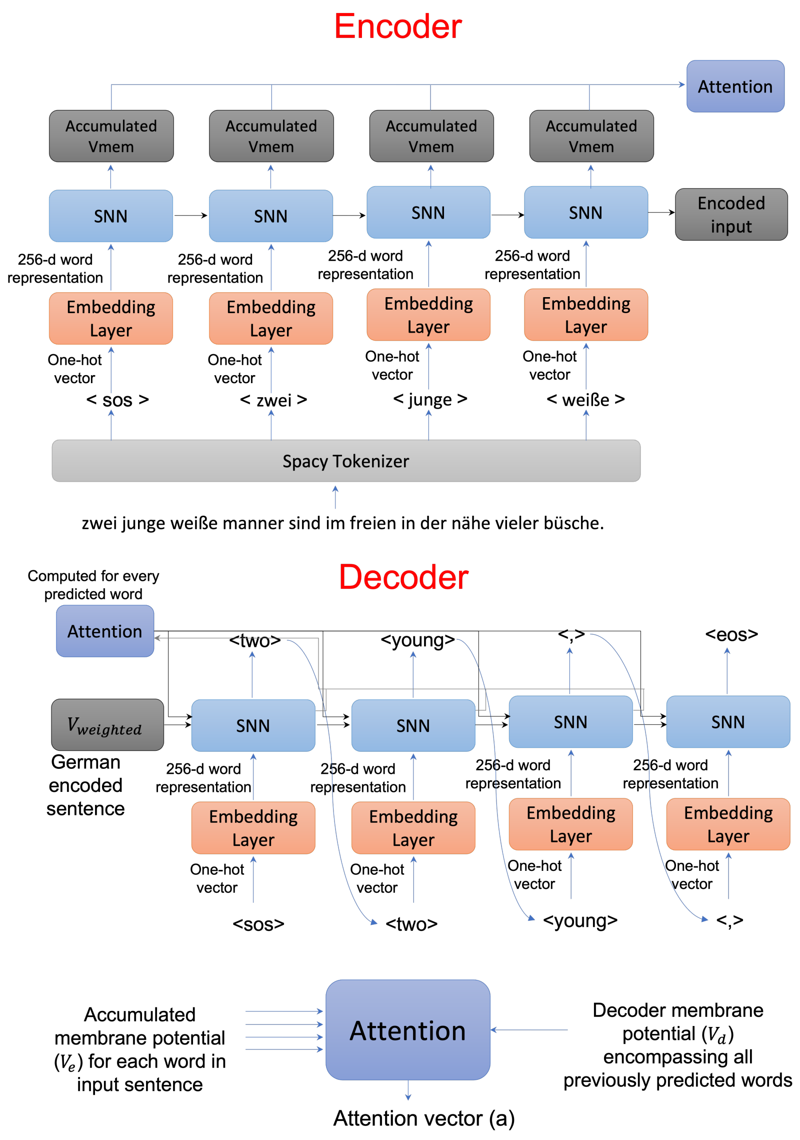
**Table 6.3.** Performance on German to English translation with vanilla SNNs. For SNN, we use two fully-connected layers (to achieve higher score) and therefore we observe less than 6× reduction in parameters compared to GRU

| Encoder | Decoder | BLEU Score | #Parameters (millions) |
|---|---|---|---|
| GRU | GRU | 22 | 12.5 |
| SNN | GRU | 16 | 8.4 |
| GRU | SNN | 20 | 9.4 |
| SNN | SNN | Unable to train | 5.2 |

### 6.5.1 Vanilla SNN

The most common sequence-to-sequence models are encoder-decoder models [135]. The encoder, usually an RNN, encodes the input sequence into a dense vector known as a context vector. The context vector is a representation of the entire input sequence. The decoder, also an RNN, generates the output sequence from the context vector, one word at a time. We replace the RNNs with SNNs in both the encoder and decoder and train the model for German-to-English translation from Multi30k dataset [136]. Fig. 6.5 shows the network architecture with SNN in both encoder and decoder. We also experiment with a combination of GRU and SNN as encoder and decoder (Table 6.3). We employ BLEU score [137] as the evaluation metric and consider N-grams (with N=4) using uniform weights, commonly known as BLEU-4. The model with both encoder and decoder as GRUs performs the best, whereas the model with SNNs fails to train. The issue of vanishing/exploding gradient is well known for recurrent networks [138] including SNNs that have inherent recurrence [139]. The issue is exacerbated in SNNs with two additional factors: 1) time-steps, 2) surrogate gradient. The number of time-steps further increases the length of the sequence as each word is processed for T time-steps. The surrogate gradient is an approximation and long sequences result in many such approximations being multiplied together leading to unstable weight updates. We also observe that encoders have a more significant role than decoders (Table 6.3) because replacing GRU with SNN in the encoder is worse than doing the same for the decoder (BLEU score 16 vs. 20).

