UNIVERSITÀ DEGLI STUDI DI PADOVA

Department of Physics and Astronomy "Galileo Galilei"

Master Thesis in Physics of Data

Exploration of Deep Spiking Neural Network Architectures for Reward-Modulated STDP Learning

Supervisor
Professor Michele Rossi
UNIVERSITÀ DEGLI STUDI DI PADOVA

Co-supervisor
Doctor Riccardo Mazzieri
Università degli Studi di Padova
Doctor Eleonora Cicciarella
Università degli Studi di Padova

Master Candidate
Aidin Attar

Academic Year
2023/2024

This work is dedicated to my family and loved ones, whose unwavering support and encouragement have always been my greatest source of strength. To my parents, who taught me the value of perseverance and curiosity, and to my friends, who have been by my side through every challenge, I am forever grateful.

> "The most beautiful thing we can experience is the mysterious. It is the source of all true art and science."
>
> – Albert Einstein

Abstract

Spiking neural networks (SNNs) are a class of neural networks that more closely mimic the behavior of biological neurons compared to traditional artificial neural networks (ANNs). Unlike ANNs, which rely on continuous activations, SNNs process information through discrete spikes, where the precise timing of these spikes encodes information. This event-driven nature makes SNNs highly efficient in terms of energy consumption, making them an attractive option for neuromorphic computing. However, the non-differentiable nature of spike-based activation functions presents challenges for training, especially with gradient-based methods like backpropagation.

Reward-modulated spike-timing-dependent plasticity (STDP) offers a biologically plausible learning mechanism for SNNs, incorporating reward signals to drive synaptic changes. While classical STDP relies solely on the timing of pre- and postsynaptic spikes, the addition of a reward signal allows for more sophisticated learning paradigms, particularly in tasks requiring temporal precision. Despite this, deep SNN architectures, while promising for neuromorphic computing due to their event-driven nature, face challenges in matching the performance of traditional deep neural networks (DNNs). Architectural innovations such as inception-like blocks, skip connections, and population coding have been explored to improve the efficiency and scalability of deep SNNs. These modifications aim to improve network adaptability and learning capacity in complex, real-world scenarios. Furthermore, Liquid State Machines (LSMs) offer an additional approach for capturing temporal dynamics in SNNs, leveraging their liquid-like behavior for efficient processing of time-varying inputs. This work examines recent advances in reward-modulated STDP and deep SNN architectures, focusing on their potential to bridge the performance gap between SNNs and traditional DNNs, while maintaining the energy efficiency and biological relevance that make SNNs attractive for neuromorphic systems.

Contents

Abstract v

List of figures ix

List of tables xi

Listing of acronyms xiii

1 Introduction 1

2 Background and related work 5
2.1 Overview of Spiking Neural Networks 6
2.1.1 Biological Foundations and Inspiration 7
2.1.2 Spiking Neural Networks: Computational Model 17
2.1.3 Architectures of SNNs 26
2.1.4 Applications of SNNs 29
2.2 Related Works 31
2.2.1 Learning Mechanisms in Spiking Neural Networks 31
2.2.2 Liquid State Machines and Computational Models 32
2.2.3 Neuromorphic Hardware Implementations 33
2.2.4 Spiking Neural Networks in Real-World Applications 34
2.2.5 Comparison with Traditional Approaches 34

3 Methodology 35
3.1 Overview 35
3.2 Dataset and preprocessing 37
3.3 Network architecture design 40
3.3.1 Baseline Model A: Shallow Convolutional SNN 40
3.3.2 Baseline Model B: Deep Convolutional SNN 42
3.3.3 Extended DCSNN with Additional Convolutional Layer 42
3.3.4 Population Coding-Based Spiking Neural Network 43
3.3.5 Inception-Like Spiking Neural Network 45
3.3.6 Residual Spiking Neural Network 48
3.3.7 Surrogate Gradient-Based Spiking Neural Network 50
3.3.8 Liquid State Machine Model 51

3.4 Training Techniques and Parameter Settings . . . 53
3.4.1 STDP and R-STDP . . . 54
3.4.2 Comparison of R-STDP and STDP . . . 56
3.4.3 Surrogate Gradient Method . . . 57
3.4.4 Training techniques for LSM . . . 59
3.5 Overfitting Avoidance Mechanisms . . . 59
3.5.1 Adaptive Learning Rate . . . 60
3.5.2 Dropout . . . 61
3.5.3 Training Schedulers . . . 62
3.5.4 Data Augmentation . . . 62
3.6 Hardware and Computational Environment . . . 63

4 Experiments and Results **67**
4.1 Experimental Setup . . . 67
4.2 Baseline Models . . . 69
4.2.1 Experiment 1: Implementing a shallow CSNN . . . 69
4.2.2 Experiment 2: R-STDP in DCSNN . . . 71
4.3 Extended SNN Architectures . . . 73
4.4 Advanced SNN Architectures . . . 75
4.4.1 Population Coding SNN . . . 75
4.4.2 Inception-like SNN . . . 76
4.4.3 ResNet-like SNN . . . 77
4.4.4 Liquid State Machine . . . 78
4.5 Surrogate Gradient-Based Training . . . 80
4.5.1 Comparison of STDP and Backpropagation . . . 80
4.6 Summary of the results . . . 81

5 Discussion **83**
5.1 R-STDP in Hierarchical SNNs . . . 84
5.2 Integration of STDP and R-STDP . . . 85
5.3 Signal Degradation in Deep Architectures . . . 86
5.4 Role of sophisticated architectures . . . 88
5.5 On the catastrophic forgetting issue . . . 89
5.6 Challenges in Hyperparameter Selection . . . 90
5.7 Reservoir Computation and Versatility . . . 91

6 Conclusion **93**

References **97**

Acknowledgments **101**

Listing of figures

2.1 Schematic representation of a single neuron . . . 8
2.2 On-centre- and Off-centre-surround bipolar cells' response to light stimuli. . . 9
2.3 Spike timing-dependent plasticity . . . 13
2.4 Reward modulated STDP . . . 14
2.5 LIF neuron model illustration. . . . 18
2.6 STDP and R-STDP . . . 23
2.7 Illustration of the surrogate gradient method . . . 24
2.8 Schematic of a Liquid State Machine . . . 28
2.9 TrueNorth Architecture Overview . . . 30

3.1 Logo of the main libraries used in the development of the models used. . . . 36
3.2 Samples from the datasets used. . . . 37
3.3 Intensity to Latency encoding scheme . . . 38
3.4 Visualization of Micro-Doppler and Range-Doppler Spectrograms for Hand Gesture Recognition. . . . 39
3.5 Schematic representation of the architecture of baseline A . . . 41
3.6 Schematic representation of the architecture of baseline B . . . 43
3.7 Inception Block . . . 47
3.8 Schematic representation of a residual connection . . . 49
3.9 Liquid State Machine schematic . . . 52
3.10 Augmentation examples . . . 63

4.1 Feature evolution and hit rate . . . 70
4.2 Comparison of recognition performance in SNNs . . . 71
4.3 Reward modulated STDP . . . 72
4.4 Filters Learned by the SNN Layers . . . 74
4.5 Training Curves for SNN models . . . 78
4.6 Impact of partitioning and size . . . 79
4.7 Results of the comparison between STDP and BP . . . 81
4.8 Training Curves for SNN models . . . 82

5.1 Critical brain hypothesis phases . . . 87
5.2 Top-5 Error Rates of Deep CNNs . . . 88

Listing of tables

3.1 Shallow CNN architecture 40
3.2 DCNN architecture 44
3.3 Deep model architecture 45
3.4 Population coding network architecture 46
3.5 Inception-like architecture 48
3.6 ResNet-like network architecture 49
3.7 Surrogate gradient-based network architecture 51
3.8 Software Environment Details 64
3.9 Key Hardware Specifications of the Training System 64
3.10 Parameters of STDP models. 65

4.1 Performance Metrics for Different SNN Architectures on MNIST and CIFAR-10 Datasets 82

Listing of acronyms

Ach ..... Acetylcholine

AI ..... Artificial intelligence

ANN ..... Artificial Neural Network

BP ..... Backpropagation

CNN ..... Convolutional Neural Network

CSNN ..... Convolutional Spiking Neural Network

DNN ..... Deep Neural Network

DCSNN ..... Deep Convolutional Spiking Neural Network

DoG ..... Difference of Gaussians

DVS ..... Dynamic Vision Sensors

ESN ..... Echo State Network

IF ..... Integrate-and-Fire

IoT ..... Internet of Things

IQR ..... Interquartile range

KNN ..... K-nearest neighbors

LIF ..... Leaky Integrate-and-Fire

LSM ..... Liquid State Machine

LTD ..... Long term depression

LTP ..... Long term potentiation

MAC ..... Multiply-Accumulate

ML ..... Machine Learning

MLP ..... Multilayer Perceptron

MSE ..... Mean Square Error

NA ..... Noradrenaline

NN ..... Neural Network

ResNet ..... Residual Network

RNN ..... Recurrent Neural Network

RSNN ..... Recurrent SNN

R-STDP ..... Reward-modulated STDP

SGD ..... Stochastic Gradient Descent

SNN ..... Spiking Neural Network

STDP ..... Spike-timing-dependent plasticity

SVM ..... Support Vector Machine

1 Introduction

Artificial intelligence (AI) and neural networks have significantly shaped the landscape of modern technology, with deep learning models now ubiquitous in fields ranging from computer vision to natural language processing. These models, however, come at a considerable computational cost. The impressive performance of deep neural networks (DNNs) is often accompanied by an exponential increase in the number of parameters, computational complexity, and energy consumption. This presents challenges, particularly as we push towards more sophisticated AI systems that must operate under stringent energy and hardware constraints.

Spiking neural networks (SNNs) have garnered attention as a potential solution to some of these challenges. SNNs are inspired by the functioning of biological brains, where neurons communicate using discrete spikes rather than continuous signals. The brain, despite its remarkable computational capabilities, operates with extreme energy efficiency, consuming only about 20W of power. This efficiency is in part due to the sparse and asynchronous nature of the spiking activity in biological neurons, which contrasts with the dense and synchronous activations seen in traditional artificial neural networks [1].

The event-driven nature of SNNs, where neurons only fire when necessary, provides an inherent advantage in terms of energy consumption. This makes SNNs particularly appealing for applications that require real-time processing and low-power operation, such as robotics, wearable devices, and edge computing. Neuromorphic hardware, designed to mimic the architecture of the brain, has been developed specifically to harness the efficiency of SNNs, with companies such as Intel and IBM leading the way in producing specialized chips that support

event-driven computation, based on discrete spike events [2].

However, the promise of energy efficiency is not the only reason for the growing interest in SNNs. From a computational perspective, SNNs offer a fundamentally different way of processing information. Unlike traditional neural networks, which rely on static representations of data, SNNs encode information in both the timing and the rate of spikes. This temporal dimension makes SNNs particularly adept at processing time-varying signals, such as sensory inputs from the environment, inspired by how biological organisms perceive and interact with the world.

This temporal aspect opens new possibilities for tasks such as event-based vision, auditory processing, and motor control, where time plays a crucial role. For example, SNNs are well suited for event-based cameras, which only capture changes in a scene rather than recording frames at fixed intervals. This allows for a more efficient representation of dynamic environments, where irrelevant static information can be discarded [3]. Similarly, in robotics, SNNs can process sensory inputs in real-time, allowing for more responsive and adaptive behavior [4].

