UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

TRAINING SPIKING NEURAL NETWORKS

Diploma Thesis

Panagiotis Tsoukas

Supervisor: Dimitrios Rafailidis

Octomber 2023

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

TRAINING SPIKING NEURAL NETWORKS

Diploma Thesis

Panagiotis Tsoukas

Supervisor: Dimitrios Rafailidis

Octomber 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΚΠΑΙΔΕΥΣΗ ΑΚΙΔΩΤΩΝ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ

Διπλωματική Εργασία

Παναγιώτης Τσούκας

Επιβλέπων/πουσα: Δημήτριος Ραφαηλίδης

Οκτώβριος 2023

Approved by the Examination Committee:

Supervisor	**Dimitrios Rafailidis** Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Christos Antonopoulos** Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Nikolaos Bellas** Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

Throughout the course of writing my diploma thesis, I received significant assistance and encouragement. I would like to extend my heartfelt gratitude to my professors, Mr. Dimitrios Rafailidis and Mr. Dimitrios Katsaros. Their wisdom, expertise, and unwavering belief, guided me through this research journey.

I must not overlook the guidance and priceless advice provided by all my professors. Particularly Mr. Elias Houstis, Professor Emeritus at the University of Thessaly, who inspired me in the early phases of my degree with his presentations and research, letting me choose this department but also earning my utmost respect.

Last but certainly not least, I want to convey my love and gratitude to my family and my closest friends for their unwavering support and unwavering faith in me.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Panagiotis Tsoukas

Diploma Thesis

TRAINING SPIKING NEURAL NETWORKS

Panagiotis Tsoukas

Abstract

The quest to replicate the intricate capabilities of the human brain in artificial intelligence has long intrigued researchers. Although traditional artificial neural networks have achieved impressive results, they struggle to match the innate efficiency and adaptability of biological neurons. Spiking Neural Networks (SNNs) present a compelling avenue for reconciling this disparity by emulating neuron spiking behavior.However, the conventional shallow architectures of SNNs have limitations when it comes to effectively representing intricate information. Efforts to train deep SNNs using input spikes have, thus far, not yielded successful outcomes. Various approaches have been proposed to circumvent this challenge, including the conversion of pre-trained deep Artificial Neural Networks (ANNs) into SNNs. Nevertheless, this conversion method falls short in capturing the nuanced temporal dynamics inherent in spiking systems. Conversely, the direct training of deep SNNs with input spike events remains a formidable task due to the discontinuous and non-differentiable nature of spike generation. To address this challenge, we propose an approximate derivative method that takes into account the leaky behavior exhibited by LIF neurons. This novel approach empowers the direct training of deep convolutional SNNs, using spike-based backpropagation with input spike events. Our experimental results validate the effectiveness of this spike-based learning technique on deep network architectures such as VGG and Residual models, consistently achieving superior classification accuracies on the MNIST, SVHN, and CIFAR-10 datasets when compared to other SNNs trained using spike-based learning. Furthermore, we conduct a comprehensive analysis of sparse event-based computations to underscore the efficiency of our proposed SNN training method for inference operations within the spiking domain.Our experimental results validate the effectiveness of this spike-based learning technique on deep network architectures such as VGG and Residual models, consistently achieving really good classification accuracies on the MNIST, SVHN and CIFAR-10 datasets when compared to other SNNs trained using spike-based learning.

Διπλωματική Εργασία

ΕΚΠΑΙΔΕΥΣΗ ΑΚΙΔΩΤΩΝ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ

Παναγιώτης Τσούκας

Περίληψη

Η αναζήτηση της αναπαραγωγής των περίπλοκων δυνατοτήτων του ανθρώπινου εγκεφάλου στην τεχνητή νοημοσύνη απασχολεί εδώ και καιρό τους ερευνητές. Αν και τα παραδοσιακά τεχνητά νευρωνικά δίκτυα έχουν επιτύχει εντυπωσιακά αποτελέσματα, αγωνίζονται να φτάσουν την έμφυτη αποτελεσματικότητα και προσαρμοστικότητα των βιολογικών νευρώνων. Τα ακιδωτά νευρωνικά δίκτυα (SNNs) παρουσιάζουν μια συναρπαστική οδό για τη συμφιλίωση αυτής της ασυμφωνίας με τη μίμηση της συμπεριφοράς των νευρώνων που εκτελούν.Ωστόσο, οι συμβατικές ρηχές αρχιτεκτονικές των SNNs έχουν περιορισμούς όσον αφορά την αποτελεσματική αναπαράσταση περίπλοκων πληροφοριών. Οι προσπάθειες για την εκπαίδευση βαθιών SNNs με τη χρήση ακίδων εισόδου δεν έχουν, μέχρι στιγμής, αποφέρει επιτυχή αποτελέσματα. Έχουν προταθεί διάφορες προσεγγίσεις για την παράκαμψη αυτής της πρόκλησης, συμπεριλαμβανομένης της μετατροπής προ-εκπαιδευμένων βαθιών Τεχνητών Νευρωνικών Δικτύων (ANNs) σε SNN. Παρ' όλα αυτά, αυτή η μέθοδος μετατροπής υπολείπεται στην αποτύπωση της διαφοροποιημένης χρονικής δυναμικής που χαρακτηρίζει τα συστήματα ακίδων. Αντίθετα, η άμεση εκπαίδευση βαθιών SNN με γεγονότα ακίδων εισόδου παραμένει ένα δύσκολο έργο λόγω της ασυνεχούς και μη διαφοροποιήσιμης φύσης του τρόπου παραγωγής ακίδων.Για την αντιμετώπιση αυτής της πρόκλησης, προτείνουμε μια μέθοδο προσεγγιστικής παραγώγισης που λαμβάνει υπόψη τη διαρρέουσα συμπεριφορά που παρουσιάζουν οι νευρώνες LIF. Αυτή η νέα προσέγγιση ενδυναμώνει την άμεση εκπαίδευση βαθιών συνελικτικών SNN, χρησιμοποιώντας οπίσθια διάδοση με βάση τις εκτελέσεις των ακιδωτών γεγονότων.Τα πειραματικά μας αποτελέσματα επικυρώνουν την αποτελεσματικότητα αυτής της τεχνικής μάθησης με βάση τις ακίδες σε αρχιτεκτονικές βαθιών δικτύων, όπως τα μοντέλα VGG και Residual, επιτυγχάνοντας αρκετά καλές τιμές ποσοστών ακρίβειας για ταξινόμηση στα σύνολα δεδομένων MNIST, SVHN και CIFAR-10 σε σύγκριση με άλλα SNN.

Table of contents

List of figures

List of tables

Abbreviations

ANN	Artifical Neural Network
BP	Back Propagation
CNN	Convolution Neural Network
DNN	Deep Neural Network
LIF	Leaky Integrate and Fire
MNIST	Modified National Institute of Standards and Technology database
N-MNIST	Neuromorphic Modified National Institute of Standards and Technology database
SNN	Spiking Neural Network
SVHN	Street View House Numbers
STDP	Spike timing dependent plasticity

Chapter 1

Introduction

1.1 Thesis Subject

In recent years, deep learning has emerged as a revolutionary force, rapidly reshaping and enhancing our approach to an assortment of cognitive tasks. The field has witnessed considerable advancements, notably in object detection, where algorithms meticulously sift through visual data, recognizing and categorizing entities with a precision that edges ever closer to human capabilities. Numerous deep learning techniques [1], [2], [3] facilitate the efficient tuning of deep Artificial Neural Networks (ANNs) by building multiple layers of feature hierarchies, demonstrating impressive outcomes that sometimes surpass human-level performance.However, due to the enormous number of parameters, immense training samples, and tremendous energy consumption required by ANN training and deployment, ANN is currently challenging to apply to edge smart devices (such as smart watches, smart detectors, and other unmanned autonomous systems). High energy consumption and high-performance computation needs have become the key barriers for the continuous development of neural networks.

Spiking Neural Network (SNN) emerges as a front-runner in addressing the limitations of neural computing, proficiently leveraging machine learning algorithms in real-world applications.The ideas behind SNN, commonly viewed as the newest generation of neural networks, draw inspiration from the biological processes of neurons that adeptly handle distinct spatio-temporal occurrences (spikes) and the main neuronal models include the leaky integrate and fire (LIF) method [4]. LIF neuron represents a fundamental spiking neuron model characterized by its unique internal condition, termed membrane potential. Once this membrane potential surpasses a specific neuronal firing limit, it accumulates inputs across time, culminating in the generation of an output spike.They hold potential for enabling highly energy-efficient processing of sequential spatiotemporal information, particularly in deep layered networks. In SNN models, it's noted that the count of spikes, and consequently the computational demands, diminish considerably in the more profound layers.At present, most efficient approaches can be divided into two categories and we will compare them on chapter 4: i) the ANN-SNN conversion method and ii) SNNs developed from direct spike-based training.