### 6.5.2 SNN with Attention



Encoder
Attention
Accumulated Vmem
SNN
Encoded input
256-d word representation
Embedding Layer
One-hot vector
< sos >
< zwei >
< junge >
< weiße >
Spacy Tokenizer
zwei junge weiße manner sind im freien in der nähe vieler büsche.
Decoder
Computed for every predicted word
$V_{weighted}$
German encoded sentence
<two>
<young>
<,>
<eos>
<sos>
Accumulated membrane potential ($V_e$) for each word in input sentence
Decoder membrane potential ($V_d$) encompassing all previously predicted words
Attention vector (a)

$$a_i = Softmax(V_d W + V_e W)$$

$$V_{weighted} = aV_e$$

**Figure 6.6.** Sequence to sequence translation with attention and SNNs

The vanilla encoder-decoder model suffers from information compression as the entire input sequence is represented by a single vector. To address this issue researchers have proposed an attention mechanism to allow the decoder to look at the encoder's intermediate representations at each decoding step [140]. An attention vector ($a$) with same length as the

**Table 6.4.** Performance on German to English translation with SNN and Attention

| Encoder | Decoder | BLEU Score | #Parameters (millions) |
|---|---|---|---|
| GRU | GRU | 33 | 15.7 |
| SNN | GRU | 19 | 11.5 |
| GRU | SNN | 24 | 9.4 |
| SNN | SNN | 15 | 5.2 |

input sequence is used to compute a weighted context vector ($z$) from all the hidden states ($h$) of the encoder ($z = \sum a_i h_i$). Each element ($a_i$) in the attention vector is between 0 and 1 and the entire vector sums to 1. The attention vector is recomputed at every decoding step. This allows the decoder to focus on different input words to generate each output word. The attention vector ($a$) is computed as a function of previous decoder state ($s_{t-1}$) and all encoder hidden states ($h$) ($a_t = f(s_{t-1}, h)$). The elements in the attention vector tell us how much we should attend to each word in the source sentence. For more information on the attention mechanism we refer to [140].

We introduce the attention mechanism to our vanilla SNN models (Fig. 6.6) and treat the accumulated membrane potentials of IF neurons as intermediate hidden states. Similarly, the membrane potential of the decoder SNN at the previous time-step is used to compute the attention for the current time-step. We compare the performance of SNN with GRU for different encoder-decoder combinations (Table 6.4). Compared to vanilla SNNs, we observe that the attention mechanism alleviates the vanishing gradient problem and we can train the model when both the encoder and decoder are SNNs. However, the performance of SNN is worse compared to GRUs. The primary reasons are vanishing gradient and vanishing spike problem. The spiking activity in SNNs decreases drastically for deeper layers because the accumulation in IF/LIF neurons results in lower output spikes than input spikes for each layer (vanishing spike problem). For longer sequences, the spiking activity decreases for latter words and accompanied by the vanishing gradient problem makes the issue worse. Therefore, applying SNNs in models that process words sequentially may not be efficient. In the next section, we discuss how we can employ convolutional SNNs to alleviate the problem.

### 6.5.3 Convolutional SNN with Attentions

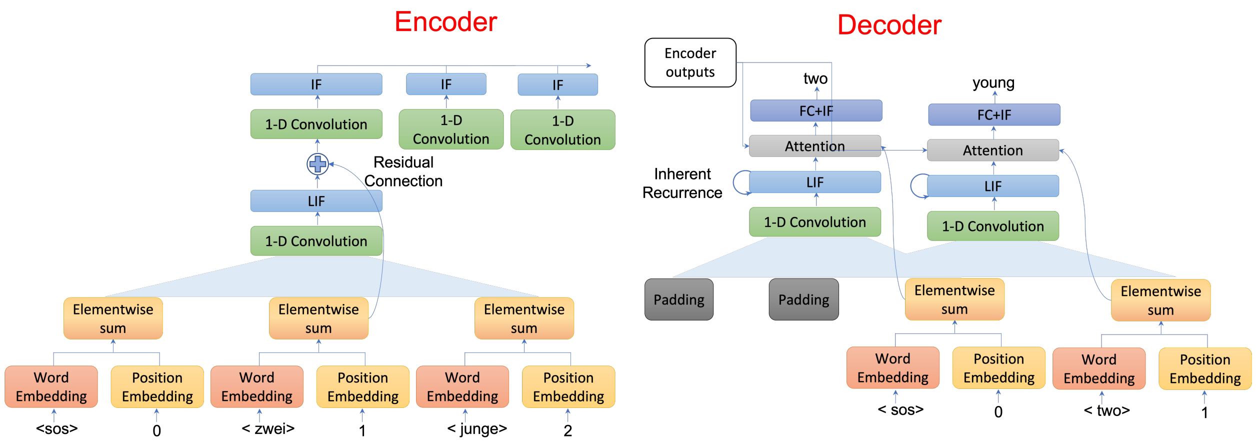


**Figure 6.7.** Sequence to sequence translation with convolutional SNNs

**Table 6.5.** German to English translation with convolutional SNN and Attention. The encoder and decoder both consist of 5 conv layers

| Encoder | Decoder | BLEU Score | #Parameters (millions) |
|---|---|---|---|
| ANN | ANN | 32 | 7.8 |
| SNN | ANN | 34 | 7.8 |
| ANN | SNN | 34 | 7.8 |
| SNN | SNN | 36 | 7.8 |