Despite these advantages, SNNs face challenges that have hindered their widespread adoption, particularly compared to traditional deep learning models. The performance of SNNs on standard benchmark tasks, such as image classification or speech recognition, still lags behind that of DNNs [5]. One of the primary reasons for this is the difficulty in training SNNs. The spiking mechanism that gives SNNs their biological plausibility also introduces complexity when it comes to optimizing network parameters. Standard training methods such as backpropagation, which are central to the success of DNNs, are not directly applicable to SNNs due to the non-differentiable nature of spikes [5].

In response to these challenges, researchers have been exploring various methods to bridge the gap between the biological inspiration of SNNs and the practical demands of machine learning tasks. Efforts include the development of new learning algorithms, hybrid architectures that combine the strengths of both spiking and non-spiking networks, and techniques for converting pre-trained deep networks into spiking equivalents [1][5]. In addition, advances in neuromorphic hardware have made it possible to simulate large-scale SNNs more efficiently, further fueling research into their potential applications.

Although the performance of SNNs on conventional tasks remains a challenge, their unique properties, such as their temporal coding capabilities and energy efficiency, suggest that they may find their greatest success in applications where these traits offer a distinct advantage. As AI systems are increasingly deployed in real-world environments, the demand for efficient, adaptive, and responsive solutions will continue to grow. In this context, SNNs present a

promising avenue for developing AI that is not only powerful but also capable of operating under the constraints of real-time and low-power scenarios.

The study of SNNs is part of a broader movement towards more biologically inspired computation, where researchers aim to understand and replicate the mechanisms that allow biological brains to achieve such remarkable efficiency and adaptability. As this field matures, SNNs could play a pivotal role in the next generation of intelligent systems, offering a path toward more sustainable and energy-efficient AI.

This thesis is structured to provide a comprehensive exploration of SNNs and the various architectures and training methods explored throughout the research. Chapter 1 introduces the motivation behind the study, providing a background on SNNs and outlining the key challenges and objectives addressed. Chapter 2 reviews relevant literature on SNNs, their biological inspiration, and existing neural architectures, establishing a foundation for the novel approaches investigated in this work. Chapter 3 details the methodology, including the experimental setup, architectural innovations, and the training strategies used for improving SNN performance. Chapter 4 presents the results of these experiments, evaluating the effectiveness of the proposed architectures and training methods across various benchmarks. Chapter 5 discusses the implications of the results in the context of previous work, highlighting the strengths, limitations, and potential applications of the findings. Finally, Chapter 6 concludes the thesis with a summary of contributions, practical applications, and suggestions for future research directions.

2

Background and related work

This chapter delves into the theoretical underpinnings of SNNs, contextualized within the broader landscape of neuromorphic computing. By leveraging the discrete nature of neuronal spikes, SNNs depart from the traditional approach of continuous activations found in conventional artificial neural networks [5]. The ability to encode information through both the timing and rate of spikes allows SNNs to process data in a fundamentally different way, enabling efficient event-driven computation that more closely mirrors biological neural systems [1]. These characteristics make SNNs particularly attractive for real-time applications, especially in scenarios where energy efficiency is critical, such as edge computing and low-power embedded systems [2]. The discussion begins with the biological motivations behind SNNs, emphasizing the role of spikes as carriers of temporally precise information. The human brain operates using such spikes to communicate between neurons, and SNNs aim to replicate this efficient method of information transfer, allowing them to outperform traditional models in terms of power consumption and adaptability to temporal dynamics [5]. Furthermore, the asynchronous nature of spike-based computation is advantageous in scenarios where quick adaptation to changing stimuli is essential, such as in robotics and neuromorphic vision systems [2].

Beyond their biological inspiration, SNNs present several architectural innovations aimed at expanding their computational potential. One significant focus of this research is the development of network architectures capable of processing complex temporal patterns without sacrificing efficiency [5]. Recent approaches have extended SNNs beyond simple feedforward

models to deeper and more sophisticated architectures, including convolutional and recurrent SNNs designed to capture spatial and temporal features more effectively. The integration of such architectures has led to improvements in tasks such as dynamic visual recognition, where SNNs can capture and process high-dimensional spatio-temporal data with greater efficiency [2, 1]. Despite their promise, SNNs face challenges in terms of scalability and trainability. The reliance on spike timing introduces difficulties in applying traditional machine learning optimization techniques, such as backpropagation, to deep SNNs [1]. Strategies proposed in the literature to overcome these challenges include biologically plausible learning mechanisms, such as unsupervised methods like spike-timing-dependent plasticity, as well as hybrid strategies that incorporate elements of supervised and reinforcement learning [5]. Advances in hardware implementations designed specifically for SNNs, such as Intel's Loihi [6], offer promising avenues to scale these models to practical applications, reducing computational burden while maintaining high performance [2].

In light of these developments, this chapter also reviews SNN training methodologies, particularly focusing on the efforts to bridge the performance gap between SNNs and traditional deep learning models. The application of surrogate gradient methods and reward-modulated learning techniques has shown promise in enabling SNNs to learn more effectively from data while maintaining their energy-efficient properties [1]. These techniques, combined with the growing research on neuromorphic hardware, are expanding the potential of SNNs to tackle complex real-world tasks, from autonomous systems to neuromorphic visual processing [3, 2]. The chapter concludes by positioning the current research within this growing body of work, highlighting the limitations of existing methods and the knowledge gaps that this thesis aims to address. By examining the evolving architecture and training strategies of SNNs, this research aims to contribute to the broader understanding of how these biologically inspired networks can be optimized for large-scale real-time applications, ultimately enhancing their role in the future of artificial intelligence.

2.1 Overview of Spiking Neural Networks

SNNs are biologically inspired neural models that communicate through discrete spikes, offering advantages like energy efficiency and natural temporal information encoding. The study of SNNs spans various disciplines, including neuroscience, computational modeling, and artificial intelligence. By leveraging biological principles, SNNs achieve adaptive and efficient computation, bridging the gap between biologically plausible learning systems and the demands of

modern AI applications.

The following sections provide an in-depth exploration of the biological underpinnings of SNNs, their computational framework, various network architectures, and the unique challenges they face, particularly in the context of training and optimization. Through this overview, we aim to highlight the foundations that make SNNs a promising alternative to traditional deep learning models and their potential in neuromorphic computing.

2.1.1 Biological Foundations and Inspiration

The human brain, a highly complex and efficient computational organ, has evolved to process vast amounts of information rapidly and with minimal energy expenditure. Weighing approximately 1.4 kg and consuming about 20 watts of power, the brain is an exceptionally efficient biological machine, managing to perform complex cognitive tasks, including perception, decision-making, and motor control, with an energy consumption equivalent to that of a dim light bulb [7]. This exceptional efficiency has inspired decades of research that aims to replicate its mechanisms in artificial systems. Unlike traditional artificial neural networks, which operate on continuous signals, the brain processes information through discrete electrical events known as action potentials or spikes. Each neuron, of which there are an estimated 86 billion in the human brain, can form thousands of synaptic connections, resulting in an intricate web of approximately 100 trillion synapses. This connectivity enables the brain to engage in massively parallel processing, contributing to its ability to manage complex tasks while conserving energy.

This spike-based process is inherently more efficient and robust, using far less energy than continuous processing and enabling rapid response times in dynamic environments. The brain's method of information transmission, through discrete spikes, allows for adaptive, event-driven processing, where energy is consumed only when necessary. This model of spike-based information transmission is foundational to the design of SNNs, which aim to replicate these processes in artificial intelligence and neuromorphic computing [1, 5, 8].

In biological systems, neurons receive, integrate, and selectively transmit signals through specialized structures called synapses. When the accumulated input surpasses a specific threshold, neurons generate action potentials that propagate along the axon, triggering responses in connected neurons. This mechanism, termed the integrate-and-fire model, serves as a basis for SNNs, allowing the network to retain information while minimizing energy consumption, a principle that has inspired the event-driven nature of SNNs [1]. A schematic representation

of a neuron is reported in Figure 2.1. The design of SNNs leverages two essential properties of biological neural systems: temporal precision and adaptive plasticity. Temporal precision enables neurons to encode information through the timing and frequency of spikes, while adaptive plasticity, or synaptic plasticity, allows neurons to strengthen or weaken their connections based on activity patterns, supporting learning and memory formation. These features enable SNNs to model complex, time-varying data more accurately than continuous-valued ANNs, particularly in scenarios that require real-time processing and dynamic responses [8, 5].

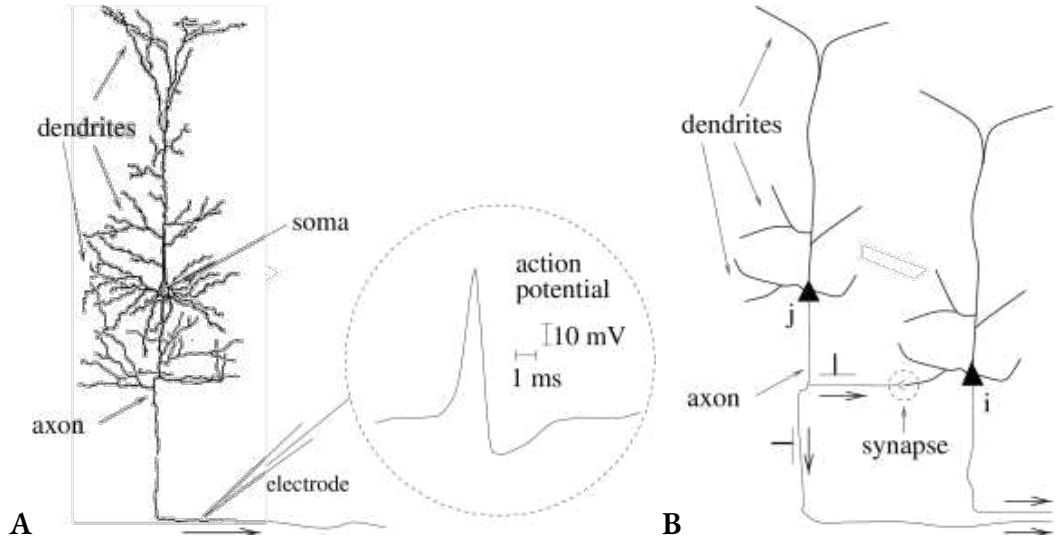
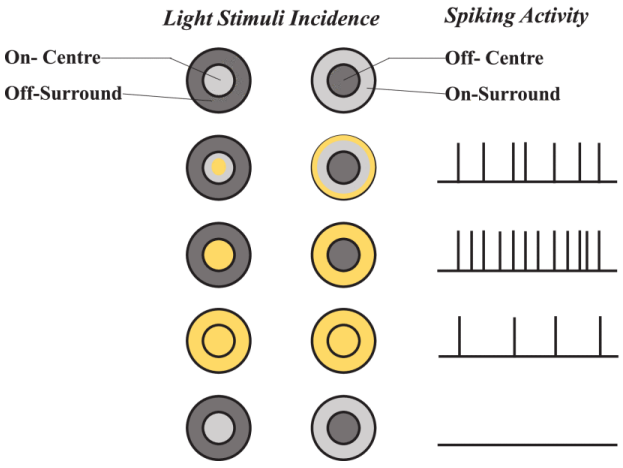


Figure 2.1: A. Single neuron in a drawing by Ramón y Cajal. Dendrite, soma, and axon can be clearly distinguished. The inset shows an example of a neuronal action potential (schematic). The action potential is a short voltage pulse of 1-2 ms duration and an amplitude of about 100 mV. B. Signal transmission from a presynaptic neuron *j* to a postsynaptic neuron *i*. The synapse is marked by the dashed circle. The axons at the lower right end lead to other neurons. Source: [9].