1.2 Related Work

The method of converting ANN to SNN involves first training the ANN and then transforming it into an SNN via a distinct process, negating the need to train the SNN separately.Sengupta [5] attained a 91.55% accuracy rate on the CIFAR-10 dataset, experiencing a minor conversion accuracy loss of 0.15%. Subsequent to the ANN-SNN conversion, Rathi [6] refined the model through fine-tuning during the training process, securing a 92.22% accuracy on the same dataset while diminishing inference time. Stockl and Maass introduced a method known as few spike conversion (FS-Conversion) [7], realizing a 92.42% accuracy on the CIFAR-10 dataset. Since state-of-the-art (SOTA) ANN training methods are utilized, numerous prevailing ANN-SNN conversion methods achieve SNN SOTA classification efficacy. However, applying the ANN-SNN conversion method usually sets limits on the original ANN, leading to a reduction in performance. Moreover, a single inference in many ANN-SNN conversion methods typically demands hundreds to thousands of time steps, unintentionally contributing to extra time delays and higher energy consumption, opposing the aim.

Conversely, SNNs developed through direct spike-based training encounter several challenges. There are two main approaches to direct spike-based training: (i) non-optimization-based methods, which are typically unsupervised and use synapse-specific signals such as the timing of pre- and post-synaptic spikes, as exemplified by STDP, and (ii) optimization-based methods, which are generally supervised and use a universal target, like a loss function. An STDP-trained two-layer network with 6,400 output neurons has demonstrated a 95% classification rate on the MNIST dataset [8]. Additionally, Kheradpisheh [9] utilized STDP alongside a support vector machine (SVM) to obtain a 98.4% classification rate on the MNIST, although this result was significantly below what ANNs have achieved.

1.3 Thesis Objective

The search for spike-based backpropagation (BP) techniques has been motivated by the need to find a differentiable substitute for the spiking unit's activation function. This search has led to the development of a surrogate function in terms of the unit's membrane potential as outlined by Bohte [10] and Lee [11]. Additionally, Panda and Roy [12] have applied BP-driven supervised training to classifiers after initial layer-by-layer autoencoder-based feature extraction training. Lee [13] further advanced the field by merging layer-wise STDP-based unsupervised training with supervised spike-based BP, leading to greater robustness, enhanced generalization capabilities, and quicker convergence rates.

The current study is dedicated to an in-depth examination of these previous approaches and incorporates the methods introduced by Lee [14] for the training of profoundly deep SNNs using an end-to-end spike-based BP learning framework.

1.3.1 Contribution

- Development of a spike-based supervised gradient descent BP algorithm.
- Utilization and adaptation of key concepts from successful deep ANN models:
  - LeNet5 [1]
  - VGG [15]
  - ResNet ([16]

  to construct efficient deep convolutional SNN architectures.
- Application of the dropout technique [2] to enhance regularization during deep SNN training.
- Demonstration of the methodology's effectiveness in visual recognition tasks across various datasets:
  - Standard character and object datasets: MNIST, SVHN, CIFAR-10
  - Neuromorphic dataset: N-MNIST

- Comprehensive efforts to quantify the advantages of our algorithm by:
  - Comparing with ANN-SNN conversion techniques

1.4 Thesis Structure

The structure of this Thesis is as follows: Chapter 2 delves into the development of artificial intelligence, detailing each segment. In Chapter 3, we dissect and supply foundational knowledge on the core elements and structures of deep convolutional spiking neural networks (SNNs). Additionally, this chapter introduces the Dropout method and the spike-based gradient descent backpropagation (BP) learning algorithm.

Chapter 4 describes the experimental setup and analyzes the results, demonstrating the effectiveness of the proposed methods on the MNIST, N-MNIST, CIFAR-10, and SVHN datasets. Finally, Chapter 5 provides a summary of our discoveries and discusses directions for future research.

Chapter 2

Background

2.1 Evolution of AI Technologies

2.1.1 Introduction to Artificial Intelligence

As we see from Figure 2.1, Artificial Intelligence (AI) represents the overarching discipline of creating machines that can perform tasks requiring human-like intelligence. At its core, AI seeks to emulate the cognitive functions that humans associate with the human mind, such as ”learning” and ”problem solving.” Over the decades, AI has rapidly evolved, giving rise to various sub-disciplines and methodologies that are targeted at specific challenges. The journey of AI development is akin to peeling layers of an onion, with each layer representing a more specialized and refined subset of the broader domain.

2.1.2 Delving into Machine Learning

Machine Learning (ML) serves as the heart of AI, offering a paradigm shift from traditional programming. Instead of relying on explicit rules, ML models discern patterns by analyzing vast datasets. Central to ML is the principle of adaptability. As new data streams in, these models refine themselves, offering improved outcomes without the need for manual recalibration.

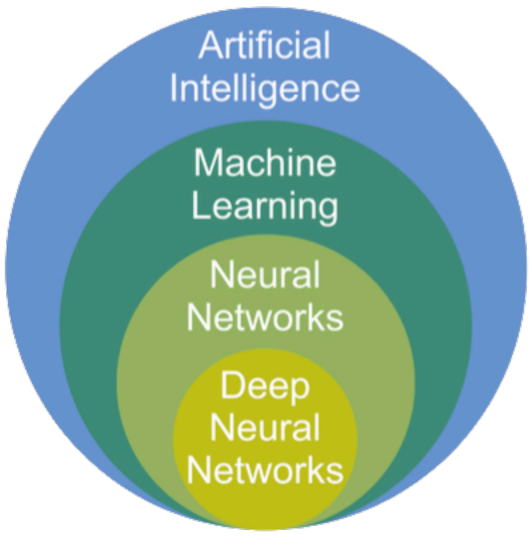

Figure 2.1: Depicting the various fields of artificial intelligence and where they fit in overall

2.1.3 Artificial Neural Networks – A Glimpse of Biological Complexity

Artificial Neural Networks (ANNs) act as the backbone of many ML applications. Composed of neurons and synapses, ANNs are designed to mimic the basic structure of our brain. These networks can be visualized as a series of layers: input, hidden, and output. Each neuron processes information, passing it forward and allowing the network to 'learn' from the data. The strength of the connections, or weights, adjust over time through a process called backpropagation, enabling the network to optimize its predictions.

2.1.4 Venturing into the Depths with Deep Neural Networks

Deep Neural Networks (DNNs) are essentially an enhancement of ANNs, distinguished by their multiple hidden layers. These layers allow DNNs to capture intricate relationships and patterns within data. As we traverse from the initial layers to the deeper ones, there's a progression from recognizing basic features to understanding more abstract and complex representations. DNNs have been pivotal in driving advancements in fields like image recognition and natural language processing.

2.1.5 Deep Learning: Advanced Neural Dynamics

Deep learning takes machine learning a step further by harnessing networks laden with interconnected layers, mimicking human brain functionalities. While they haven't achieved the full prowess of the human brain, their capability to digest massive datasets remains commendable. Notably, these networks employ multiple hidden layers, enhancing their predictive precision. One central process within them is ’forward propagation,’ where calculations traverse seamlessly through the network. In such architectures, the most discernible layers are the input, where data is ingested, and the output, where final predictions or classifications materialize.

2.1.6 The Spark of Spiking Neural Networks

Spiking Neural Networks (SNNs) represent a step closer to biological authenticity. Unlike ANNs and DNNs, which process information in a continuous manner, SNNs function based on discrete time spikes. These spikes are events where neurons send signals, much akin to the electrochemical processes in our brains. One significant model within SNNs is the Leaky Integrate-and-Fire (LIF) model. It captures the way a neuron accumulates input and, upon reaching a threshold, produces a spike and then resets. This spiking mechanism introduces temporal dynamics, making SNNs adept at processing time-series data and sensory inputs, akin to real neural systems.

2.2 Convolutional Neural Networks: Mastering Vision and Pattern Recognition

Convolutional Neural Networks (CNNs) stand as a pivotal subtype within the expanse of deep learning. Tailored primarily for image and video recognition, they have etched their significance by adeptly learning spatial hierarchies from input visuals. This prowess lets them not just parse pixels but fathom the essence of images. CNNs are woven with multiple layers, each serving a unique purpose. Yet, among these, three core layer types form the essence:

1. **Convolutional Layer**: At the heart of a CNN, this layer runs convolution operations, sculpting numerous petite feature maps from the primal image. It’s here that the network discerns rudimentary features, be it the soft curve of an edge or the intricate grain of a texture.

2. **Pooling Layer**: Strategically placed post the convolutional bouts, this layer's prerogative is dimensionality curtailment. Such spatial abridgment streamlines computations. A prevalent method here is 'max pooling', which gleans the paramount value from a chunk of the feature map, forging a condensed version thereof.