RNNs combined with attention mechanisms have achieved great success in solving sequence-to-sequence learning tasks [140]. However, these models suffer from gradient propagation and can not be fully parallelized over the input sequence. On the other hand, convolutional neural networks (CNNs) have achieved significant success in computer vision tasks and the computations are fully parallelized. In recent years, several CNN-based models have been proposed for sequence-to-sequence learning that addresses the shortcomings of RNNs [127]. Also, several convolutional SNN models perform extremely well on image classification tasks [6], [123]. Combining the success of training SNNs for image classification with CNN-based sequence to sequence models, we propose a convolutional SNN model for the language translation task. Fig. 6.7 shows the network architecture with 1-D conv layers and LIF neurons in

**Table 6.6.** German to English translation with convolutional SNN and Attention. The encoder and decoder both consist of 10 conv layers

| Encoder | Decoder | BLEU Score | #Parameters (millions) |
|---|---|---|---|
| ANN | ANN | 36 | 15.7 |
| SNN | ANN | 34 | 15.7 |
| ANN | SNN | 35 | 15.7 |
| SNN | SNN | 36 | 15.7 |

both encoder and decoder. The final layer in both encoder and decoder consists of integrate neurons (IF neurons with very high threshold to avoid firing) to accumulate spikes from all time-steps. Since all the input words are processed in parallel, positional embedding is added to encode the order of the words within a sequence. We still employ the attention mechanism (not shown in Fig. 6.7) on all encoder hidden states to compute the context vector. The decoder also processes all the target words in parallel and therefore to ensure that the filters translating token i only look at tokens that appear before i, we add padding only at the beginning of the sentence. As shown in Fig. 6.7, to predict the word 'two' the decoder looks at two padding tokens and the 'sos' token. If the padding was distributed equally in the beginning and end, the conv kernel would look at the word it is trying to predict and will simply learn to copy it instead of learning to translate it. We compare the performance of SNN with conv layers and LIF neurons with an ANN having a similar number of conv layers and ReLU activation. Table 6.5 and 6.6 shows the performance of ANN and SNN with encoder and decoder each having 5 and 10 convolutional layers, respectively. Unlike the $6-8\times$ benefit in the number of parameters as compared to RNNs, the number of parameters in ANN and SNN are similar for the same number of conv layers. SNN has slightly more parameters due to membrane potential and leak. Each neuron has its own membrane potential, whereas leak is shared by all neurons in the same layer. The SNN with 5 conv layers performs similar to the ANN with 10 conv layers. Note, we achieve $2\times$ benefit in the number of parameters for the same BLEU score. SNN's better performance comes from its inherent recurrence in membrane potential. Although convolutional SNNs process the sequence in parallel, the inherent recurrence in SNNs stores the history of nearby words in its membrane potential and provides the benefit of both parallel processing and recurrence. It also solves the spike/gradient vanishing problem by reducing the gradient propagation path to the sum of the number of layers and time-steps. Thus, we believe, SNNs are more suitable for solving sequential tasks with convolutional layers and achieve similar performance as ANNs with 2× less number of parameters.

## 6.6 Energy Efficiency

In this section, we delve into the compute energy efficiency of SNNs. In RNN and ANN, each operation computes a dot-product involving one floating-point (FP) multiplication and one FP addition (MAC), whereas, in SNN, each operation is only one FP addition (AC) due to binary spikes. The computations in SNN implemented on neuromorphic hardware [82], [95] are event-driven and therefore, in the absence of spikes there are no computations and no active energy is consumed. We compute the energy cost/operation in 45nm CMOS technology (Table 6.7). The energy cost for 32-bit MAC operation ($4.6pJ$) is 5.1× more than the AC operation ($0.9pJ$) [109]. These numbers may vary for different technologies, but generally, in most technologies, the addition operation is much cheaper than the multiplication operation. The number of operations in one LSTM and GRU cell is

$$\#OP_{LSTM} = 4 \times (f_{\text{in}} \times f_{out} + f_{out} \times f_{out}) \tag{6.3}$$

$$\#OP_{GRU} = 3 \times (f_{\text{in}} \times f_{out} + f_{out} \times f_{out}) \tag{6.4}$$