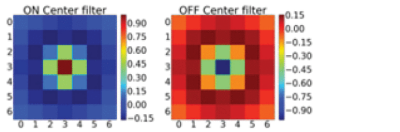
The efficiency of the human brain is also attributed to its hierarchical structure and modular organization, which allows different regions to specialize in particular functions, such as vision, language, or motor control, while maintaining the ability to integrate information seamlessly. This distributed processing approach, combined with the brain's inherent plasticity, makes it a model of inspiration for designing more advanced, adaptable, and energy-efficient artificial systems. By emulating these biological characteristics, SNNs hold the promise of advancing the field of artificial intelligence, particularly in areas where low-power consumption and real-time adaptability are crucial [7].

Insights from the Mammalian Retina

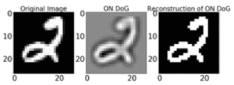
The mammalian retina provides a particularly compelling model for spike-based information processing, illustrating the power of spiking networks in handling sensory information. The retina comprises multiple layers of neurons, including photoreceptors, bipolar cells, and retinal ganglion cells, each responsible for encoding different aspects of visual information. Photoreceptors, including rods and cones, capture light signals and convert them into electrical signals. These signals are then relayed to bipolar cells, which play a crucial role in transmitting the information to retinal ganglion cells, the output neurons of the retina. Among the bipolar cells, a schematic representation of which is shown in Figure 2.2, there are two main types: on-centre and off-centre-surround bipolar cells. On-centre bipolar cells are activated when light hits the center of their receptive field, causing a depolarization that leads to signal transmission. In contrast, off-centre-surround bipolar cells are inhibited by light in the center of their receptive field but are activated by light in the surrounding area. This push-pull mechanism of activation and inhibition enhances the visual system's ability to detect contrast and distinguish edges, thereby contributing to a more efficient and precise encoding of visual information. This differentiation in response allows the visual system to emphasize changes in light intensity, such as edges and motion, which are critical for interpreting complex visual scenes.



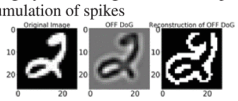
(a) Cells' response to light stimuli.



(a) On/Off DoG filters. On-centre filter has higher values in the centre as opposed to the Off-centre filter. Colour code shows the filter values.



(b) Left: Original gray-scale image. Centre: Output of the On DoG filter. Right: Accumulation of spikes



(c) Left: Original gray-scale image. Centre: Output of the Off DoG filter. Right: Accumulation of spikes

(b) DoG filters.

Figure 2.2: (a). On-centre- and Off-centre-surround bipolar cells' response to light stimuli. If there is maximum overlap between light incidence and the cell's RF, it outputs a maximum response pattern, demonstrated by the highest firing rates. However, if there is a weak correspondence, we observe a weak firing response or no spikes at all. **(b)**, On-centre/Off-centre DoG filters. Source: [5]

Retinal ganglion cells receive inputs from bipolar cells and produce spikes in response to

specific visual features such as brightness, contrast, and movement. These spikes are then transmitted to the brain via the optic nerve, where they are further processed to generate visual perception. Each type of ganglion cell is tuned to respond to particular aspects of the visual stimulus, thereby enabling an efficient and feature-specific representation of the visual environment. This layered processing compresses the vast amount of sensory data received by the retina into essential, feature-specific signals, conserving both processing time and energy. Importantly, retinal ganglion cells rely on both rate and temporal coding to maximize the efficiency and responsiveness to visual stimuli. Rate coding is used to represent the intensity of a stimulus, while temporal coding provides precise timing information that is crucial for detecting rapid changes in the environment. This dual coding strategy has inspired the development of neuromorphic sensors, such as dynamic vision sensors (DVS), which emulate the retina by emitting spikes only when they detect changes in motion or contrast. These sensors are capable of capturing visual information in a rapid and power-efficient manner, making them particularly advantageous in applications that require real-time, responsive systems, such as autonomous driving, robotics, and edge computing [3, 5]. The principles underlying retinal information processing illustrate how biological systems achieve an optimal balance between energy efficiency and computational effectiveness. By integrating mechanisms like on-centre and off-centre-surround processing, the mammalian retina efficiently extracts critical features from visual input, allowing for fast and adaptive responses. This spike-based approach to information processing not only serves as an inspiration for SNNs but also highlights the potential of SNNs in creating artificial systems that can replicate the high efficiency, adaptability, and precision found in biological vision.

Information Encoding in SNNs

One of the defining characteristics of biological neural systems is their remarkable ability to encode information not only in the rate of spikes, but also in the precise timing between them, known as temporal coding. This mechanism allows neurons to represent sensory information with greater fidelity, emulating how the brain processes dynamic environments. Temporal coding enables biological neurons to convey rich details about external stimuli, such as direction, intensity, and timing, facilitating efficient and adaptive responses to changing conditions. For instance, neurons in the auditory cortex leverage precise spike timing to detect subtle differences in sound wave frequencies, a feature critical for differentiating complex auditory patterns like speech and music [5]. Temporal coding is also pivotal in other sensory systems, such as the visual system, where retinal ganglion cells use it to detect motion and recognize rapid changes in

the visual scene. In the mammalian auditory system, neurons synchronize their spiking with the phase of incoming sound waves, thereby encoding information that goes beyond simple rate-based representation [1]. This precise temporal synchronization is crucial for distinguishing between closely spaced sound frequencies and identifying subtle auditory features, which are essential for understanding nuanced auditory signals such as language or music.

Rate coding represents another significant mechanism employed by biological and artificial neural systems. In rate coding, information is encoded in the average firing rate of neurons over a given time window. This method allows for a more stable representation of sensory input by smoothing out the variability in individual spike events, which can be advantageous in certain situations where robustness to noise is necessary. In biological systems, rate coding is often observed in sensory pathways, where the frequency of action potentials correlates with stimulus intensity. For instance, neurons in the visual and somatosensory cortices utilize rate coding to represent the intensity of light or tactile pressure. The primary advantage of rate coding lies in its simplicity and the ability to average out noise, providing a consistent and reliable signal that can be easily interpreted by downstream neurons.

However, traditional ANNs primarily rely on continuous activation functions, where information is represented by the strength of neuron activations rather than precise timing. This conventional approach can be limiting, as it often fails to capture the sequential order or precise timing of inputs, leading to a loss of critical temporal information necessary for processing dynamic and real-time data streams. While traditional ANNs perform well on static pattern recognition tasks, they struggle with tasks requiring an understanding of the temporal structure inherent in sequential data.

By adopting temporal coding, SNNs are able to detect and exploit intricate spatio-temporal patterns in data, enhancing their ability to process sequential information effectively [5, 8]. Temporal coding provides a richer and more dynamic way to represent information, allowing SNNs to excel in scenarios where capturing and responding to precise timing is essential. This capability is particularly advantageous for applications that require real-time audio-visual processing, robotics, and neuromorphic computing, where capturing and responding to temporal dynamics with minimal delay is crucial.

Hebbian Learning and Spike-Timing-Dependent Plasticity

Hebbian learning, often encapsulated by the phrase "neurons that fire together, wire together", is a foundational principle in neuroscience that describes how synaptic connections are strengthened through repeated co-activation of neurons. This mechanism provides a biologically plau-

sible basis for learning and memory formation, as it allows neurons that are consistently active together to reinforce their connections, thus encoding associations and patterns. Hebbian learning is unsupervised, relying solely on the correlation between neuronal activities rather than external labels or error signals [1].

In the context of SNNs, Hebbian learning has been extended to include Spike-Timing Dependent Plasticity, a learning rule that refines the Hebbian principle by incorporating the precise timing of neuronal spikes. STDP is a form of synaptic plasticity where the adjustment of synaptic strength is governed by the relative timing of spikes between pre- and post-synaptic neurons. Specifically, if a pre-synaptic spike precedes a post-synaptic spike within a critical time window, the synapse is potentiated (strengthened), whereas if the post-synaptic spike occurs before the pre-synaptic one, the synapse is depressed (weakened). This temporal dependence allows the network to encode causality and learn temporal correlations present in the input stimuli, making it well-suited for tasks that require precise timing, such as sensory processing and motor control [1]. As shown in Figure 2.3, synaptic strength is modulated by the relative timing of synaptic events and postsynaptic spikes, in the phenomena of LTD and LTP.

Mathematically, the change in synaptic weight Δw_{ij} between a pre-synaptic neuron i and a post-synaptic neuron j can be represented as:

$$\Delta w_{ij} = \begin{cases} A_+ e^{-\Delta t/\tau_+} & \text{if } \Delta t > 0 \\ -A_- e^{\Delta t/\tau_-} & \text{if } \Delta t < 0 \end{cases} \tag{2.1}$$

where $\Delta t = t_{\text{post}} - t_{\text{pre}}$ is the time difference between the post-synaptic spike (t_{post}) and the pre-synaptic spike (t_{pre}), A_+ and A_- are the maximum amounts of potentiation and depression, respectively, and τ_+ and τ_- are the time constants that determine the temporal window for potentiation and depression. This rule ensures that synaptic changes depend on the precise timing of spikes, allowing for the encoding of causal relationships between neuronal activities.

STDP plays a crucial role in both unsupervised and reinforcement learning paradigms. In unsupervised settings, STDP enables SNNs to extract features from the input data by reinforcing synaptic connections that contribute to consistently predictive patterns, allowing the network to learn without explicit labels. This form of learning supports the development of internal representations that reflect the statistical properties of the input, which can be beneficial for clustering and pattern recognition tasks [1]. Moreover, STDP has also been integrated into reinforcement learning through reward-modulated versions of the learning rule, such as reward-modulated STDP. In R-STDP, synaptic changes are influenced not only by the relative