3. **Fully Connected Layer**: Anchoring the tail end of the architecture, this layer mirrors those in orthodox neural networks. With its neurons in total linkage with the antecedent layer, it orchestrates the network's final verdict, which could range from categorical classification to nuanced regression predictions.

Chapter 3

Spiking Neural Network:Study Design and Implementation

3.1 Components and Design Principles

3.1.1 Neuron Model

The Leaky Integrate and Fire (LIF) neuron model, which notably streamlines the action potential process while preserving its three fundamental characteristics (leaky, integrate, and fire) of the neuron membrane potential, can be articulated through the following formula, as illustrated in Figure 3.1:

$$\frac{dV_{\text{mem}}}{dt} = \frac{-V_{\text{mem}} + I(t)}{\tau_m} \tag{3.1}$$

where V_{mem} denotes the post-neuronal membrane potential, and τ_m represents the time constant associated with the decay of the membrane potential. The input current, $I(t)$, is described as the summation of pre-spikes, weighted for each time instance, as presented below.

$$I(t) = \sum_{i=1}^{n'} w_i \sum_{k} \theta_i(t - t_k) \tag{3.2}$$

where n' signifies the total count of pre-synaptic weights and w_i represents the synaptic strength between the i^{th} pre-neuron and its subsequent neuron.

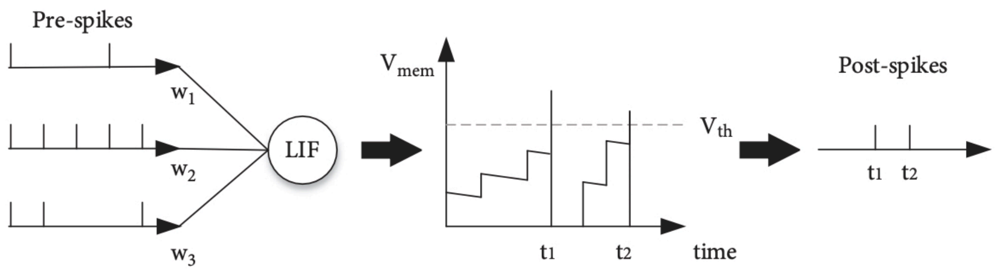

Figure 3.1: Illustration of the leaky, integrate, and fire (LIF) properties of LIF neurons: The postneuron's membrane potential starts gathering after receiving current from the preneuron. This potential diminishes in an exponential manner over time. Once this accumulated potential surpasses the firing threshold, the LIF neuron releases a spike in reverse and the membrane potential is reset.

The term $\theta_i(t - t_k)$, where θ_i,θ_0 represent pre- and post- spikes, denotes the spike occurrence from the i^{th} pre-neuron at time t_k and can be characterized using the Kronecker delta function:

$$\theta(t - t_k) = \begin{cases} 1 & \text{if } t = t_k \\ 0 & \text{otherwise} \end{cases}$$

where V_{k} represents the moment the kth spike takes place.

3.2 Deep Convolutional Spiking Neural Network

3.2.1 Modeling

In this study, we create a training approach for convolutional SNN models, which have three layers: an input layer, a middle hidden layer, and a top classification layer. The chance of a spike being generated is inversely correlated with pixel intensity in the input layer's encoding of pixel images as Poisson-distributed spike trains. Multiple convolutional (C) and spatial-pooling (P) layers make up the hidden layers, which are frequently placed alternately. These spatial-pooling (P) and convolutional (C) layers are the initial phases of the feature extractor. The fully-connected (FC) layers produce the final classification by combining the spikes from the feature extractor to create a one-dimensional vector input as illustrated in figure 3.2. While the spatial-pooling layers are defined a priori, the convolutional and fully connected layers have trainable parameters (i.e. synaptic weights). Weight kernels in the convolutional layers can encode feature representations of the input patterns at various hierarchical levels through the training process.

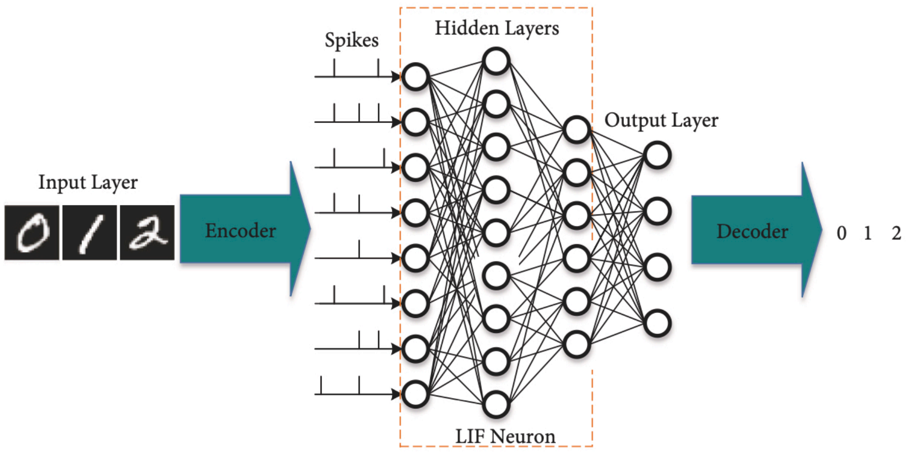

Figure 3.2: The encoder transforms input images into spike signals. The hidden layers, composed of LIF neurons, can adopt various network architectures like LeNet, VGG, or ResNet. Finally, the decoder translates the resulting spikes back into relevant classification outcomes.

We will analyze figure 3.3, which shows an operational simplified example of: i) convolutional and ii) pooling layer.

Firstly on left (A) part, we introduce Convolutional implementation, consisting of LIF neurons over three time steps (assuming 2-D input and 2-D weight kernel). During each time step, every neuron convolves its incoming spikes with the weight kernel to calculate its input current, subsequently integrating this into its membrane potential, denoted as V_{mem}. If V_{mem} surpasses the threshold voltage (V_{th}), the neuron generates a spike and V_{mem} is reset to 0. If not, V_{mem} is treated as residual in the succeeding time step while undergoing leakage in the current step.

Moreover, on the right (B) side of our illustration gets displayed the basic functionality of a pooling layer, which decreases the size from the preceding convolutional layer but maintains the spatial (topological) details.

There are two primary methods and both methods have been applied in SNNs, max-pooling [17] and average-pooling [18].We prefer average-pooling for its straightforwardness.

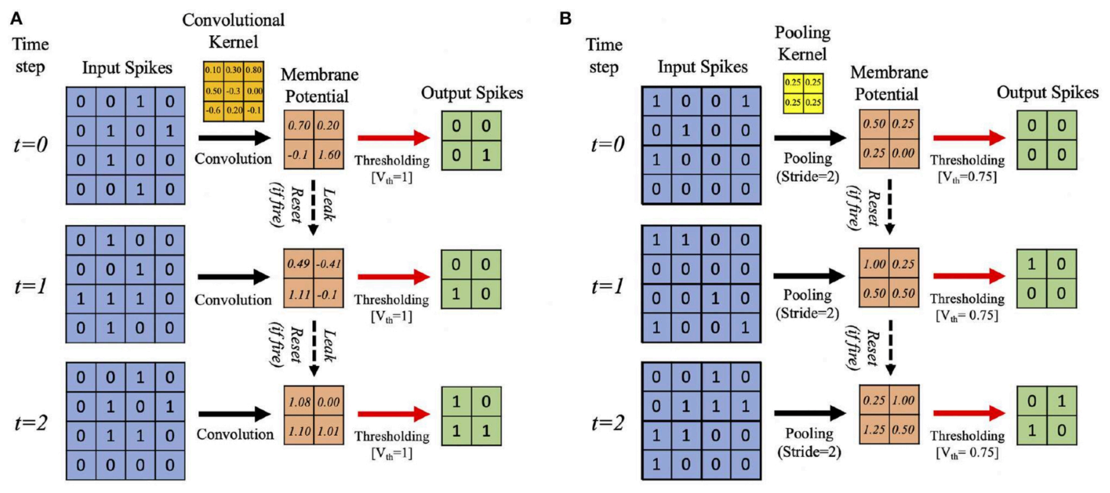

Figure 3.3: Simplified Operational Example

In the context of SNNs, there's an added thresholding step after the average is taken to yield output spikes. As an example, consider a static 2x2 kernel, with each part having a 0.25 weight. This kernel moves across a convolutional map without overlap, emitting an output spike at its matching position in the pooled map if the total of the weighted spikes from its four inputs surpasses a specified firing threshold (designated as 0.75 in our study). If not achieved, the membrane potential persists into the subsequent cycle. As long as we see Pooling implementation on figure 3.4, threshold for average-pooling must be judiciously determined to ensure continuous spike propagation during pooling. A threshold that's too low could lead to an excessive number of spikes, risking a loss in the spatial positioning of the feature extracted from the prior layer. Conversely, setting the threshold too high might hinder adequate spike propagation to subsequent layers.