**Table 6.7.** Energy costs of addition and multiplication in 45nm CMOS [109]

| | |
|---|---|
| FP ADD (32 bit) | $0.9pJ$ |
| FP MULT (32 bit) | $3.7pJ$ |
| FP MAC (32 bit) | $(0.9 + 3.7) = 4.6pJ$ |

and for one fully-connected SNN layer is

$$\#OP_{SNN} = SpikeRate_l \times f_{\text{in}} \times f_{out} \tag{6.5}$$

$$SpikeRate_l = \frac{\#TotalSpikes_l \text{ over all inference timesteps}}{\#Neurons_l} \tag{6.6}$$

where $f_{\text{in}}(f_{out})$ is the number of neurons in input (output) layer, $SpikeRate_l$ is the total spikes in layer $l$ over all timesteps divided by the number of neurons in layer $l$. Therefore, in SNNs the number of operations can be reduced by both lowering the number of parameters and reducing the spike rate. Table 6.8 shows the compute energy of different models for sentiment analysis task discussed in section 6.4. Since we employ direct input coding for SNNs, the first layer performs MAC operations whereas the other layers perform AC operations. For RNN and LSTM, all computations are MAC operations. Compared to LSTM, SNN achieves $42\times$ benefit in compute energy where $\sim 8\times$ comes from reduction in parameters, $\sim 5\times$ from the difference in MAC vs AC, and the rest from the sparsity in SNNs.

**Table 6.8.** Energy-efficiency of various sentiment analysis models

| Model | #MACs (millions) | #ACs (millions) | Spike-rate | Normalized Energy (lower is better) | Accuracy |
|---|---|---|---|---|---|
| Vanilla RNN | 0.06 | 0 | N/A | 10.5 | 82.89% |
| LSTM | 0.25 | 0 | N/A | 42.1 | 89.26% |
| SNN | 0.013 | 0.017 | 0.37 | 1.0 | 88.54% |

Next, we compare the energy of ANN and SNN with convolutional layers. The number of operations in a conv layer is

$$\#OP_{conv} = k_w \times k_h \times c_{\text{in}} \times h_{out} \times w_{out} \times c_{out} \tag{6.7}$$

**Table 6.9.** Energy-efficiency of various language translation models

| Model | #MACs (millions) | #ACs (millions) | Spike-rate | Normalized Energy (lower is better) | BLEU-4 |
|---|---|---|---|---|---|
| GRU | 15.7 | 0 | N/A | 28.0 | 33 |
| SNN | 3.1 | 2.1 | 0.32 | 1.8 | 15 |
| Conv(5) | 7.8 | 0 | N/A | 13.9 | 32 |
| ConvSNN(5) | 1.6 | 6.3 | 0.35 | 1.0 | 36 |
| Conv(10) | 15.7 | 0 | N/A | 28.0 | 36 |
| ConvSNN(10) | 1.6 | 14.1 | 0.36 | 1.03 | 36 |

and the number of operations in a conv SNN layer is

$$\#OP_{conv-snn} = SpikeRate_l \times \#OP_{conv} \tag{6.8}$$

where $k_w(k_h)$ is kernel width (height), $c_{in}(c_{out})$ is the number of input (output) channels, $h_{out}(w_{out})$ is the height (width) of the output feature map. A spike rate of 1 (every neuron fired once) implies that the number of operations for ANN and SNN are the same (though operations are MAC in ANN while addition in SNNs). Lower spike rates denote more sparsity in spike events and higher energy-efficiency. Table 6.9 shows the compute energy comparison for different language translation models discussed in section 6.5. Convolutional SNN with 5 conv layers in both encoder and decoder achieves the minimum energy and the highest performance. To achieve same performance ANN requires 28× more energy. Therefore, employing SNNs for sequence-to-sequence learning tasks results in energy-efficient models.

## 6.7 Conclusions

SNNs have achieved decent performance on image classification tasks, and recent developments in training methodologies have improved their inference latency and energy. In this work, we further extended the capabilities of SNNs to handle sequential tasks in energy and memory-efficient manner. SNNs are dynamical systems and are naturally better suited to handle sequential inputs than static images. We showed, through experiments, on various NLP tasks, how SNNs provide advantages over LSTMs/GRUs. SNNs require 8$\times$ lower storage compared to LSTMs while performing similarly on sentiment analysis tasks. For language translation tasks, convolutional SNNs require 2$\times$ less parameters than ANNs for same performance and achieve better energy-efficiency.

# 7. SUMMARY AND FUTURE DIRECTIONS

Spiking neural networks (SNNs) are brain-inspired emerging machine learning models that aim to address the high energy cost of current deep learning models. SNNs compute and communicate via event-driven binary signals (spikes) distributed over time. The asynchronous computing combined with low-power neuromorphic hardware makes SNNs a perfect candidate to enable energy-efficient machine intelligence on next-generation battery-operated edge devices.

In this dissertation, we proposed several training algorithms to design low latency, high accuracy, high sparsity SNNs for image classification and sequential learning tasks. We explored bio-plausible unsupervised learning methods with multimodal inputs, proposed hybrid learning techniques that reduce training time, presented learning methodologies that drastically reduce inference latency, and developed SNN models that exploit inherent recurrence in spiking neurons to solve sequential tasks. We showed that with advancements in SNN training algorithms, the latency of inference has been significantly reduced from thousands of timesteps to a few tens of timesteps while maintaining significant spike sparsity. This highlights SNN's prowess towards utilizing the temporal information in a much more productive manner, potentially achieving real-time and energy-efficient inference. Although not discussed in detail in this research, but the efficacy of the SNNs largely depends on the underlying hardware architecture. Since SNNs are event-driven, to reap the full benefits, the hardware running these models should be able to support sparse and asynchronous operations.