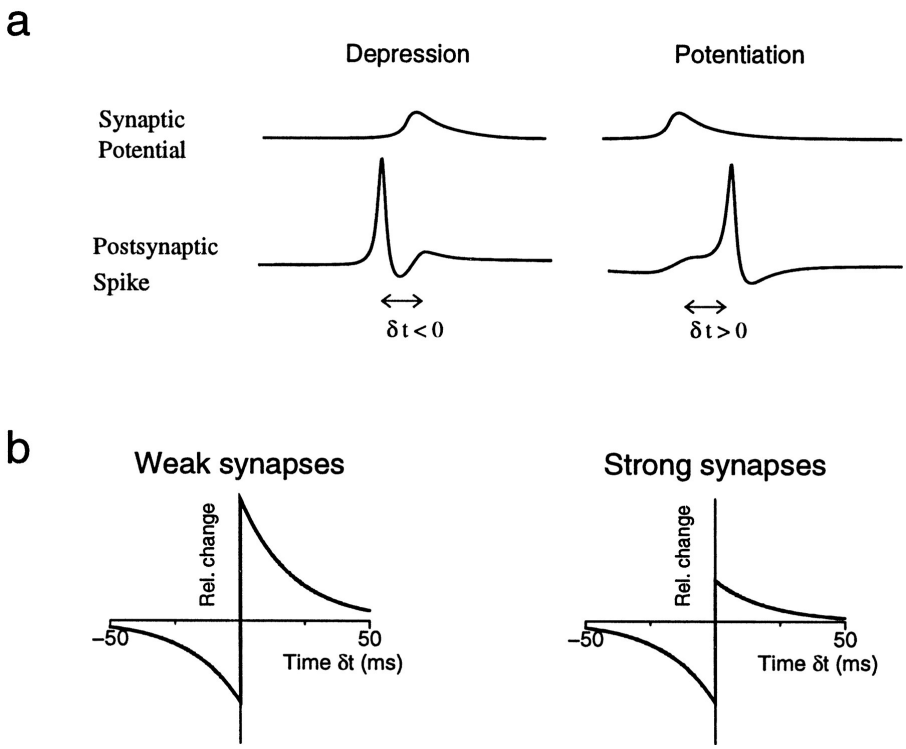


Figure 2.3: Spike timing-dependent plasticity. **(a)** Synapses are strengthened when the synaptic event occurs before the postsynaptic spike, while they are weakened when the synaptic event follows the postsynaptic spike. **(b)** The window for synaptic modification is time-sensitive, with the magnitude of synaptic change plotted against the time difference between the synaptic event and the postsynaptic spike. The effect decreases exponentially as the time difference increases. Additionally, the amount of potentiation diminishes for stronger synapses, whereas the level of depression remains consistent regardless of synaptic strength. Source: [10]

timing of spikes but also by a global reward signal that modulates the degree of synaptic potentiation or depression. This reward signal acts as a neuromodulator, similar to how dopamine functions in biological systems, reinforcing behaviors that lead to positive outcomes and weakening those that result in negative feedback [5]. The role of neuromulators is further expained in Figure 2.4.

The biological plausibility of STDP and its variants makes them particularly attractive for neuromorphic computing, where the goal is to replicate the efficiency and adaptability of biological neural systems. Unlike traditional backpropagation, which requires a global error signal and extensive computational resources, STDP operates locally at each synapse, enabling distributed and parallel learning that is energy efficient. This local learning mechanism allows SNNs to continuously adapt to changing environments in real-time, which is crucial for applications such as robotics, autonomous systems, and sensory processing [1]. In addition to

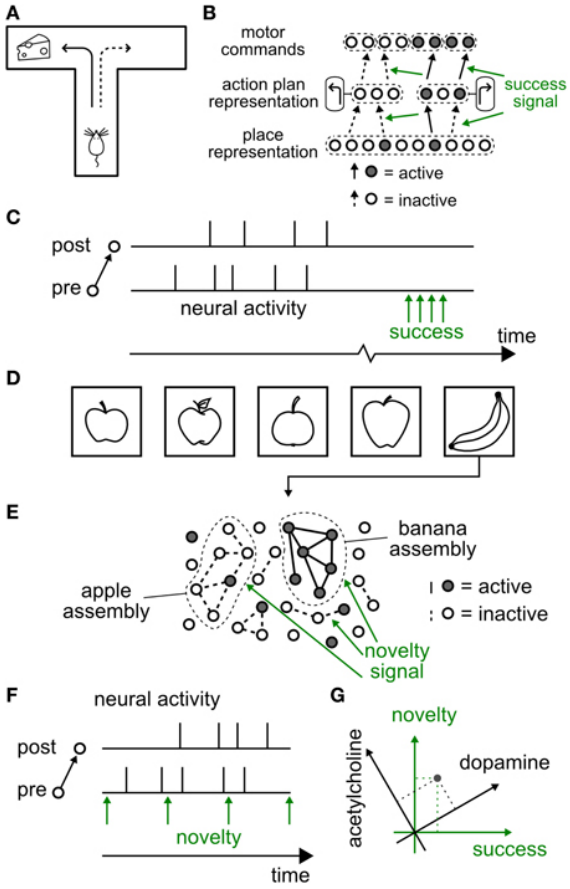


Figure 2.4: Hypothetical role of neuromodulated synaptic plasticity. (A–C) Reward-modulated learning: An animal learns an action sequence (e.g., moving in a T-maze) via trial and error with rewards. The hippocampus represents the current position and influences neurons coding for actions like turning left or right, which project to motor neurons. A success signal, indicating reward outcome, modulates plasticity at synapses marked by coincident pre- and postsynaptic activity (A, B). Neuromodulatory signals occur much later than neural activity (C). (D–F) Novelty-modulated learning: Novelty occurs when a stimulus does not match prior experience. If a network trained to recognize an apple sees a banana, a novelty signal triggers synaptic strengthening between neurons representing the new stimulus (D, E). The timing of novelty signals may coincide with or follow neural activity (F). (G) Neuromodulators like dopamine can transmit signals combining novelty and success, potentially involving multiple neuromodulators such as acetylcholine and dopamine. Source: [1].

its role in unsupervised and reinforcement learning, STDP also facilitates the emergence of complex network dynamics that resemble those observed in biological neural circuits. By shaping synaptic strengths based on spike timing, STDP helps to organize the network into functional modules that can process information more effectively. This self-organizing capability is a key advantage of SNNs over traditional ANNs, particularly in tasks that involve temporal sequences and require a high degree of temporal precision [1].

Neuromodulation and Adaptive Learning

Biological neurons exhibit an additional level of complexity through neuromodulation, where chemicals such as dopamine, acetylcholine, and other neuromodulators influence neural signaling, allowing neural systems to dynamically adapt to changing environmental conditions. This neuromodulation prioritizes certain stimuli over others based on factors like reward, novelty, or importance, fundamentally contributing to learning and memory processes [1, 5]. Neuromod-

ulators can regulate the likelihood of spike transmission and alter synaptic strengths, thereby modulating the plasticity of neural circuits. This adaptive mechanism is key to how biological systems manage learning in a real-time, context-sensitive manner, by altering the impact of stimuli based on the significance they hold at any given moment.

In biological systems, neuromodulation works by adjusting synaptic plasticity, often described through models such as STDP, which modifies synaptic strength based on the relative timing of spikes between neurons. However, neuromodulators, such as dopamine, add an additional layer to STDP by influencing when and how strongly these plasticity changes occur, essentially providing a context-dependent gating mechanism for synaptic changes. For instance, dopamine is often associated with reward, signaling when a particular event is important enough to reinforce synaptic changes, thereby linking neuromodulated STDP to reinforcement learning paradigms [1]. This mechanism is described by three-factor learning rules, where synaptic changes are determined not only by the pre- and post-synaptic activity but also by the influence of neuromodulatory signals, thus allowing synapses to adapt based on the perceived value or importance of the stimulus.

In artificial SNNs, this concept of neuromodulation has inspired advanced training techniques, such R-STDP, which incorporates a reward signal that modulates synaptic plasticity, mimicking the role of biological neuromodulators in reinforcing successful behaviors. The reward signal functions similarly to how dopamine acts in biological systems, providing feedback that enhances synaptic efficacy when an outcome is deemed beneficial, thus enabling the network to learn behaviors that maximize rewards over time [1]. This approach has proven particularly effective in tasks that require adaptive and real-time responses, where the flexibility of reward-based plasticity is crucial to managing the dynamic and often unpredictable nature of the environment. Furthermore, surrogate gradient methods have been developed to address the challenges of training SNNs, especially when leveraging neuromodulation-inspired plasticity. These methods allow for backpropagation-like optimization, even in spiking networks, by approximating gradients that are not directly computable due to the non-differentiable nature of spikes. When combined with neuromodulation-inspired rules like R-STDP, surrogate gradients provide a powerful tool for training deep SNN architectures, bridging the gap between biologically plausible learning and efficient computational methods. By integrating these biologically inspired mechanisms—such as neuromodulated plasticity and surrogate gradient optimization—SNNs can capture a level of adaptability and efficiency reminiscent of biological neural systems [11]. This combination allows SNNs to perform well in complex environments, particularly in scenarios where the ability to learn from rewards and adapt synaptic

strengths dynamically can lead to improved decision-making and real-time processing capabilities [1, 5].

Implications for Artificial Intelligence and Neuromorphic Computing

SNNs fully exploit their event-based nature when deployed on specialized neuromorphic hardware platforms, such as Intel's Loihi [6] and IBM's TrueNorth [12], which are optimized for event-driven computation. These hardware platforms support large-scale parallel processing while minimizing energy usage, a critical advantage for AI applications operating in resource-constrained environments [6, 13].

Neuromorphic hardware such as Loihi and TrueNorth is designed to implement key features of SNNs, including temporal coding, asynchronous processing, and local synaptic plasticity. Temporal coding allows information to be encoded in the precise timing of spikes, enabling the network to capture the temporal structure of sensory inputs effectively [5]. This is particularly advantageous for applications involving continuous sensory data, such as vision and auditory processing, where the ability to respond to temporal cues is crucial. For instance, dynamic vision sensors, which mimic the human retina, leverage the event-driven nature of SNNs to provide low-latency, power-efficient visual processing, making them ideal for real-time applications such as robotics and autonomous navigation [3].

The efficiency and adaptability of SNNs make them highly attractive for AI applications that require continuous interaction with dynamic environments. In robotics, for example, SNNs can be used to control movement by continuously adapting to sensory feedback, providing a level of responsiveness that is challenging to achieve with traditional ANNs [5, 1]. Additionally, the low-power requirements of SNNs make them suitable for edge computing scenarios, where devices must operate autonomously with limited energy resources. This is particularly relevant for the Internet of Things (IoT), where distributed smart sensors can benefit from the energy efficiency and on-device learning capabilities of SNNs [6, 3]. Furthermore, the modular and distributed nature of SNNs allows scalable architectures that can be deployed on neuromorphic chips to achieve high-performance computing with minimal energy consumption. The ability of SNNs to perform both unsupervised and reinforcement learning, combined with their event-driven operation, positions them as a key technology for the next generation of AI systems, particularly in applications that demand real-time processing, adaptability, and energy efficiency [1, 13]. By drawing inspiration from the structure and functionality of biological neural systems, SNNs represent a significant shift towards more sustainable and biologically-inspired AI, paving the way for advancements in neuromorphic computing and intelligent sys-

tems.

2.1.2 Spiking Neural Networks: Computational Model

In SNNs, neurons communicate through discrete events called spikes, mirroring the way biological neurons interact. Unlike traditional artificial neurons, which use continuous signals to transmit information, spiking neurons rely on action potentials, brief and sudden changes in membrane voltage, that propagate through the network. This biologically inspired mechanism forms the core of the SNN computational model, providing several advantages in terms of efficiency, adaptability, and the ability to encode temporal information.

Leaky Integrate-and-Fire Neuron Model

The Leaky Integrate-and-Fire (LIF) neuron model serves as one of the most fundamental computational units in SNNs, providing a simplified but powerful representation of biological neuron dynamics. This model captures key features of biological neurons, such as the integration of incoming spikes, the leaky nature of the membrane potential, and the emission of action potentials when a certain threshold is exceeded [9, 5]. A schematics of the LIF neuron model is in Figure 2.5.