3.2.2 VGG and residual SNNs

Deep learning models are essential for understanding complex input sequences because they allow for the layered learning of data representations. For the purpose of creating sophisticated Spiking Neural Network (SNN) designs, we have investigated well-known deep neural architectures, particularly VGG and ResNet.

VGG stands out for utilizing small (3x3) convolutional kernels consistently throughout its architecture. Convolution layers can be stacked effectively thanks to these small kernels while yet maintaining a manageable parameter volume for complex setups.

Utilizing ”Spiking VGG Blocks,” we designed deep convolutional SNNs in our work with more than five configurable layers. These comprise of convolutional layer sequences with regular usage of tiny (3x3) kernels. A Spiking VGG block, as shown in Figure 3.4[A], combines two consecutive convolutional segments, each of which is topped with a Leak Integrate and Fire (LIF) neuron layer. Synaptic interlinkages are shown by the convolutional portion, while activation components are shown by the LIF section.

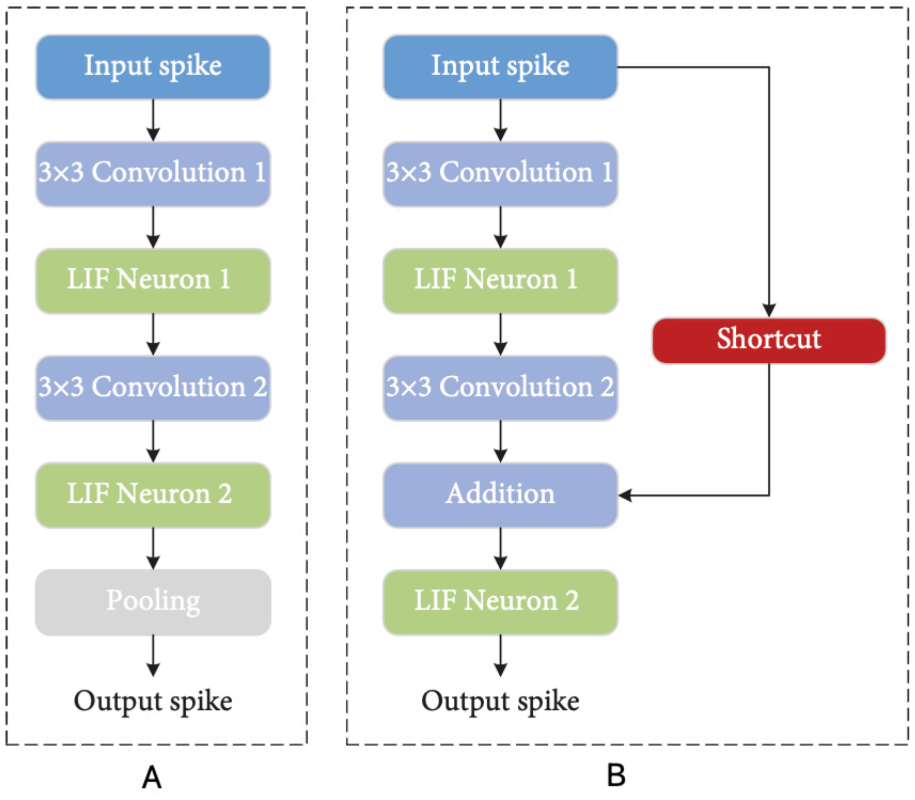

Figure 3.4: The foundational components of the outlined convolutional SNN designs are: (A) The Spiking VGG Block and (B) The Spiking ResNet Block.

ResNet advanced by including skip connections, which have now been demonstrated to be essential in enabling the successful training of increasingly deeper models. ResNet, crucially, provides a remedy for the decreasing training accuracy observed when adding more layers to traditional feedforward setups. By utilizing skip connections, we were able to create deep residual SNNs with 7 to 11 adjustable layers. The ”Spiking Residual Block” seen in Figure 3.4 is made up of separate residual and non-residual pathways. A LIF neuron layer connects two convolutional segments that are held in the non-residual track. The skip connection, on the other hand, mirrors the input-output feature maps or, in the case of a mismatch in the number of these maps, adopts 1x1 convolutional kernels.

The outputs from these two tracks combine at the LIF neuron layer's terminal level to create the spikes that are the end result.An "Spiking VGG Block" or "Spiking Residual Block" typically comes after a further average-pooling layer within the general feature extractor structure. Notably, certain Spiking Residual Blocks have convolutional and residual connections that end with a stride of 2, simulating the function of spatial-pooling segments. The final step in the extraction procedure involves channeling the features obtained from the final average-pooling layer into a fully linked layer that is then shown as a 1-D vector for further analysis.

3.3 Supervised Training

3.3.1 Spike-Based Gradient Descent Backpropagation Algorithm

Standard BP [19] in the ANN area served as the inspiration for the spike-based BP algorithm used in SNN. In conventional BP, the network parameters are iteratively changed to reduce the discrepancy between the network's final outputs and the target labels. The common BP technique accomplishes this by employing gradient descent to back-propagate the output error through the hidden layers.

The key distinction between ANNs and SNNs lies in the nature of their neuronal outputs. While an artificial neuron (like sigmoid, tanh, or ReLU) transmits using continuous values, a spiking neuron emits binary spikes sequentially over time. Within SNNs, networks take in sequences of spikes distributed over time and space as input. Consequently, the results produced by spiking neurons are temporal spike occurrences. This difference renders the conventional BP technique unsuitable for SNN training since it's ill-equipped to back-propagate gradients through an indifferential spike-producing function.

In our research, we introduce a near-accurate derivative for LIF neuron responses, enabling the utilization of gradient descent. We present a spike-focused BP approach designed to discern temporal sequences in spike patterns. This approach encompasses two steps: forward propagation, backward propagation and weight update, all of which are detailed in subsequent two subsections.

Forward Propagation

During the forward pass, input data is represented as spike patterns and introduced to the network to deduce the expected outputs. The transformation of pixel values into Poisson-based spike trains facilitates this. When these spike trains interact with synaptic weights, they generate an input current. This current gradually increases the membrane potential of subsequent neurons, as illustrated in Equation 3.1. When the membrane potential crosses a specific neuronal threshold, an output spike is emitted by the post-neuron and it then resets. If it doesn't reach the threshold, the potential diminishes exponentially over time. Neurons in every layer (barring the output layer) sequentially perform this mechanism, influenced by the weighted spikes from their preceding layer.

Over the course of time, the accumulated weighted sum of the pre-spike sequences (denoted as net) is given by:

$$net_j(t) = \sum_{i=1}^{n^{l-1}} w_{ij}^{l-1} x_i^{l-1}(t), \quad \text{where} \quad x_i^{l-1}(t) = \sum_t \sum_k \theta_i^{l-1}(t - t_k) \tag{3.3}$$

Here, $net_j(t)$ indicates the entire current influx assimilated into the membrane potential of the j^{th} post-neuron in layer l throughout the duration t. n^{l-1} is the total count of pre-neurons in layer $l-1$, while $x_i^{l-1}(t)$ symbolizes the aggregated spike train from the i^{th} pre-neuron across the time t. The combined post-spike sequences is denoted by $a_j(t)$ for the j^{th} post-neuron:

$$a_j(t) = \sum_t \sum_k \theta_j(t - t_k) \tag{3.4}$$

Here's a small summarization:

1. **Equivalence of Post and Pre-Spike Trains**: The total post-spike train, symbolized by $a_l(t)$, mirrors the total pre-spike train $x_l(t)$ for the succeeding neural layer. In simpler terms, the spike outputs from an initial layer transform into spike inputs for its adjacent layer.

2. **Threshold for the Final Layer**: The final layer, designated for classification, has its neuronal firing threshold set deliberately to an elevated value. This strategy ensures that this particular layer's neurons refrain from generating spikes. This elevated threshold ensures that the output activity is moderated and not easily triggered.

3. **Role of Pre-Spikes in the Final Layer**: Even though spiking is suppressed in the final layer, the neurons remain functionally significant. The incoming pre-spikes get weighted and contribute to the neuron's membrane potential, which undergoes a decay over a defined period.