Going forward, there is a need to develop stable learning algorithms to train SNNs faster. Current gradient-based learning methods are slow and memory intensive as they rely on unrolling the network over time. We extensively explored the application of SNNs for image classification and briefly delved into natural language processing tasks. With the tools and algorithms developed in this dissertation, it is worth exploring other applications like object detection, scene segmentation, speech recognition, etc., that can benefit from the energy efficiency of SNNs. Sequential tasks, in particular, may benefit from the spatio-temporal processing in SNNs.

# REFERENCES

[1] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "Deep learning's diminishing returns: The cost of improvement is becoming unsustainable," *IEEE Spectrum*, vol. 58, no. 10, pp. 50–55, 2021.

[2] D. Li, X. Chen, M. Becchi, and Z. Zong, "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)*, IEEE, 2016, pp. 477–484.

[3] D. D. Cox and T. Dean, "Neural networks and neuroscience-inspired computer vision," *Current Biology*, vol. 24, R921–R929, 2014.

[4] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[5] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[6] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, 2019.

[7] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016.

[8] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in neuroscience*, vol. 12, p. 774, 2018.

[9] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.

[10] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.

[11] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems*, 2018, pp. 787–797.

[12] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *arXiv preprint arXiv:1901.09948*, 2019.

[13] D. S. Modha, “Introducing a brain-inspired computer,” *Published online at http://www. research. ibm. com/articles/brain-chip. shtml*, 2017.

[14] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[15] P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in computational neuroscience*, vol. 9, 2015.

[16] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” *Frontiers in neuroscience*, vol. 10, 2016.

[17] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, “A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm,” in *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, IEEE, 2011, pp. 1–4.

[18] Y.-P. Lin, C. H. Bennett, T. Cabaret, D. Vodenicarevic, D. Chabi, D. Querlioz, B. Jousselme, V. Derycke, and J.-O. Klein, “Physical realization of a supervised learning system built with organic memristive synapses,” *Scientific reports*, vol. 6, p. 31 932, 2016.

[19] A. Ankit, A. Sengupta, P. Panda, and K. Roy, “Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks,” in *Proceedings of the 54th Annual Design Automation Conference 2017*, ACM, 2017, p. 27.

[20] B. Liu, W. Wen, Y. Chen, X. Li, C.-R. Wu, and T.-Y. Ho, “Eda challenges for memristor-crossbar based neuromorphic computing,” in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, ACM, 2015, pp. 185–188.

[21] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, “Complementary resistive switches for passive nanocrossbar memories,” *Nature materials*, vol. 9, no. 5, p. 403, 2010.

[22] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, “Memristor crossbar-based neuromorphic computing system: A case study,” *IEEE transactions on neural networks and learning systems*, vol. 25, no. 10, pp. 1864–1878, 2014.

[23] S. Park, H. Kim, M. Choo, J. Noh, A. Sheri, S. Jung, K. Seo, J. Park, S. Kim, W. Lee, *et al.*, “Rram-based synapse for neuromorphic system with pattern recognition function,” in *Electron Devices Meeting (IEDM), 2012 IEEE International*, IEEE, 2012, pp. 10–2.

[24] P. Krzysteczko, J. Münchenberger, M. Schäfers, G. Reiss, and A. Thomas, “The memristive magnetic tunnel junction as a nanoscopic synapse-neuron system,” *Advanced Materials*, vol. 24, no. 6, pp. 762–766, 2012.

[25] M. Sharad, C. Augustine, G. Panagopoulos, and K. Roy, “Spin-based neuron model with domain-wall magnets as synapse,” *IEEE Transactions on Nanotechnology*, vol. 11, no. 4, pp. 843–853, 2012.

[26] P. R. Huttenlocher *et al.*, “Synaptic density in human frontal cortex-developmental changes and effects of aging,” *Brain Res*, vol. 163, no. 2, pp. 195–205, 1979.

[27] G. Chechik, I. Meilijson, and E. Ruppin, “Synaptic pruning in development: A computational account,” *Neural computation*, vol. 10, no. 7, pp. 1759–1777, 1998.

[28] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in *Advances in Neural Information Processing Systems 2*, 1990, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage.pdf.

[29] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *International Conference on Learning Representations (ICLR)*, 2015.

[30] S. Dora, S. Sundaram, and N. Sundararajan, “A two stage learning algorithm for a growing-pruning spiking neural network for pattern classification problems,” in *Neural Networks (IJCNN), 2015 International Joint Conference on*, IEEE, 2015, pp. 1–7.

[31] J. Iglesias, J. Eriksson, F. Grize, M. Tomassini, and A. E. Villa, “Dynamics of pruning in simulated large-scale spiking neural networks,” *Biosystems*, vol. 79, no. 1-3, pp. 11–20, 2005.

[32] H. Graf, L. Jackel, R. Howard, B. Straughn, J. Denker, W. Hubbard, D. Tennant, and D. Schwartz, “Vlsi implementation of a neural network memory with several hundreds of neurons,” in *AIP conference proceedings*, AIP, vol. 151, 1986, pp. 182–187.