In the LIF model, each neuron integrates incoming spikes, accumulating their effect in its membrane potential, denoted as $V(t)$. The evolution of the membrane potential is governed by the following differential equation:

$$\tau_m \frac{dV(t)}{dt} = -V(t) + R_m I(t) \tag{2.2}$$

where τ_m is the membrane time constant, R_m represents the membrane resistance, and $I(t)$ denotes the input current resulting from synaptic inputs [9]. The membrane potential $V(t)$ decays exponentially over time if no additional input is received, reflecting the leaky property of the neuron. This decay process ensures that the membrane potential does not remain indefinitely elevated, allowing the neuron to integrate new information effectively while "forgetting" older inputs, which is crucial for dynamic, real-time signal processing.

The LIF model also incorporates a threshold mechanism. When the membrane potential $V(t)$ exceeds a predefined threshold V_{th}, the neuron fires an action potential, or "spike," and the potential is reset to a resting value V_{reset}, typically set to zero or another low value:

$$V(t) \geq V_{\text{th}} \implies V(t) \to V_{\text{reset}} \tag{2.3}$$

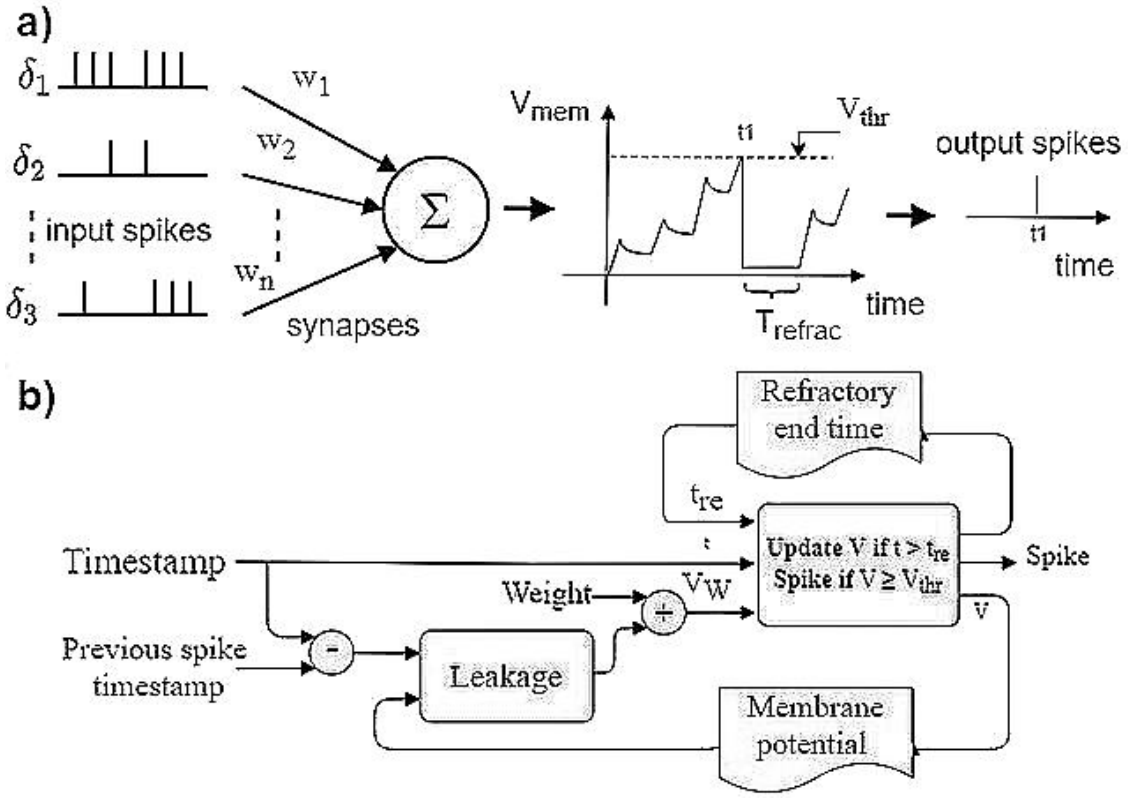


Figure 2.5: LIF neuron model. (a) Behavior of the neuron. (b) Digital behavior, neurons receive spikes in the form of a timestamp. It integrates them if it is not in the refractory period. It would emit a spike if the membrane potential exceeded the threshold. The leakage block is implemented with a bit shift for resource savings. Source: [14]

This process of thresholding and resetting is akin to the refractory period observed in biological neurons, where a neuron cannot immediately fire again after emitting a spike, thus preventing overexcitation and promoting temporal regulation of spiking activity [1, 5].

The input current $I(t)$ in the LIF model can represent multiple synaptic events, including both excitatory and inhibitory inputs. Excitatory inputs increase $V(t)$, making it more likely for the neuron to reach the firing threshold, whereas inhibitory inputs decrease $V(t)$, reducing the likelihood of spiking. This balance between excitation and inhibition plays a critical role in maintaining stability and preventing uncontrolled activity within neural networks [9]. The interaction between these opposing forces allows the LIF model to replicate the natural behavior of biological neural circuits, where inhibition is essential for regulating excitatory signals and ensuring efficient processing of sensory information.

The dynamics of the LIF model can also be explored in the frequency domain to understand how neurons respond to oscillatory inputs. Given an input current of the form $I(t) = I_0 e^{i\omega t}$,

the frequency response of the membrane potential is characterized by:

$$\left|\hat{V}(\omega)\right| = \frac{R_m I_0}{\sqrt{1 + (\omega\tau_m)^2}} \tag{2.4}$$

This frequency response indicates that the LIF neuron behaves like a low-pass filter, preferentially responding to lower frequency inputs while attenuating high-frequency components [9]. This filtering property is beneficial for eliminating noise and focusing on the most relevant temporal features of the input signal.

The LIF model is often used as the basis for building larger networks of SNNs, where it serves as a foundational unit for studying emergent behaviors such as synchronization, oscillatory activity, and spatiotemporal pattern formation. By adjusting parameters such as the membrane time constant τ_m, threshold V_{th}, and synaptic weights, the LIF model can be tuned to exhibit a wide range of behaviors that are observed in biological neural systems. Additionally, the simplicity of the LIF model makes it computationally efficient, allowing for its integration into neuromorphic hardware platforms, which are designed to support large-scale simulations of spiking networks with minimal energy consumption [5, 1].

Temporal Dynamics and Spike Generation

The temporal dynamics of Leaky LIF neurons enable SNNs to process spatio-temporal data in a more natural and effective way compared to traditional neural networks. In an SNN, the firing of a neuron is determined not only by the magnitude of the incoming current but also by the precise timing of these inputs. This feature allows SNNs to utilize temporal coding, where the exact timing of individual spikes carries significant information, unlike rate coding, which relies on the average firing rate of neurons over time [15, 9]. Temporal coding gives SNNs a unique advantage in tasks involving sensory processing, where timing is crucial, such as in auditory localization or visual motion detection [16, 5].

In auditory processing, for example, the temporal differences between the spikes generated by sounds that reach each ear allow the network to determine the direction of the sound source with high precision [9]. Similarly, in visual motion detection, the sequence and timing of spikes encode information about the movement of objects across the visual field. This temporal structure helps SNNs distinguish between different movement patterns and respond accordingly [15, 5].

Spike generation in an SNN is influenced by several key factors, including the synaptic weights of incoming connections, the temporal integration of inputs, and the inherent leaky dynamics

of the neuron's membrane potential. When a presynaptic neuron fires, it generates a spike that travels across the synapse and contributes to the membrane potential of the postsynaptic neuron. The magnitude of this contribution is determined by the synaptic weight, which can be either excitatory or inhibitory, affecting the likelihood that the postsynaptic neuron will reach the firing threshold [1, 9]. The membrane potential of a LIF neuron accumulates input from multiple synapses over time, and this accumulation is subject to a constant decay due to the leaky nature of the membrane. This means that inputs arriving close together in time have a greater chance of causing the neuron to fire, while inputs that are spaced far apart may decay before the neuron can reach the firing threshold. This temporal integration allows LIF neurons to act as coincidence detectors, effectively responding to correlated inputs that arrive within a short temporal window [5, 9]. The exact timing of spike generation is crucial for encoding information in SNNs. Unlike traditional artificial neurons that output continuous values, LIF neurons emit discrete spikes, and the temporal pattern of these spikes forms the basis of information processing. The timing of a spike is influenced by both the excitatory and inhibitory inputs a neuron receives, as well as the rate at which the membrane potential leaks. When the cumulative effect of all incoming spikes pushes the membrane potential above the threshold, a spike is generated, and the membrane potential is reset. This reset mechanism introduces a refractory period during which the neuron is temporarily unable to fire, ensuring that spikes are temporally spaced and that the neuron does not become overactive [16, 1].

The ability to precisely control spike timing gives SNNs a significant advantage in representing and processing temporal information. For example, in sensory systems, the precise timing of spikes can be used to detect subtle differences in stimulus features, such as the direction of a moving object or the location of a sound source. This temporal precision is facilitated by mechanisms like STDP, which adjusts synaptic weights based on the relative timing of pre- and post-synaptic spikes, thereby reinforcing temporal patterns that are behaviorally relevant [1, 9]. Furthermore, the interaction between excitatory and inhibitory inputs plays a crucial role in shaping the temporal dynamics of spike generation. Excitatory inputs increase the membrane potential, making it more likely for the neuron to fire, while inhibitory inputs counteract this effect by decreasing the membrane potential. The balance between excitation and inhibition ensures that the neuron responds selectively to specific temporal patterns, filtering out noise and preventing excessive firing [13, 5]. This balance is essential for maintaining stability in the network and for enabling complex behaviors such as pattern recognition, synchronization, and rhythmic activity [16]. Temporal dynamics in LIF neurons also contributes to emergent behaviors observed in larger SNNs. By adjusting parameters such as the membrane time constant,

synaptic weights, and the firing threshold, these neurons can exhibit behaviors similar to those found in biological systems, such as oscillatory activity and synchronization. These emergent properties are particularly beneficial for neuromorphic computing, where networks of spiking neurons are used to perform complex computations in an energy-efficient manner [13, 5]. The ability of SNNs to leverage precise spike timing and adapt synaptic weights based on temporal correlations makes them highly suitable for real-time applications, including robotics, sensory processing, and autonomous systems.

Energy Efficiency and Event-Driven Processing

One of the key advantages of SNNs over traditional ANNs is their inherent energy efficiency. Biological neurons are generally silent, firing only when they need to communicate essential information. This sparseness is replicated in SNNs, where neurons remain quiescent unless activated by relevant input. The event-driven processing nature of SNNs implies that neurons compute only when spikes are received, thereby reducing redundant calculations and saving energy [2, 1].

Event-driven computation is not only energy-efficient but also computationally efficient, allowing SNNs to handle asynchronous data streams, much like biological neural systems. This means that an SNN can process inputs as they arrive, without waiting for synchronous clock cycles, making them particularly well-suited for real-time processing tasks in neuromorphic hardware [15, 5]. For example, neuromorphic platforms like Intel's Loihi have been designed to exploit this event-driven approach, enabling low-power, large-scale implementations of SNNs suitable for on-device AI applications [13].

Synaptic Dynamics, Learning Mechanisms, and Model Adaptation in SNNs

The ability of SNNs to learn and adapt is fundamentally rooted in their synaptic dynamics and the biological principles that inspire them. Synaptic plasticity, seen as the ability of synapses to strengthen or weaken over time in response to neural activity, is central to both biological learning and the learning capabilities of SNNs. The learning processes in SNNs are made possible through several mechanisms, including STDP, R-STDP, surrogate gradient methods, and the adaptation of pre-trained ANNs into spiking counterparts. Each of these methods plays a critical role in enabling SNNs to learn temporal patterns, optimize synaptic connections, and bridge the gap between conventional and biologically plausible learning.