4. **Quantifying the Output**: Post the execution of all designated time intervals, the aggregated membrane potential in the final layer's neurons is normalized by the overall time intervals, denoted by T. This methodology yields the output distribution, illustrated in the provided Equation 3.5. Essentially, this equation showcases the procedure to compute the mean membrane potential, which signifies the neural network's conclusive output.

$$\text{output} = \frac{V_{\text{mem}}^{L}(T)}{\text{number of timesteps}} \tag{3.5}$$

Backward Propagation & Weight Update

We now elaborate on the reverse computation process for introduced spike-based back-propagation method as it's cited [14]. Post the forward computation stage, the loss function quantifies the disparity between the intended labels and the network's forecasted outputs. This discrepancy's gradients are then determined at the terminal layer. Using the recursive chain methodology, these gradients are relayed backward to the starting layer via the intermediary layers, as expressed in the relation:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a_{LIF}} \frac{\partial a_{LIF}}{\partial \text{net}} \frac{\partial \text{net}}{\partial w} \tag{3.6}$$

The subsequent Equations, in conjunction with Figure 3.5, elucidate the method for deducing the partial derivatives of the terminal output error concerning weight aspects.

For each output neuron, its forecast discrepancy is determined by contrasting its output to the anticipated label obtained from the input spike sequences, as captured in:

$$e_j = \text{output}_j - \text{label}_j \tag{3.7}$$

The associated loss function, depicted as E, as given by:

$$E = \frac{1}{2}\sum_{j=1}^{n^L} e_j^2 \tag{3.8}$$

is the cumulative squared variance in predictions for all output neurons. To derive the terms $\frac{\partial E}{\partial a_{LIF}}$ and $\frac{\partial a_{LIF}}{\partial \text{net}}$ in the equation 3.6, a distinct activation function and an approach to differentiate this function within the context of an LIF neuron are necessitated.

In Spiking Neural Networks (SNN), the "activation function" signifies how the weighted sums of inputs before and after neuron firing relate over time. When processing information forward, there are different activation techniques for the concluding layer and other hidden layers. For the last layer, the "output" in Equation 3.9 is used as the neuronal activation, taking into account certain irregularities during spike times as disturbances. This leads to the formula:

$$\frac{\partial E}{\partial \text{output}} = \frac{\partial}{\partial \text{output}}\left(\frac{1}{2}(\text{output} - \text{label})^2\right) = \text{output} - \text{label} = e \tag{3.9}$$

In the back-propagation stage, we treat the inconsistencies of the membrane potential in the final layer neurons as noise. This perspective suggests that the overall membrane potential for a neuron can be seen as the sum of input currents it gets over a specified forward time duration(T). This relationship is represented as:

$$V_{\text{mem,j}}^L(T) \approx \sum_{i=1}^{n^L-1}(w_{ij}x_i(T)) = \text{net}^L(T) \tag{3.10}$$

For the final layer, the derivative of post-neuronal activation with respect to the net is given by:

$$\frac{\partial \text{output}}{\partial \text{net}} = \frac{\partial V_{\text{mem}}^L(T)/T}{\partial \text{net}} = \frac{\partial \text{net}(T)/T}{\partial \text{net}} = \frac{1}{T} \tag{3.11}$$

In the context of hidden layers, where outputs are represented as post-spike trains, the process of spike generation introduces discontinuities during firing. To address this, a surrogate derivative method is employed for LIF neuronal activation, denoted as $a'(\text{net})$, for back-propagation. This is primarily to approximate the term:

$$\frac{\partial a_{LIF}}{\partial \text{net}} \tag{3.12}$$

found in equation 3.6 for hidden layers alone.

To compute this surrogate derivative of LIF activation in relation to the net, several steps are taken. Firstly, the derivative of the "Integrate and Fire" (IF) neuron's activity is assessed.

Building upon this, and incorporating the IF neuron's derivative, a compensatory term is introduced to counter the dwindling influence of the membrane potential in LIF activation. The resulting derivative for LIF neuronal activation is then:

$$a'(\text{net}) = \text{Derivative of IF neuron} + \text{Compensatory term} \tag{3.13}$$

If a neuron in the hidden layer does not emit a spike, its derivative is set to zero.

The spike generation mechanism of an IF neuron operates as a strict threshold function, producing an output of either +1 or 0. When the accumulated input currents in the membrane potential surpass the designated firing threshold, the IF neuron produces a post-spike. It's important to note that the IF neuron doesn't have any leakage in its membrane potential. Consequently, the membrane potential of a post-neuron at any given time, t, is expressed by:

$$V_{\text{mem}}(t) \approx \sum_{i=1}^{n} w_i x_i(t) - V_{\text{th}} a(t) \tag{3.14}$$

Here, n symbolizes the count of pre-neurons, $x_i(t)$ represents the cumulative spike events from the i^{th} pre-neuron across time t, and $a_{\text{IF}}(t)$ illustrates the aggregated post-spike sequences over time t. This equation incorporates both the integration conduct and the fire/reset nature of the membrane potential dynamics.

Assuming a membrane potential, V_{mem}, of zero (a minor approximation), the IF neuron's activation can be given by $a_{\text{IF}}(t)$ and can be described as:

$$a_{\text{IF}}(t) \approx \frac{1}{V_{\text{th}}} \sum_{i=1}^{n} w_i x_i(t) = \frac{1}{V_{\text{th}}} \text{net}(t) \tag{3.15}$$

When differentiating this activation with respect to net, the derivative of the IF neuronal activation, as represented by the equation

$$\frac{\partial a_{IF}}{\partial \text{net}} \approx \frac{1}{V_{\text{th}}} \tag{3.16}$$

is approximately linear. This derivative indicates that the IF neuronal activation varies inversely with the threshold voltage, V_{th}.

Both the IF and LIF neuron models utilize the same spike generation mechanism, which is the strict threshold function.

Although the effective neural thresholds differ, the LIF (Leaky Integrate-and-Fire) neuron model demonstrates a unique leaky effect in its membrane potential. This implies that, in contrast to the IF (Integrate-and-Fire) neuron, a more significant input current is required by the LIF neuron to reach and surpass the neural threshold, leading to a post-spike. Thus, the threshold for the LIF neuron can be described as:

$$V_{th} + \epsilon$$

where ϵ is a positive factor representing the leakiness in membrane potential dynamics.

The LIF neuron's activation rate of change relative to its input, denoted by $\frac{\partial a_{LIF}}{\partial \text{net}}$, can be viewed as a strict threshold function, akin to the IF neuron. Its output is:

$$\frac{1}{V_{th} + \epsilon}$$

This suggests that both its firing threshold and inherent leaky property influence the LIF neuron's output. In contrast, the IF neuron's output relies solely on its firing threshold.

To evaluate ϵ, consider the ratio β between the cumulative membrane potentials of the IF and LIF neurons at a specific forward propagation time T (referenced in Figure 3.5). The term $V_{\text{total mem}}(t)$ depicts a theoretical total membrane potential integrated with the input current, disregarding any reset mechanisms.

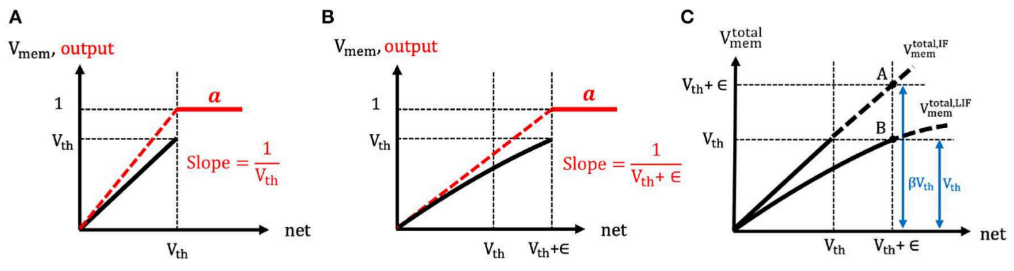

Figure 3.5: (A) IF (B) LIF and (C) The illustration of the estimation of the ratio (β) between the total membrane potential (Vtotal) of LIF and IF neurons.

Assuming equal total input currents (expressed as net(T)) for both IF and LIF neurons, the anticipated membrane potential for the LIF neuron would be lesser than the IF neuron, based on:

$$V_{\text{total,LIF mem}}(T) : V_{\text{total,IF mem}}(T) = 1 : \beta$$

where $\beta > 1$. Comparing membrane potentials of the two neuron types, the relationship between ϵ and β is:

$$V_{th} + \epsilon = \beta V_{th} \tag{3.17}$$

Here, $V_{th} + \epsilon$ signifies the IF neuron's overall membrane potential, and V_{th} relates to the LIF neuron's potential, assuming identical net input. The subsequent phase involves deducing β

by examining the spike output's evolution in tandem with the progression of the total membrane potential over time. Notably, the IF neuron's total input current and total membrane potential are analogous.