[33] D. Goodman and R. Brette, “Brian: A simulator for spiking neural networks in python,” *Frontiers in neuroinformatics*, vol. 2, 2008.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[36] G. Bradski, “The opencv library.,” *Dr. Dobb’s Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[37] E. J. Merced-Grafals, N. Dávila, N. Ge, R. S. Williams, and J. P. Strachan, “Repeatable, accurate, and high speed multi-level programming of memristor 1t1r arrays for power efficient analog computing applications,” *Nanotechnology*, vol. 27, no. 36, p. 365 202, 2016.

[38] H. McGurk and J. MacDonald, “Hearing lips and seeing voices,” *Nature*, vol. 264, no. 5588, p. 746, 1976.

[39] G. A. Calvert, “Crossmodal processing in the human brain: Insights from functional neuroimaging studies,” *Cerebral cortex*, vol. 11, no. 12, pp. 1110–1123, 2001.

[40] B. E. Stein and M. A. Meredith, *The merging of the senses.* The MIT Press, 1993.

[41] K. Von Kriegstein and A.-L. Giraud, “Implicit multisensory associations influence voice recognition,” *PLoS biology*, vol. 4, no. 10, e326, 2006.

[42] S. M. Bohte, J. N. Kok, and J. A. La Poutré, “Spikeprop: Backpropagation for networks of spiking neurons.,” in *ESANN*, 2000, pp. 419–424.

[43] S. R. Kulkarni, J. M. Alexiades, and B. Rajendran, “Learning and real-time classification of hand-written digits with spiking neural networks,” in *Electronics, Circuits and Systems (ICECS), 2017 24th IEEE International Conference on*, IEEE, 2017, pp. 128–131.

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

[45] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, “Stdp-based spiking deep convolutional neural networks for object recognition,” *Neural Networks*, vol. 99, pp. 56–67, 2018.

[46] J. J. Wade, L. J. McDaid, J. A. Santos, and H. M. Sayers, “Swat: A spiking neural network training algorithm for classification problems,” *IEEE Transactions on Neural Networks*, vol. 21, no. 11, pp. 1817–1830, 2010.

[47] Y. Zhang, P. Li, Y. Jin, and Y. Choe, “A digital liquid state machine with biologically inspired learning and its application to speech recognition,” *IEEE transactions on neural networks and learning systems*, vol. 26, no. 11, pp. 2635–2649, 2015.

[48] M. Haller, H.-G. Kim, and T. Sikora, “Audiovisual anchorperson detection for topic-oriented navigation in broadcast news,” in *Multimedia and Expo, 2006 IEEE International Conference on*, IEEE, 2006, pp. 1817–1820.

[49] N. Srivastava and R. Salakhutdinov, “Learning representations for multimodal data with deep belief nets,” in *International conference on machine learning workshop*, vol. 79, 2012.

[50] N. Srivastava and R. R. Salakhutdinov, “Multimodal learning with deep boltzmann machines,” in *Advances in neural information processing systems*, 2012, pp. 2222–2230.

[51] C. Hong, J. Yu, J. Wan, D. Tao, and M. Wang, “Multimodal deep autoencoder for human pose recovery,” *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5659–5670, 2015.

[52] A. Wang, J. Lu, J. Cai, T.-J. Cham, and G. Wang, “Large-margin multi-modal deep learning for rgb-d object recognition,” *IEEE Transactions on Multimedia*, vol. 17, no. 11, pp. 1887–1898, 2015.

[53] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, “Multimodal deep learning for robust rgb-d object recognition,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, IEEE, 2015, pp. 681–687.

[54] C. Sanderson and K. K. Paliwal, “Identity verification using speech and face information,” *Digital Signal Processing*, vol. 14, no. 5, pp. 449–480, 2004.

[55] P. Panda, G. Srinivasan, and K. Roy, “Ensemblesnn: Distributed assistive stdp learning for energy-efficient recognition in spiking neural networks,” in *Neural Networks (IJCNN), 2017 International Joint Conference on*, IEEE, 2017, pp. 2629–2635.

[56] R. Lyon, “A computational model of filtering, detection, and compression in the cochlea,” in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP’82.*, IEEE, vol. 7, 1982, pp. 1282–1285.

[57] M. Slaney, “Auditory toolbox,” *Interval Research Corporation, Tech. Rep*, vol. 10, p. 1998, 1998.

[58] M. Liberman *et al.*, “Ti 46-word ldc93s9,” *Web Download. Philadelphia: Linguistic Data Consortium*, 1993.

[59] S. Basu, M. Karki, S. Ganguly, R. DiBiano, S. Mukhopadhyay, S. Gayaka, R. Kannan, and R. Nemani, “Learning sparse feature representations using probabilistic quadtrees and deep belief nets,” *Neural Processing Letters*, vol. 45, no. 3, pp. 855–867, 2017.