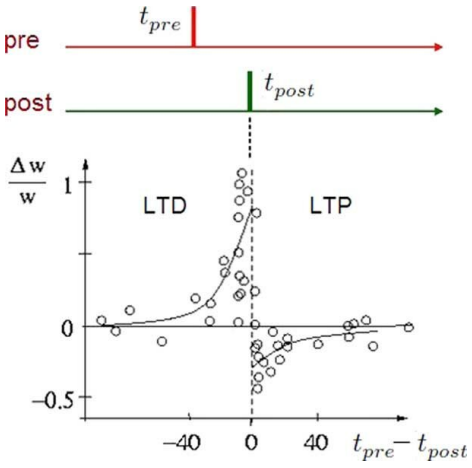
STDP AND R-STDP STDP is a form of Hebbian learning in which the synaptic weight is adjusted according to the precise timing of spikes from presynaptic and postsynaptic neurons. If a presynaptic spike occurs shortly before a postsynaptic spike, the synaptic connection is strengthened, reinforcing the causal relationship between the two neurons. This process is referred to as Long-Term Potentiation (LTP), which results in a long-lasting increase in synaptic strength. In contrast, if the postsynaptic spike precedes the presynaptic spike, the synapse is weakened, a process known as Long-Term Depression (LTD), leading to a reduction in synaptic strength. The referring equation is given in 2.1. This temporal-based adaptation allows SNNs to capture causal relationships in input data, which is crucial for processing temporal sequences, sensory data, and dynamic environments.

Reward-Modulated STDP extends the concept of STDP by incorporating a reward signal, akin to biological neuromodulation processes. In R-STDP, synaptic adjustments are influenced by both spike timing and the presence of a reward or punishment signal, indicating the success or failure of a particular output. Practically, R-STDP involves maintaining an eligibility trace for each synapse, which accumulates based on the timing of presynaptic and postsynaptic spikes. When a reward signal is received, this eligibility trace is used to update synaptic weights accordingly. The eligibility trace is a decaying memory of recent activity, ensuring that synaptic changes reflect the temporal sequence of neural activity. This reward-based adjustment is particularly useful in reinforcement learning tasks where an agent must adapt its behavior to maximize a cumulative reward [1, 2].

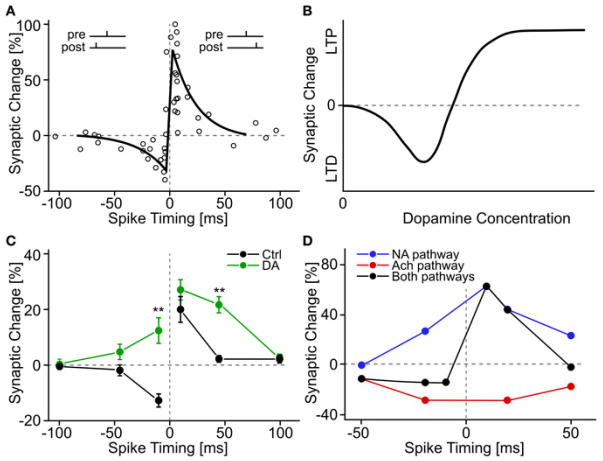
The weight change rule for R-STDP can be expressed as:

$$\Delta W_{ij} = \begin{cases} \eta_+ e^{-\Delta t/\tau_e} R, & \text{if reward and } \Delta t > 0, \\ \eta_- e^{\Delta t/\tau_e} R, & \text{if reward and } \Delta t < 0, \end{cases} \tag{2.5}$$

where η_+ and η_- are learning rates for potentiation and depression, τ_e represents the eligibility decay rate, and R is the reward signal received based on the network's performance [2]. This mechanism is particularly useful in reinforcement learning tasks, where the objective is to maximize the cumulative reward over time. R-STDP provides a biologically plausible way to link neural activity to behavioral outcomes, allowing the network to self-organize based on experience and adapt dynamically to changes in the environment [1]. Experimental data corroborating this model are in Figure 2.6.



(a) Spike-timing dependent plasticity scheme



(b) STDP measurements

Figure 2.6: (a). Schematic representation of spike-timing-dependent plasticity. The STDP function depicts changes in synaptic weight as a function of the relative timing between pre-synaptic and post-synaptic spikes. The experimental recordings are represented by circles, and the solid curve illustrates a fitted STDP function. Source [17]. **b.** Experimental evidence for STDP and neuromodulated STDP. (A) A classic example of STDP measurement. Each circle represents synaptic change as a function of the time difference between pre- and postsynaptic spikes, with a black line indicating an "STDP window." Data from hippocampal cell culture from rat. (B) Summary effect of dopamine concentration on long-term plasticity in corticostriatal synapses. (C) Effect of extracellular dopamine (DA) on the STDP window in hippocampal cell culture from rat. (D) Effect of activation of neuromodulatory pathways (noradrenaline (NA) via β-adrenergic receptors, and acetylcholine (Ach) via M1-muscarinic receptors) on the STDP window in rat visual cortex slices. Source: [1]

Surrogate Gradient Methods Training SNNs with traditional backpropagation reveals a challenge because of the non-differentiable nature of spike events. Unlike ANNs, where the activation functions are differentiable, the discontinuous spikes in SNNs prevent direct gradient computation. To overcome this, surrogate gradient methods are employed. These methods approximate the gradient of the spiking function using a continuous surrogate function, which makes it possible to apply gradient-based optimization techniques such as Stochastic Gradient Descent (SGD). The surrogate gradient is typically chosen to be a sigmoid or piecewise linear function that approximates the steep change in the membrane potential at the spiking threshold. For instance, the surrogate gradient $\hat{\delta}(V)$ for a neuron's membrane potential $V(t)$ can be defined as:

$$\hat{\delta}(V) = \frac{1}{1 + e^{-\beta(V - V_{\text{th}})}} \tag{2.6}$$

where β controls the steepness of the gradient and V_{th} is the spiking threshold [5, 1]. This approximation enables the computation of gradients during the backward pass, allowing the network to be trained in a manner similar to ANNs. The use of surrogate gradients has signifi-

cantly improved the ability to train deep SNNs and achieve performance on par with ANNs for tasks such as image recognition and classification. Surrogate gradient methods are essential for training SNNs in a supervised fashion, making it possible to apply established deep learning techniques while retaining the energy-efficient, event-driven benefits of spiking neurons. By using surrogate gradients, the network can learn to optimize weights in response to both spatial and temporal patterns in the data, making these methods suitable for complex tasks such as speech recognition and neuromorphic vision systems. In Figure 2.7 the visualization of the approximation used in surrogate gradient methods.

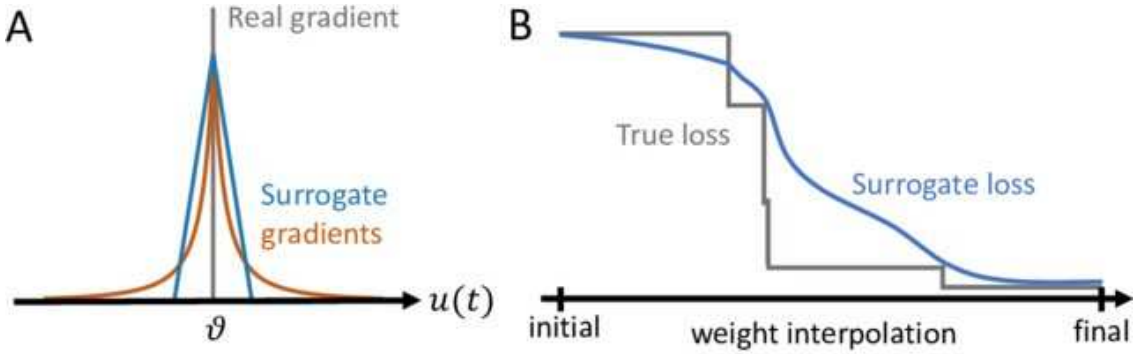


Figure 2.7: A. The threshold function's gradient is a Dirac delta, which spikes to infinity at $u = \theta$ and is zero elsewhere (gray). To circumvent this, the surrogate gradient method replaces this sharp gradient with a smoother approximation, such as a mirrored Lorentz function (orange) or a triangular function (blue). **B**. In SNNs, the loss function exhibits discontinuities, especially plateaus of constant loss that impede learning (gray). Surrogate gradients are used to 'smooth' the loss landscape (blue trace), promoting better learning dynamics. The illustrated loss landscape represents an interpolation between two weight sets of a SNN. Source: [18]

Adaptation of Pre-Trained ANNs into SNNs A further approach to leverage the strengths of traditional ANNs in SNNs is the adaptation of pre-trained ANN models into spiking counterparts. This conversion process begins with training an ANN using standard backpropagation techniques, followed by converting it into an SNN by replacing activation functions with spike-based equivalents. During this adaptation, the network's weights are preserved, but additional adjustments are made to account for the differences in activation dynamics. Typically, the firing rates of spiking neurons in the converted SNN are made proportional to the continuous activation values of the original ANN. This rate coding allows the SNN to approximate the behavior of its ANN counterpart, while still benefiting from the sparse, event-driven computation that characterizes SNNs [1, 5]. One of the critical challenges during this conversion is determining appropriate thresholds and scaling factors for the spiking neurons

to ensure the network maintains its performance. Weight normalization techniques are often applied to facilitate this transition, ensuring that the spiking thresholds are set in a way that closely matches the activation dynamics of the original ANN. The ANN-to-SNN conversion approach provides a practical way to utilize the power of deep learning while incorporating the energy efficiency of spiking neurons. This hybrid approach is particularly effective for tasks where pre-trained feature extractors can be used to identify spatial features, while the spiking component is employed to handle the temporal aspects of the input data. The integration of pre-trained ANN models with spike-based fine-tuning has been shown to be effective for real-time applications, such as robotics and sensory processing, where both spatial and temporal precision are crucial [13, 1].

Network-Level Computation: Spatio-Temporal Processing

At the network level, the computational model of SNNs enables spatio-temporal processing, where information is encoded in both the spatial pattern of active neurons and the temporal sequence of their spiking. This dual encoding allows SNNs to extract meaningful features from dynamic input streams, such as recognizing moving objects in visual scenes or interpreting complex auditory signals.

The connectivity between neurons in SNNs often includes both feedforward and recurrent connections, allowing for the propagation of information across layers in a manner similar to both traditional deep learning models and biological brains. Recurrent connectivity in particular provides SNNs with a form of memory, as previous spikes influence future activity, enabling the network to maintain a temporal context that is crucial for tasks like speech recognition and sequential pattern classification [16, 5].

Implications for Neuromorphic Hardware

The event-driven and energy-efficient characteristics of SNNs have significant implications for neuromorphic computing. Neuromorphic hardware is designed to mimic the architecture and operational principles of biological brains, implementing spiking neurons and synapses directly in silicon. By using asynchronous circuits, neuromorphic chips like Loihi can emulate the spiking behavior of biological neurons, performing complex computations with extremely low power consumption. This architecture makes SNNs a natural choice for deployment in real-world applications that require continuous, real-time interaction with their environment, such as robotics, autonomous navigation, and wearable devices [13, 15].