The absence of a leaky effect permits the derivation of the equation:

$$a_{IF}(t) \approx \frac{1}{V_{th}}\text{net}(t) \approx \frac{1}{V_{th}}v_{\text{total,IF mem}}(t) \tag{3.18}$$

By differentiating the above, we get:

$$\frac{\partial a_{IF}(t)}{\partial t} \approx \frac{1}{V_{th}}\frac{\partial v_{\text{total,IF mem}}(t)}{\partial t} \tag{3.19}$$

For the IF neuron scenario, the membrane potential evolution can be described by:

$$v_{\text{total,IF mem}}(t)\frac{\partial t}{\partial t} \approx \frac{1}{V_{th}}\frac{\partial a_{IF}(t)}{\partial t} \approx \frac{1}{V_{th}}v_{\text{total,LIF mem}}(t)\frac{\partial t}{\partial t} \tag{3.20}$$

After evaluating the above relations, the inverse of β is:

$$\frac{1}{\beta} = 1 + \frac{f(t)}{\text{rate}(t)}\frac{\partial t}{\partial t} \tag{3.21}$$

The initial term reflects the influence of mean input currents observed from the IF neuron's approximate derivation. Meanwhile, the subsequent term indicates the LIF neuron's leaky effect during forward propagation. Using this with the previous equation, the derivative for LIF neuronal activation can be deduced. The leaky effect during back-propagation is adjusted by the forward propagation time magnitude, leading to the approximate derivative for the LIF neuronal activation.

The activation in hidden layers combines both the direct estimation and an approximation of the IF neuron's derivative. This takes into account the leaky effect on the neuron's membrane potential as described by:

$$\frac{\partial u}{\partial net} = \frac{1}{V_{th}+c} \approx \frac{1}{V_{th}}\left(1+\frac{f'(t)}{k}\right)\left(1+\frac{1-e^{-\frac{t}{\tau_m}}}{\gamma k}\right) \tag{3.22}$$

Briefly, this equation provides a method to approximate a spike-based BP by considering:

- For the back-propagation phase, the hidden layer's membrane potentials are approximated as near-instantaneous activations relative to the total input current.
- The leaky effect for a LIF neuron in the hidden layers is estimated using $f(t)$, a function that represents the leak of output spikes over time. This function is continuous except at the moments of spikes.

- To approximate the LIF neuron's activation in hidden layers, a combination of straight-through estimation and the IF neuron's derivative approximation is used.

For training SNNs, the algorithm uses the above approximations with direct spike inputs in a spike-based BP method. The error gradient for the final layer, δ^L, is related to the difference in post-neuronal activation and can be derived as:

$$\delta^L = \frac{\partial E}{\partial output^L} \times \frac{\partial E}{\partial net^L} = \frac{e}{\tau} \tag{3.23}$$

Here, the change in net with respect to weight is straightforward and can be represented as:

$$\frac{\partial net}{\partial w^l} = x(t) \tag{3.24}$$

Combining these, the weight gradient, Δw^l, can be described by:

$$\Delta w^l = \frac{\partial E}{\partial w^l} = x(t) * (\delta^{l+1})^T \tag{3.25}$$

And finally, to update the weights, the equation is:

$$w_{\text{updated}} = w^l - \eta \Delta w^l \tag{3.26}$$

3.3.2 Dropout Method

Dropout, as emphasized by Srivastava [2], stands out as a key regularization technique for training deep artificial neural networks (ANNs). It operates by randomly deactivating certain units at each training stage with a pre-set probability (p), thereby helping to prevent overfitting. However, when adapting this technique for spiking neural networks (SNNs), alterations are necessary. Unlike ANNs, where training occurs through mini-batch iterations within each epoch, SNNs experience multiple forward propagations within a single iteration due to their reliance on time steps. Thus, for SNNs, it's crucial that the selection of active units remains unchanged during an iteration. Randomly reconnecting units at every time step within SNNs can dilute the effectiveness of dropout, making it almost insignificant by the time the forward propagation is complete and the error is back-propagated for parameter updates. To preserve the benefits of dropout, the same subset of units should be deactivated consistently over the entire duration of the time window for one iteration.

Our research on SNNs indicates that the ideal dropout probability (p) should be between 0.2 and 0.25, which is lower than the common 0.5 used in ANNs, due to the naturally more sparse activation in SNN forward propagations.

Chapter 4

Experiments and results

4.1 Experiments

4.1.1 Experimental configuration

For the empirical component of this thesis, we leveraged the computational capabilities of the NVIDIA T4 GPU, available through the Google Colab platform. The T4, a key member of NVIDIA's Tesla series, boasts a Turing architecture with 2,560 CUDA cores, 320 Turing Tensor Cores, and a substantial 16 GB of GDDR6 memory. This GPU's performance metrics, which peak at 8.1 teraflops for single-precision computations, made it an ideal choice for the resource-intensive nature of our experiments.

All specialized simulations for the SNN framework has been created using the PyTorch deep learning platform, compatible with Python 3.6.12 and PyTorch 1.1.0.

Our deep SNNs, based on the convolutional design, embed biologically inspired LIF neurons with a consistent firing threshold. These neurons are connected through modifiable synapses. Initially, the synaptic weights are configured using a Gaussian random distribution with a zero mean, and the standard deviation is computed as $\sqrt{\frac{k}{nl}}$ where nl signifies the count of incoming synapses. It's noteworthy that the initialization coefficient k is not constant; for instance, a value of 2 is used for non-residual architectures, while a value of 1 is designated for residual designs.

About training involves a comprehensive approach utilizing a spike-based BP technique, as we have already explained in the previous section. On static datasets, our models undergo 150 training epochs using mini-batch stochastic gradient descent BP, with modifications to its learning rate during the 70th, 100th, and 125th epochs.

Conversely, for neuromorphic dataset, the Adam [20] training approach is employed, altering its learning trajectory at the 40th, 80th, and 120th epochs. Additionally, insights encompassing the datasets adopted, network designs, input spike generation process, and the computation involved in training and inference phases are elaborated in the following segments.

4.1.2 Dataset Description

The MNIST (Modified National Institute of Standards and Technology) dataset is a standard for evaluating handwritten digit classification algorithms. It comprises 28x28 pixel grayscale images of handwritten digits from 0 to 9. The dataset is split into a training set with 60,000 examples and a test set containing 10,000 samples. Each image is represented as a 784-dimensional vector when flattened, and the pixel values range from 0 to 255.

Moving forward, the CIFAR-10 dataset, which stands for Canadian Institute For Advanced Research, contains 60,000 32x32 color images equally distributed across 10 classes, making it 6,000 images per class. The images come in RGB channels, translating each image into a 3,072-dimensional vector when flattened. The 10 classes include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is conventionally split with 50,000 images for training and 10,000 for testing.

SVHN, or Street View House Numbers, derives from Google Street View images. Unlike the simple and clean MNIST images, SVHN is colored, and the digits come from real-world scenarios, meaning they can be occluded, skewed, or in various other non-ideal conditions. The dataset comprises 32x32 pixel images, with 73,000 training and 26,000 testing samples. Its real-world nature makes it a more challenging dataset than MNIST.

N-MNIST is a neuromorphic variant of the MNIST dataset. Instead of the conventional 28x28 grayscale images of MNIST, N-MNIST offers 34x34 pixel images that encapsulate both ON and OFF spikes from a Dynamic Vision Sensor. These spikes represent changes in pixel intensity, much like the firing of biological neurons. The dataset contains 60,000 training samples and 10,000 testing samples.

Table 4.1: Benchmarking datasets.

Dataset	Image	Train sets	Test sets	Category
MNIST	28 x 28 gray	60000	10000	10
SVHN	32 x 32 color	73000	26000	10
CIFAR-10	32 x 32 x 3 color	50000	10000	10
NMNIST	32 x 34 x 2 ON/OFF	60000	10000	10

We can summerize technical characteristics on Table 4.1, which provides specifics on the datasets used in the experiments.

4.1.3 Spike Trigger Mechanism

For static image datasets like MNIST, SVHN, and CIFAR-10, each pixel's intensity is transformed into a sequence of spike events with matching firing rates. Specifically, during every time interval, the intensity of a pixel is adjacented with a uniformly random value between 0 and 1. If the pixel's intensity surpasses this random value during a particular interval, a spike is produced. This method of encoding through spikes is employed to provide the neural network with input spikes throughout both the training and inferencing stages. When dealing with datasets containing colored images, a horizontal flip is applied as a preprocessing step before producing spikes. Subsequently, pixel values are standardized to have a mean of zero and a standard deviation of one. Following this, pixel intensities are adjusted to lie within the range of [-1,1], covering the entire spectrum of pixel values. Once normalized, these pixel intensities get translated into spikes following a Poisson distribution, resulting in bipolar spikes.

As for the neuromorphic version of the dataset, N-MNIST, the network is directly trained and tested using the raw, unmodified spike streams within the temporal domain.