[60] P. Panda, J. M. Allred, S. Ramanathan, and K. Roy, “Asp: Learning to forget with adaptive synaptic plasticity in spiking neural networks,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2017.

[61] M. Asano, M. Imai, S. Kita, K. Kitajo, H. Okada, and G. Thierry, “Sound symbolism scaffolds language development in preverbal infants,” *cortex*, vol. 63, pp. 196–205, 2015.

[62] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, “Direct training for spiking neural networks: Faster, larger, better,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1311–1318.

[63] E. Hunsberger and C. Eliasmith, “Spiking deep networks with lif neurons,” *arXiv preprint arXiv:1510.08829*, 2015.

[64] Y. Cao, Y. Chen, and D. Khosla, “Spiking deep convolutional neural networks for energy-efficient object recognition,” *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[65] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, “Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware,” in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, IEEE, 2016, pp. 1–8.

[66] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[67] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[68] D. Huh and T. J. Sejnowski, “Gradient descent for spiking neural networks,” in *Advances in Neural Information Processing Systems*, 2018, pp. 1433–1443.

[69] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, “Spatio-temporal backpropagation for training high-performance spiking neural networks,” *Frontiers in neuroscience*, vol. 12, 2018.

[70] C. Lee, S. S. Sarwar, and K. Roy, “Enabling spike-based backpropagation in state-of-the-art deep neural network architectures,” *arXiv preprint arXiv:1903.06379*, 2019.

[71] P. J. Werbos *et al.*, “Backpropagation through time: What it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[72] S. Song, K. D. Miller, and L. F. Abbott, “Competitive hebbian learning through spike-timing-dependent synaptic plasticity,” *Nature neuroscience*, vol. 3, no. 9, p. 919, 2000.

[73] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[74] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[75] Y.-L. Boureau, J. Ponce, and Y. LeCun, “A theoretical analysis of feature pooling in visual recognition,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 111–118.

[76] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.

[77] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[78] S. M. Bohte, J. N. Kok, and J. A. La Poutré, “Spikeprop: Backpropagation for networks of spiking neurons.,” in *ESANN*, 2000, pp. 419–424.

[79] F. Zenke and S. Ganguli, “Superspike: Supervised learning in multilayer spiking neural networks,” *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018.

[80] S. B. Shrestha and G. Orchard, “Slayer: Spike layer error reassignment in time,” in *Advances in Neural Information Processing Systems*, 2018, pp. 1412–1421.

[81] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, “Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.

[82] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[83] Z. F. Mainen and T. J. Sejnowski, “Reliability of spike timing in neocortical neurons,” *Science*, vol. 268, no. 5216, pp. 1503–1506, 1995.

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[85] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[86] K. Freund. (2019). “Google cloud doubles down on nvidia gpus for inference,” [Online]. Available: https://www.forbes.com/sites/moorinsights/2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference/.

[87] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[88] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 784–800.

[89] I. Chakraborty, D. Roy, I. Garg, A. Ankit, and K. Roy, "Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence," *Nature Machine Intelligence*, pp. 1–13, 2020.

[90] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*, Springer, 2016, pp. 525–542.

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

[92] S. R. Kheradpisheh and T. Masquelier, "Temporal backpropagation for spiking neural networks with one spike per neuron," *International Journal of Neural Systems*, 2020.

[93] A. Almomani, M. Alauthman, M. Alweshah, O. Dorgham, and F. Albalas, "A comparative study on spiking neural network encoding schema: Implemented with cloud computing," *Cluster Computing*, vol. 22, no. 2, pp. 419–433, 2019.

[94] C. Lee, A. Kosta, A. Z. Zhu, K. Chaney, K. Daniilidis, and K. Roy, "Spike-flownet: Event-based optical flow estimation with energy-efficient hybrid neural networks," *arXiv preprint arXiv:2003.06696*, 2020.

[95] E. P. Frady, G. Orchard, D. Florey, N. Imam, R. Liu, J. Mishra, J. Tse, A. Wild, F. T. Sommer, and M. Davies, "Neuromorphic nearest-neighbor search using intel's pohoiki springs," *arXiv preprint arXiv:2004.12691*, 2020.

[96] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge university press, 2002.

[97] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1xSperKvH.

[98] B. Han, G. Srinivasan, and K. Roy, “Rmp-snns: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural networks,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[99] H. Mostafa, “Supervised learning based on temporal coding in spiking neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3227–3235, 2017.

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

[101] B. Yin, F. Corradi, and S. M. Bohté, “Effective and efficient computation with multiple-timescale spiking recurrent neural networks,” *arXiv preprint arXiv:2005.11633*, 2020.

[102] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[103] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, “A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5394–5407, 2018.

[104] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, “Random synaptic feedback weights support error backpropagation for deep learning,” *Nature communications*, vol. 7, no. 1, pp. 1–10, 2016.

[105] A. Samadi, T. P. Lillicrap, and D. B. Tweed, “Deep learning with dynamic spiking neurons and fixed feedback weights,” *Neural computation*, vol. 29, no. 3, pp. 578–602, 2017.