2.1.3 Architectures of SNNs

SNNs can be applied to a range of different architectures, each tailored to solve specific types of tasks. The architectures of SNNs are designed to leverage the spiking nature of neurons to achieve efficient and effective computations for both spatial and temporal information processing. The following subsections provide an overview of several key SNN architectures and their respective strengths.

Feedforward Spiking Neural Networks Feedforward SNNs are the simplest form of spiking network, where information flows unidirectionally from input to output without feedback loops. These networks are used primarily for basic classification and pattern recognition tasks. Feedforward SNNs function similarly to traditional artificial feedforward neural networks, but with the added benefit of temporal coding, allowing them to encode information more accurately through spike timing. This architecture is well-suited for tasks such as static image recognition or sensor data interpretation, where a direct mapping from input to output is sufficient [15].

Recurrent Spiking Neural Networks Recurrent SNNs (RSNNs) incorporate feedback connections, forming cycles within the network that allow information to be maintained and revisited over time. These recurrent connections introduce memory-like properties, enabling RSNNs to process sequential data effectively. RSNNs are ideal for tasks that require contextual information and temporal dependencies, such as speech recognition, sequence prediction, and dynamic signal analysis [16]. The recurrence also enables RSNNs to exhibit behaviors such as temporal integration and sustained activation, allowing them to solve more complex tasks compared to feedforward architectures.

Convolutional Spiking Neural Networks Convolutional Spiking Neural Networks (CSNNs) extend the concept of convolutional neural networks into the spiking domain. By incorporating convolutional filters, CSNNs are capable of capturing spatial hierarchies in data, making them suitable for tasks involving image and video recognition. CSNNs exploit both spatial and temporal features by processing spatiotemporal patterns through convolutional layers, followed by spiking neurons that produce event-driven responses. This makes CSNNs particularly advantageous for dynamic vision sensing, where visual scenes change over time, such as object tracking, gesture recognition, and visual attention tasks [8, 13]. CSNNs benefit from the integration of temporal coding mechanisms that enhance their ability to capture dynamic

changes within visual data. By leveraging the temporal precision of spikes, CSNNs can achieve a more nuanced representation of spatiotemporal relationships, making them highly effective for applications that require both spatial feature extraction and temporal event detection. This has led to CSNNs being applied in areas like autonomous navigation and robotics, where recognizing and responding to changing environments in real time is critical [19].

Spiking Deep Neural Networks The concept of deep learning can also be applied to SNNs, giving rise to Deep Spiking Neural Networks. These architectures consist of multiple layers of spiking neurons, similar to the layers in deep artificial neural networks. By stacking layers, deep SNNs can learn progressively more abstract representations of data, which enhances their capacity for feature extraction and pattern recognition [2]. Training deep SNNs is challenging due to the non-differentiable nature of spike generation. However, recent advances in surrogate gradient methods have made it possible to approximate traditional backpropagation, enabling effective training of deep spiking architectures. Surrogate gradient methods involve defining a differentiable proxy for the non-differentiable spike function, which allows gradient-based optimization to be used during training. This approach has led to significant improvements in the performance of deep SNNs, making them capable of solving complex tasks such as image classification and natural language processing [5].

Liquid State Machines One of the most promising SNN architectures is the Liquid State Machine. LSMs belong to a class of models known as reservoir computing. A schema is reported in Figure 2.8. They consist of a large, randomly connected pool of spiking neurons, the "liquid", which processes time-varying input by exploiting its internal transient dynamics. The reservoir acts as a high-dimensional projection space, where complex input signals are transformed into rich temporal patterns that are subsequently read by a simple classifier. LSMs are inspired by the intricate recurrent structure of the cortex in biological brains and leverage this randomness to generate a wide range of spatiotemporal responses. The high-dimensional representation produced by the reservoir can be used for classification, regression, or decision-making tasks. Liquid State Machines have been particularly successful in real-time processing tasks such as speech recognition and radar-based hand gesture recognition, where the ability to generate and analyze complex temporal dynamics is crucial [15, 16, 8, 19].
The adaptability of LSMs also makes them well-suited for applications that require continuous adaptation to streaming input data, such as in robotic control, event-based sensing, and neuromorphic signal processing. Unlike traditional architectures that rely on precise, fixed weights,

LSMs use the reservoir to create transient states that are inherently adaptable to varying input conditions, providing a powerful mechanism for real-time computation and decision-making.

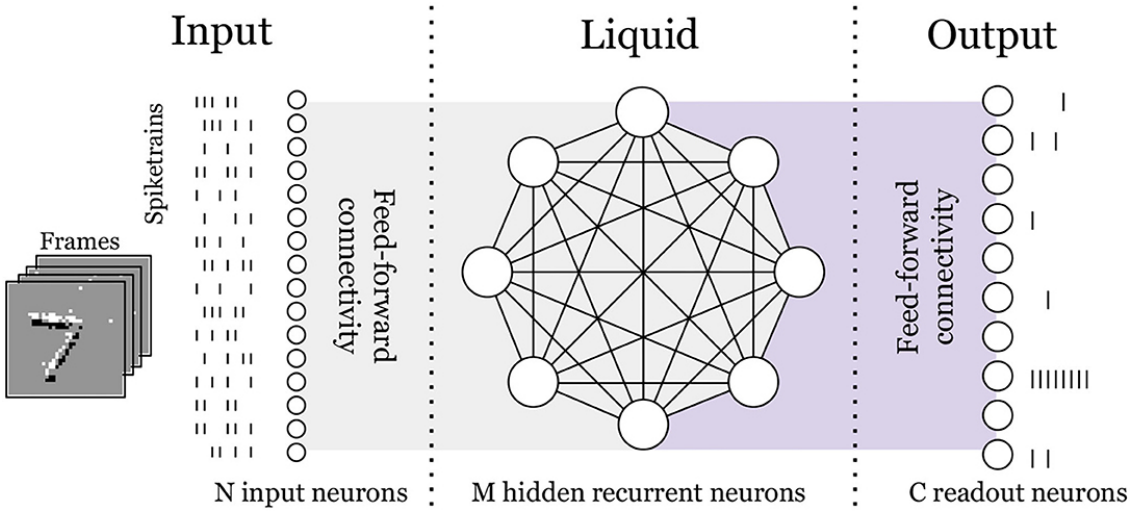


Figure 2.8: Illustration of the LSM architecture for processing a 2D temporal input (represented as frames) into a 1D spike train. The number of input neurons (N) and hidden neurons (M) is reduced for simplification purposes. The gray-shaded region represents the non-trainable synaptic connections, whereas the purple-shaded region indicates trainable synaptic weights. Source: [20]

Reservoir Computing and Echo State Networks In addition to LSMs, another form of reservoir computing model adapted to spiking neurons is the Echo State Network (ESN). ESNs, like LSMs, use a fixed, randomly connected network as a reservoir, but they are typically implemented with non-spiking neurons. Recent research has focused on adapting the ESN concept to spiking neurons to enhance their ability to process temporal information in a biologically plausible way [16]. These networks are effective for tasks that require temporal sequence learning and can be employed in situations where training recurrent weights is infeasible or undesirable due to resource constraints.

Applications and Comparisons Each SNN architecture offers unique strengths, and the choice of architecture largely depends on the nature of the task. For example:

- Feedforward SNNs are well-suited for tasks involving static data, where temporal relationships are not critical.
- Recurrent SNNs and LSMs are ideal for dynamic tasks requiring memory and temporal pattern recognition, such as speech and gesture recognition.

- Convolutional SNNs excel in handling visual data that contain both spatial and temporal features, making them suitable for vision-based tasks like video analysis and object tracking.

Neuromorphic hardware platforms like Intel's Loihi and IBM's TrueNorth provide the ideal environment to implement these architectures efficiently, leveraging the event-driven and sparse nature of SNNs to achieve low-power, high-speed computation [6, 12]. These platforms support large-scale deployment of SNN architectures and are optimized to fully exploit the temporal dynamics and event-driven characteristics of spiking neurons, making them highly suitable for real-world applications in robotics, autonomous vehicles, and embedded intelligent systems. In Figure 2.9 we report a scheme of the TrueNorth architecture.

2.1.4 Applications of SNNs

The unique properties of SNNs make them suitable for a wide range of applications, particularly in neuromorphic computing and real-time sensory processing. SNNs excel in applications that benefit from their energy efficiency, temporal coding, and event-driven processing capabilities. The following subsections discuss the application areas of SNNs in terms of real-time processing and embedded/edge computing environments.

Real-Time Processing

SNNs have been used successfully in dynamic vision systems for real-time object detection and classification, exploiting their ability to process spatio-temporal data efficiently [5, 19]. Dynamic vision sensing, such as that used in event-based cameras, relies heavily on the precise temporal dynamics that SNNs can naturally model. By responding only to changes in the visual field, SNNs provide a power-efficient and highly responsive solution for tasks such as object tracking, motion detection, and scene understanding in autonomous systems. These properties make SNNs an ideal choice for applications that require high-speed processing with minimal latency.

Another area where SNNs have demonstrated significant success is radar-based gesture recognition. Due to their temporal coding capabilities, SNNs are able to capture the intricate temporal patterns associated with different gestures, allowing for accurate and low-latency recognition, making SNNs ideal for real-time human-computer interaction, particularly in scenarios where responsiveness is critical, such as wearable devices and augmeanted reality systems [8, 19].

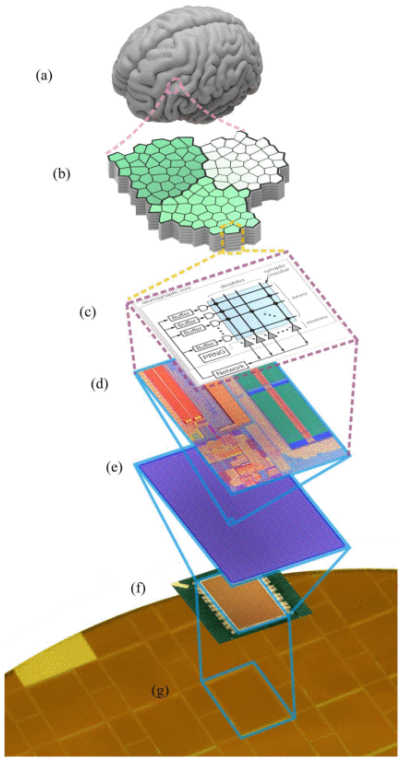


Figure 2.9: The TrueNorth architecture draws inspiration from (a) the human brain and (b) the cortical column, a small unit consisting of densely interconnected neurons with similar functionality. The architecture's key element, the (c) neurosynaptic core, includes 256 input axons, 256 neurons, and a 64k synaptic crossbar, integrating memory, computation, and communication within a compact area of 240×390 μm of (d) silicon. The integration of 4096 neurosynaptic cores in a 2-D grid constitutes (e) the TrueNorth chip, with an area of 4.3 cm^2, manufactured on a (f) 28 nm low-power CMOS process, consuming only 65 mW during typical computer vision applications. This design effectively approximates cortical structure using an (g) inorganic silicon substrate. Source: [12].

Beyond gesture and vision applications, SNNs have shown promise in speech processing and auditory signal analysis. The temporal resolution of SNNs allows them to encode auditory signals with high precision, making them suitable for applications such as speech recognition, speaker identification, and auditory scene analysis. The ability to process both rate-coded and temporally-coded information makes SNNs well-suited for the complex, time-varying nature of auditory data [5]. The inherent capability of SNNs to handle real-time data makes them a strong candidate for various sensory processing applications where traditional ANN architectures often struggle to capture the temporal characteristics effectively.