4.1.4 Network Structures

For the MNIST and N-MNIST datasets, we adopt a structure akin to the LeNet5 architecture. This includes two sequential pairs of convolutional and spatial-pooling layers, followed by a duo of fully-connected layers. The details such as the type of layer, kernel size, number of output feature maps, and stride for the MNIST model are specified in Table 4.2. It is pertinent to mention that the kernel size referenced here pertains to 3-D convolution, with the first dimension corresponding to input feature maps and the remaining dimensions to the size of the convolutional kernels.

For the SVHN and CIFAR-10 datasets, our approach is to use more complex models comprising 7 to 11 trainable layers, which include convolutional, spatial-pooling, and fully-connected layers. It is notable that networks exceeding five trainable layers are equipped with smaller convolutional kernels of size 3 x 3. The elaborate deep convolutional SNN configuration featuring 3 x 3 convolutional kernels, devoid of residual links, is termed as ”VGG SNN,” whereas the structure that incorporates skip (or residual) connections is referred to as ”Residual SNN.” In this latter architecture, certain convolutional layers adopt kernels that stride in both the x and y dimensions, effectively combining the role of spatial-pooling.

We provide a detailed examination of these deep convolutional SNN architectures in the subsequent tables.
Table 4.2 presents the LeNet5 Model specifics employed for the MNIST and N-MNIST datasets.

Table 4.2: LeNet5 network architecture

Layer	Kernel Size	Channel	Stride
Convolutional	1x5x5	20	1
Average-pooling	2x2	20	2
Convolutional	20x5x5	50	1
Average-pooling	2x2	50	2
Fully-connected		200	
Output		10	

Moreover, on table 4.3 is illustrated VGG7 and ResNet7 which used for SVHN dataset.

Table 4.3: VGG7 and ResNet7 network architectures.

VGG7				ResNet7			
Layer type	Kernel size	Channel	Stride	Layer type	Kernel size	Channel	Stride
Convolution	3x3x3	64	1	Convolution	3x3x3	64	1
Convolution	64x3x3	64	2	Average-pooling	2x2	64	2
Average-pooling	2x2	64	2	Convolution	64x3x3	128	1
Convolution	64x3x3	128	1	Convolution	128x3x3	128	2
Convolution	128x3x3	128	2	Skip convolution	64x1x1	128	2
Convolution	128x3x3	128	2	Convolution	128x3x3	256	1
Average-pooling	2x2	128	2	Convolution	256x3x3	256	2
				Skip convolution	128x1x1	256	2
Fully-connected		1024		Fully-connected		1024	
Output		10		Output		10	

Lastly, we have table 4.1.4, which demonstrate ResNet11, ResNet9 and VGG9 architectures for CIFAR-10 dataset.

Table 4.4: VGG9, ResNet9 and ResNet11 network architectures.

VGG9				ResNet9				ResNet11			
Layer type	Kernel size	Channel	Stride	Layer type	Kernel size	Channel	Stride	Layer type	Kernel size	Channel	Stride
Convolution	3x3x3	64	1	Convolution	3x3x3	64	1	Convolution	3x3x3	64	1
Convolution	64x3x3	64	1	Average-pooling	2x2	64	2	Average-pooling	2x2	64	2
Average-pooling	2x2	64	2	Convolution	64x3x3	128	1	Convolution	64x3x3	128	1
Convolution	64x3x3	128	1	Convolution	128x3x3	128	1	Convolution	128x3x3	128	1
Convolution	128x3x3	128	1	Skip convolution	64x1x1	128	1	Skip convolution	64x1x1	128	1
Average-pooling	2x2	128	2	Convolution	128x3x3	256	1	Convolution	128x3x3	256	1
Convolution	128x3x3	256	1	Convolution	256x3x3	256	2	Convolution	256x3x3	256	2
Convolution	256x3x3	256	1	Skip connection	128x1x1	256	2	Skip convolution	128x1x1	256	2
Convolution	256x3x3	256	1	Convolution	256x3x3	512	1	Convolution	256x3x3	512	1
Average-pooling	2x2	256	2	Convolution	512x3x3	512	2	Convolution	512x3x3	512	1
				Skip convolution	256x1x1	512	2	Convolution	512x3x3	512	2
								Skip convolution	512x1x1	512	2
Fully-connected		1024		Fully-connected		1024		Fully-connected		1024	
Output		10		Output		10		Output		10	

4.1.5 Time-Steps

The length of a spike train, measured in time-steps, is a critical parameter for Spiking Neural Networks (SNNs). For example, a spike train spanning 100 time-steps could contain approximately 50 random spikes if the corresponding pixel intensity is halfway between 0 and 1. If the number of time-steps is too limited, the SNN may not glean enough information, which can impede its training or inference performance. On the other hand, an excessive number of time-steps may undermine the stochastic essence of SNNs, resulting in higher latency and energy consumption. This could diminish the energy efficiency that SNNs have over conventional Artificial Neural Networks (ANNs). To strike an optimal balance, we ran a series of experiments with a range of time-steps to identify the most suitable quantity for both training and inference phases.

Our training of VGG9 models on the CIFAR-10 dataset explored time-steps from 10 to 120, as documented. We found that a mere 10 time-steps rendered the network ineffective due to the paucity of spike data, evidenced by a lack of spikes in the output. This inefficacy arises because the preliminary charge buildup in the LIF neuron falls short of triggering spikes in the succeeding layers.

As a result, input spikes fail to propagate to the output neurons, leading to a uniform zero output distribution. Consequently, the gradients are zeroed out, thwarting any network learning. Nonetheless, within a span of 35–50 time-steps, the network's learning becomes efficient, eventually stabilizing at an acceptable level of performance. Beyond 70 time-steps, the accuracy starts to plateau.

Reaching around 100 time-steps, there is no significant enhancement in training results. This concurs with Sarwar [21]'s assertions that 8-bit inputs and activations are adequate for peak performance in standard image classification tasks. In theory, 128 time-steps would be perfect for representing 8-bit inputs through bipolar spikes.

However, our experimental outcomes indicate that 100 time-steps suffice, with further increases yielding little to no additional advantage. This pattern held true in other SNN configurations like VGG7, ResNet7, ResNet9, and ResNet11 when assessed on both the SVHN and CIFAR-10 datasets. Thus, we adopted 100 time-steps as the optimal count for our training endeavors. For the MNIST dataset, we chose 50 time-steps in light of its 4-bit precision requirement, as highlighted by Sarwar [21].

4.2 Results

In this section, we evaluate the classification efficacy and efficiency of our proposed spike-based training approach for deep convolutional SNNs, contrasting it with the performance of SNNs transformed using the ANN-SNN conversion method.

Existing literature primarily reports classification performances for SNNs on the MNIST and CIFAR-10 datasets. Two dominant SNN training techniques have emerged: unsupervised learning based on ”Spike Time Dependent Plasticity (STDP)” and supervised learning using ”spike-based backpropagation”.Some research even attempted to merge these methods, aiming to harness their combined strengths.

However, such integrative techniques haven’t been able to train deep SNNs effectively or match the performance of ANNs. This led to the exploration of ANN-SNN conversion methods, with the most successful inference outcomes for the CIFAR-10 dataset stemming from these conversion techniques. Our results, alongside others, are displayed in Table 4.5. Notably, our work was pretty close with others in MNIST inference accuracy using a LeNet5-based structure. For the CIFAR-10 dataset, our results also could compete Wu’s [22] approach around and reached 90%.

Table 4.5: The classification accuracy of SNN on the CIFAR-10 dataset.

Author	**Method**	**Accuracy (%)**
Hunsberger and Eliasmith [23]	Conversion	82.95
Esser et al. [22]	Conversion	89.32
Rueckauer et al. [17]	Conversion	88.82
Sengupta et al. [5]	Conversion	91.55
Rathi et al. [6]	Conversion + STDB	92.22
Stöckl and Maass [7]	FS-conversion	92.42
Wu et al. [24]	Spike-based BP	90.53
Lee et al. [14]	Spike-based BP	90.95
Fang et al. [25]	Surrogate gradient	93.50
Saeed Kheradpisheh and Maryam [26]	Proxy	93.11
Wu et al. [27]	Spike-based HP	91.08
Liu et al. [28]	SABP	91.03
Our Work	Spike-based BP	90.95

Table 4.6: The classification accuracy of SNN on the MNIST dataset.