[106] S. Lu and A. Sengupta, “Exploring the connection between binary and spiking neural networks,” *Frontiers in Neuroscience*, vol. 14, p. 535, 2020, ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00535. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2020.00535.

[107] J. Wu, Y. Chua, M. Zhang, G. Li, H. Li, and K. C. Tan, “A tandem learning rule for efficient and rapid inference on deep spiking neural networks,” *arXiv*, arXiv–1907, 2019.

[108] W. Zhang and P. Li, “Temporal spike sequence learning via backpropagation for deep spiking neural networks,” *arXiv preprint arXiv:2002.10085*, 2020.

[109] M. Horowitz, "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)*, IEEE, 2014, pp. 10–14.

[110] Y. Shen, M. Ferdman, and P. Milder, "Escher: A cnn accelerator with flexible buffering to minimize off-chip transfer," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2017, pp. 93–100.

[111] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016.

[112] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.

[113] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[114] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[115] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, "The rise of deep learning in drug discovery," *Drug discovery today*, vol. 23, no. 6, pp. 1241–1250, 2018.

[116] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[117] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *arXiv preprint arXiv:2101.03961*, 2021.

[118] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.

[119] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2018.

[120] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[121] L. Torrey and J. Shavlik, “Transfer learning,” in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010, pp. 242–264.

[122] B. Acun, M. Murphy, X. Wang, J. Nie, C.-J. Wu, and K. Hazelwood, “Understanding training efficiency of deep learning recommendation models at scale,” in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2021, pp. 802–814.

[123] N. Rathi and K. Roy, “Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization,” *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[124] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[125] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

[126] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[127] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, “Convolutional sequence to sequence learning,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 1243–1252.

[128] M. Zhang, Z. Gu, N. Zheng, D. Ma, and G. Pan, “Efficient spiking neural networks with logarithmic temporal coding,” *IEEE Access*, vol. 8, pp. 98 156–98 167, 2020.

[129] L. Patrick, C. Posch, and T. Delbruck, “A 128x 128 120 db 15$\mu$ s latency asynchronous temporal contrast vision sensor,” *IEEE journal of solid-state circuits*, vol. 43, pp. 566–576, 2008.

[130] C. Lee, A. K. Kosta, and K. Roy, “Fusion-flownet: Energy-efficient optical flow estimation using sensor fusion and deep fused spiking-analog network architectures,” *arXiv preprint arXiv:2103.10592*, 2021.

[131] R. Massa, A. Marchisio, M. Martina, and M. Shafique, “An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor,” *arXiv preprint arXiv:2006.09985*, 2020.

[132] H. Liu, D. P. Moeys, G. Das, D. Neil, S.-C. Liu, and T. Delbrück, "Combined frame-and event-based detection and tracking," in *2016 IEEE International Symposium on Circuits and systems (ISCAS)*, IEEE, 2016, pp. 2511–2514.

[133] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[134] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[135] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[136] D. Elliott, S. Frank, K. Sima'an, and L. Specia, "Multi30k: Multilingual english-german image descriptions," *arXiv preprint arXiv:1605.00459*, 2016.

[137] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[138] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[139] W. Ponghiran and K. Roy, "Spiking neural networks with improved inherent recurrence dynamics for sequential learning," *arXiv preprint arXiv:2109.01905*, 2021.

[140] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

# VITA

Nitin Rathi received B.Tech. degree in electronics and communications engineering from the West Bengal University of Technology, Kolkata, India, in 2013. In 2016, he joined the Ph.D. program in electrical and computer engineering at Purdue University, West Lafayette, IN, USA. He worked as a machine learning intern with GlobalFoundries, Santa Clara, during summer 2020 where he developed machine learning algorithms to identify defects from chip design images. His research interests include machine learning algorithms, neuromorphic computing, and energy-efficient deep learning.

## PUBLICATIONS

1. **N. Rathi**, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in International Conference on Learning Representations (ICLR), 2020.

2. **N. Rathi** and K. Roy, "Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization,"IEEE Transactions on Neural Networks and Learning Systems (TNNLS), 2021.

3. **N. Rathi**, P. Panda, and K. Roy, "Stdp-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition,"IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2018.

4. **N. Rathi** and K. Roy, "Stdp-based unsupervised multimodal learning with cross-modal processing in spiking neural network,"IEEE Transactions on Emerging Topics in Computational Intelligence (TETCI), 2018.

5. **N. Rathi**, A. Agrawal, C. Lee, A. K. Kosta, and K. Roy, "Exploring spike-based learning for neuromorphic computing: Prospects and perspectives," in 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021.

6. S. Sharmin, **N. Rathi**, P. Panda, and K. Roy, "Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations," in European Conference on Computer Vision (ECCV), 2020.

7. A. Agrawal, M. Ali, M. Koo, **N. Rathi**, A. Jaiswal, and K. Roy, "Impulse: A 65-nmdigital compute-in-memory macro with fused weights and membrane potential for spike-based sequential learning tasks,"IEEE Solid-State Circuits Letters, 2021.