Author	**Method**	**Accuracy (%)**
Diehl and Cook [29]	STDP	95.00
Kheradpisheh et al. [30]	STDP	98.40
Hunsberger and Eliasmith [31]	Conversion	98.37
Diehl et al. [32]	Conversion	99.10
Rueckauer et al. [33]	Conversion	99.44
Lee et al. [34]	Spike-based BP	99.31
Jin et al. [35]	HM2-BP	99.49
Wu et al. [36]	Spike-based BP	99.42
Lee et al. [37]	STDP + spike-based BP	99.28
Lee et al. [38]	Spike-based BP	99.59
Fang et al. [39]	Surrogate gradient	99.72
Wu et al. [40]	Spike-based HP	99.50
Liu et al. [28]	SABP	99.62
Our Work	Spike-based BP	99.50

After initializing network weights, we employed a spike-based backpropagation algorithm to train SNNs, using Poisson-generated spike train inputs. Our MNIST evaluation yielded a stellar 99.50% accuracy, really good accuracy by other SNN schemes and on par with ANN-SNN conversion methods.

About N-MNIST performance, as we can see from 4.8 we achieve around 99.44% among three other approaches.

Table 4.7: The classification accuracy of SNN on the SVHN dataset.

Author	**Method**	**Accuracy (%)**
Lee et al. [38]	Spike-based BP	96.21
Our Work	Spike-based BP	96.06

Our methods achieved an inference accuracy of around 96.06% for both non-residual and residual SNNs on the SVHN dataset, a dataset seldom evaluated in existing literature.

Table 4.8: The classification accuracy of SNN on the N-MNIST dataset.

Author	**Method**	**Accuracy (%)**
Hunsberger and Eliasmith [23]	Conversion	-
Esser et al. [22]	Conversion	-
Rueckauer et al. [17]	Conversion	-
Sengupta et al. [5]	Conversion	-
Rathi et al. [6]	Conversion + STDB	-
Stöckl and Maass [7]	FS-conversion	-
Wu et al. [24]	Spike-based BP	98.78
Lee et al. [14]	Spike-based BP	99.09
Fang et al. [25]	Surrogate gradient	-
Saeed Kheradpisheh and Maryam [26]	Proxy	-
Wu et al. [27]	Spike-based HP	99.53
Liu et al. [28]	SABP	-
Our Work	Spike-based BP	99.44

For the CIFAR-10 dataset, we designed three unique networks. In the VGG9 network, the ANN-SNN conversion resulted in performance nearly identical to the original ANN, while our method achieved 90.43% accuracy.

Table 4.9: Comparison of different model performances on CIFAR-10 dataset.

Dataset	**Model**	**ANN**	**ANN-SNN et al. [5]**	**SNN et al. [14]**	**SNN (Our work)**
CIFAR-10	VGG9	91.98%	92.01%	90.45%	90.45%
CIFAR-10	ResNet9	91.85%	91.59%	90.35%	90.26%
CIFAR-10	ResNet11	91.87%	91.65%	90.95%	90.86%

For the ResNet9, conversion methods stayed within 1% of the original ANN, whereas our approach was about 1.5% off. In the ResNet11 comparison, our SNN training showed an approximately 0.5% improvement over the ResNet9. In summary, our training method offers inference accuracy for both ResNet and VGG networks that rivals both traditional ANNs and ANN-SNN conversion approaches.

Chapter 5

Conclusions

5.1 Summary and conclusion

Addressing the challenge of CNNs being energy-intensive for edge smart devices, this paper introduces a versatile brain-inspired SNN design alongside an efficient spike-based backpropagation method. We tailor network structures based on dataset specifics. Our experiments confirm the method's robust performance in deep SNNs. Energy analyses highlight SNN's energy efficiency, making it more apt for edge device deployment compared to CNNs. Presently, our SNN training approach closely matches top accuracies on MNIST, N-MNIST and CIFAR-10, and notably excels on the SVHN dataset. Utilizing appropriate neuromorphic hardware, our method promises substantial reductions in computation and energy for classification tasks.

5.2 Future work

Future endeavors will focus on minimizing SNN classification delays, broadening dataset experimentation and optimizing configurations for reduced test losses. Finally, it would be great idea if we see implementation for Hardware applications and try to maximise accuracy results.

Bibliography

[1] Bengio Y. LeCun Y., Bottou L. and Haffner P. Gradient-based learning applied to document recognition. *IEEE 86*, page 2278–2324, 1998.

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

[3] Ioffe S. and Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv, 2015. 1502.03167.

[4] J. N. Kok S. M. Bohte and H. La Poutre´. Error-back- propagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002.

[5] R. Wang C. Liu A. Sengupta, Y. Ye and K. Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13(95):95, 2019.

[6] P. Panda N. Rathi, G. Srinivasan and K. Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. arXiv, 2020. 2005.01807.

[7] C. Stockl and W. Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, 3(3):230–238, 2021.

[8] P. U. Diehl and M. Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9(99), 2015.

[9] M. Ganjtabesh S. R. Kheradpisheh and T. Masquelier. Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing*, 205:382–392, 2016.

[10] Kok J. N. Bohte S. M. and La Poutre H. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing 48*, pages 17–37, 2002. doi: 10.1016/S0925-2312(01)00658-0.

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

[12] Panda P. and Roy K. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. In *2016 International Joint Conference on Neural Networks (IJCNN) (Vancouver, BC: IEEE)*, 2016.

[13] Srinivasan G. Lee C., Panda P. and K. Roy. Training deep spiking convolutional neural networks with stdp based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.*, 12(435), 2018. doi: 10.3389/fnins.2018.00435.

[14] P. Panda G. Srinivasan C. Lee, S. S. Sarwar and K. Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14:119, 2020.

[15] Simonyan K. and Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv, 2014. 1409.1556.

[16] Ren S. He K., Zhang X. and Sun J. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (Las Vegas, NV)*, 2016.

[17] Hu Y. Pfeiffer M. Rueckauer B., Lungu I.-A. and Liu S.-C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:682, 2017. doi: 10.3389/fnins.2017.

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

[19] Hinton E. Rumelhart D. and Williams R. J. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[20] D. P. Kingma and Ba J. Adam: A method for stochastic optimization. arXiv, 2014. 1412.6980.

[21] Srinivasan G. Han B. Wijesinghe P. Jaiswal A. Sarwar S., Srinivasan S. and Panda P. Energy efficient neural computing: a study of crosslayer approximations. *IEEE J. Emerg. Sel. Top. Circuits Syst.*, (8):796–809, 2018.

[22] P. A. Merolla S. K. Esser and J. V. Arthur. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, (45):11441, 2016.

[23] E. Hunsberger and C. Eliasmith. Spiking deep networks with lif neurons. arXiv, 2015. 1510.08829.

[24] G. Li J. Zhu Y. Xie Y. Wu, L. Deng and L. Shi. Direct training for spiking neural networks: faster, larger, better,. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[25] Y. Chen T. Masquelier T. Huang W. Fang, Z. Yu and Y. Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Pro- ceedings of the IEEE/CVF International Conference on Com- puter Vision (ICCV)*, 2021.

[26] R. Saeed Kheradpisheh and M. T. Maryam. Spiking neural networks trained via proxy. arXiv, 2021. 2109.13208.

[27] R. Zhao Y. Wu and J. Zhu. Brain-inspired global-local learning incorporated with neuromorphic computing. *Nature Communications*, (13):13, 2022.

[28] R. Liu K. Cao R. Wang Y. Wang W. Zhao Y. Liu, M. Tian and Y. Zhou. Spike-based approximate backpropagation algorithm of brain-inspired deep snn for sonar target classification. 2021.

[29] P. U. Diehl and M. Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, (9):99, 2015.

[30] S. R. Kheradpisheh, M. Ganjtabesh, and T. Masquelier. Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing*, (205):382–392, 2016.

[31] E. Hunsberger and C. Eliasmith. Spiking deep networks with lif neurons. *arXiv*, page abs/1510.08829, 2015.

[32] 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. *Proceedings of the IEEE International Conference on Rebooting Computing (ICRC)*, page 1–8, 2016.

[33] I.-A. Rueckauer, Y. Hu Lungu, M. Pfeiffer, and S.-C. Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, (11):682, 2017.

[34] T. Delbruck J. H. Lee and M. Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, (508):508, 2016.

[35] Y. Jin, P. Li, and W. Zhang. Hybrid macro/micro level backpropagation for training deep spiking neural networks. *Proceedings of the Advances in Neural Information Processing Systems*, page 7005–7015, 2019.

[36] Y. Wu, L. Deng, J. Zhu, and L. Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, (12):331, 2018.

[37] G. Srinivasan C. Lee, P. Panda and K. Roy. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in Neuroscience*, (12):435, 2018.

[38] P. Panda G. Srinivasan C. Lee, S. S. Sarwar and K. Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, (14):119, 2020.

[39] Y. Chen T. Masquelier T. Huang W. Fang, Z. Yu and Y. Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Pro- ceedings of the IEEE/CVF International Conference on Com- puter Vision (ICCV)*, 2021.

[40] R. Zhao Y. Wu and J. Zhu. Brain-inspired global-local learning incorporated with neuromorphic computing. In *Nature Communications*, 2022.