Embedded and Edge Applications

SNNs are particularly advantageous for deployment in resource-constrained environments, such as mobile devices and edge computing platforms. Their power efficiency and adaptability make them a natural choice for implementing real-time AI solutions that require on-device processing without the support of extensive computational resources [19]. Neuromorphic hardware platforms like Intel's Loihi and IBM's TrueNorth are specifically designed to leverage the sparse, event-driven nature of SNNs, enabling the efficient execution of large-scale networks

in applications such as environmental monitoring, robotics, and intelligent sensor networks.

In robotics, SNNs are used for motor control, decision-making, and sensory integration. The event-driven nature of SNNs allows robots to react quickly to changes in their environment, while their ability to integrate sensory information from multiple modalities makes them suitable for complex navigation and manipulation tasks. For instance, SNNs can be used in robotic arms to dynamically adjust grip based on sensory feedback, or in autonomous drones for real-time path planning and obstacle avoidance [15]. The low-power, high-efficiency operation of SNNs makes them an attractive solution for robotics applications, particularly in autonomous and mobile robots that require long battery life.

The role of SNNs in neuromorphic computing extends beyond robotics. SNNs represent a cornerstone technology for building energy-efficient, brain-inspired computing systems. By mimicking the sparse, asynchronous nature of biological neural networks, SNNs offer a pathway to developing computational models that not only perform complex tasks but also do so with minimal power consumption. This makes them an attractive option for next-generation AI systems, where traditional neural networks are often limited by their power requirements and lack of scalability. Neuromorphic hardware that implements SNNs is designed to optimize computation by emulating biological processes, providing robust solutions for tasks such as anomaly detection in sensor networks, adaptive control in smart environments, and real-time decision making at the edge.

2.2 Related Works

Recent advancements in neuromorphic computing and SNNs have demonstrated substantial promise in the areas of energy-efficient computing and biologically plausible AI architectures. A wide variety of methodologies and frameworks have been proposed to enhance the functionality of SNNs for real-time applications, leveraging both biological insights and advancements in neural computation. This section reviews the recent developments in SNN learning mechanisms, Liquid State Machines, and neuromorphic hardware implementations that drive these advancements.

2.2.1 Learning Mechanisms in Spiking Neural Networks

Spike-Timing-Dependent Plasticity and its reward-modulated variant have been effective in training SNNs, facilitating robust learning capabilities that are biologically plausible. Moza-

fari et al. (2018a) [2] proposed an R-STDP-based visual categorization model that utilized first-spike latencies to achieve improved energy efficiency, avoiding the computationally expensive backpropagation techniques typically seen in ANNs. This method aligned closely with the hardware constraints of neuromorphic platforms, as it reduced the need for multiply-accumulate (MAC) operations, which are energy-intensive. The visual categorization approach by Mozafari et al. was later extended in Mozafari et al. (2018b) [3], where the authors compared R-STDP to standard STDP for feature extraction in convolutional SNNs, concluding that R-STDP offered enhanced efficiency with fewer computational resources.

Fremaux and Gerstner (2015) [1] provided a detailed overview of learning mechanisms for SNNs, focusing on the biological plausibility and effectiveness of three-factor learning rules like R-STDP. These rules combine pre- and post-synaptic spike timing with a global reward signal, making them ideal for reinforcement learning scenarios. Similarly, Legenstein and Maass (2008) [21] proposed a theoretical framework for reward-modulated spike-timing-dependent plasticity, further establishing the significance of R-STDP in real-time neural computation, particularly in applications such as biofeedback systems.
Pfeiffer and Pfeil (2018) [22] reviewed the opportunities and challenges of deep learning with spiking neurons, providing insight into how biologically inspired spike-based computation can offer energy-efficient alternatives to traditional deep learning models. The authors emphasized the integration of SNNs in neuromorphic hardware, which allows for on-chip learning with a significant reduction in power consumption. The use of surrogate gradient methods for training deep SNNs has also been proposed to overcome the challenges associated with backpropagation in spiking networks, allowing for effective learning in multi-layered architectures [5].

2.2.2 Liquid State Machines and Computational Models

Liquid State Machines represent a significant computational model for real-time data processing in SNNs. Maass et al. (2002) [15] introduced LSMs as a framework capable of transforming temporal data into a high-dimensional representation, which can then be used for complex pattern recognition tasks. This model was particularly valuable for applications involving dynamic inputs, such as speech and gesture recognition. In a subsequent study, Maass et al. (2011) [16] further explored the computational power of LSMs, demonstrating their ability to handle nonlinear temporal transformations without requiring stable states, thus mimicking the flexible information processing capabilities of biological brains.

Joshi and Maass (2004) [23] studied the "edge-of-chaos" behavior in LSMs, showing that

their computational power is maximized near the edge of stability, which is akin to the criticality observed in biological neural circuits. The significance of this behavior has been validated in subsequent studies that confirmed the edge-of-chaos hypothesis in both artificial and biological networks.

Biswas et al. (2023) [8] proposed the MAdapter, an LSM-based multimodal processing architecture capable of adapting to different types of input modalities, including vision and speech. By using a shared reservoir for different modalities, the model demonstrated high classification accuracy without requiring dataset-specific hyperparameter tuning, making it highly efficient for real-world applications with varying input types.

2.2.3 Neuromorphic Hardware Implementations

The advent of neuromorphic hardware, such as Intel's Loihi and IBM's TrueNorth, has significantly enhanced the practical deployment of SNNs. IBM's TrueNorth, introduced by Merolla et al. (2014) [24], was inspired by the structure of the human brain and cortical columns, integrating computation, memory, and communication in a compact architecture. Each neurosynaptic core within TrueNorth contains 256 axons, 256 neurons, and a 64k synaptic crossbar, enabling efficient parallel processing with minimal power consumption. This architecture was instrumental in pushing the boundaries of hardware efficiency, as it consumes only 65 mW of power while performing computer vision tasks [24].

Nunes et al. (2022) [5] reviewed the integration of LSMs and other recurrent SNN architectures with neuromorphic hardware, emphasizing their suitability for applications that require dynamic, spatiotemporal processing. The Loihi chip, introduced by Davies et al. (2018) [6], provided on-chip learning capabilities using local plasticity rules, such as STDP, and featured a flexible mesh network of neuromorphic cores. These characteristics allowed the implementation of real-time adaptive learning, crucial for edge and autonomous applications where traditional machine learning techniques often fall short due to latency and power constraints.

Izhikevich (2003) [25] introduced a simple yet powerful spiking neuron model that serves as the basis for many neuromorphic hardware implementations. The Izhikevich model combines biological accuracy with computational efficiency, making it highly suitable for large-scale neuromorphic platforms, as later demonstrated in Loihi and TrueNorth. The versatility of these models has enabled the use of neuromorphic chips in applications ranging from robotics to environmental monitoring, where real-time responsiveness and low power consumption are critical.

2.2.4 Spiking Neural Networks in Real-World Applications

SNNs have shown great promise in a wide range of real-world applications, particularly in resource-constrained environments such as mobile devices and edge computing. Tsang et al. (2021) [19] explored the application of SNNs in edge computing platforms, highlighting their suitability for low-power environments. They discussed the deployment of SNNs for tasks such as environmental monitoring and robotics, leveraging the event-driven and sparse activation characteristics of spiking neurons to achieve efficient real-time processing.

2.2.5 Comparison with Traditional Approaches

The versatility of SNNs, particularly when combined with neuromorphic hardware, provides distinct advantages over traditional deep learning models. Pfeiffer and Pfeil (2018) [22] highlighted that, unlike ANNs, which require high power and extensive computational resources, SNNs operate efficiently by employing sparse, event-driven computation. This makes them particularly advantageous in real-time applications, where both power efficiency and adaptability are crucial.
Recent works, such as Mozafari et al. (2018b) [3], have compared convolutional SNNs with traditional CNNs, showing that SNNs can achieve similar levels of accuracy while drastically reducing power consumption. The combination of biological plausibility, power efficiency, and suitability for neuromorphic hardware makes SNNs a promising alternative to traditional deep learning models for next-generation AI systems.

The advancements in neuromorphic hardware and SNN frameworks underscore the potential of spiking neurons to revolutionize real-time computing. From biologically inspired learning mechanisms like R-STDP to versatile computational models such as LSMs, SNNs are paving the way for energy-efficient and adaptive AI systems. The development of neuromorphic hardware platforms like TrueNorth and Loihi has further expanded the practical applications of SNNs, offering powerful tools for deploying low-power, high-performance neural networks in real-world scenarios.

3 Methodology

This section outlines the methodologies employed to investigate and extend the capabilities of SNNs. The approach began with reproducing results from key literature to establish a solid foundation, followed by exploring various extensions aimed at enhancing network depth and complexity.

3.1 Overview

The initial phase of this investigation involved replicating the results presented by Mozafari et al. in their studies [2, 3], utilizing the open-source library `SpykeTorch`* (Figure 3.1(a)) as detailed in [13]. This framework, implemented in Python, allowed for the development of SNNs with a focus on biologically plausible synaptic plasticity mechanisms such as STDP-based learning. Reproducing these results not only provided a solid baseline for further exploration but also validated the experimental setup. All experiments were conducted in Python, and the corresponding code is available on GitHub†.

To expand the scope of the investigation, `snnTorch`‡ (Figure 3.1(b)), proposed in [26], was employed for experiments involving surrogate gradient methods. Unlike `SpykeTorch`, which was tailored for STDP-centric models, `snnTorch` enabled the exploration of more complex ar-

*`SpykeTorch` documentation can be accessed at `https://github.com/miladmozafari/SpykeTorch`
†The full code can be accessed at `https://github.com/aidinattar/snn`.
‡`snnTorch` documentation can be accessed at `https://snntorch.readthedocs.io/`

chitectures by approximating gradient descent for spike-based learning. Both libraries are built on the `PyTorch` framework, ensuring efficient computation and integration with additional tools. Supporting libraries such as `NumPy` and `Matplotlib` were also used for numerical operations and visualizations, respectively.

(a) `SpykeTorch` Logo. **(b)** `snnTorch` Logo.

Figure 3.1: Logo of the main libraries used in the development of the models used.

Building on this foundation, the next phase focused on extending the depth and capabilities of SNN models. This involved incorporating additional layers into the network and experimenting with different training methodologies, specifically STDP and R-STDP. These approaches were evaluated for their effectiveness in enhancing the network's ability to learn complex features. Additionally, population coding strategies were explored, leveraging majority voting mechanisms to increase the robustness and accuracy of information representation within the network.

In parallel, we implemented versions of similar networks that utilized surrogate gradient methods instead of STDP. The surrogate gradient approach allowed us to approximate gradient-based optimization, enabling the training of deeper spiking networks more effectively. This method was crucial in overcoming the limitations posed by the non-differentiable nature of spike-based activation functions, thereby allowing for the application of backpropagation-like techniques in SNNs.

To further improve network performance, concepts from traditional neural networks were adapted and integrated into the SNN architecture. In particular, inception blocks were employed to capture multi-scale features concurrently, while skip connections were introduced to facilitate gradient flow and mitigate the vanishing spike problem, thereby stabilizing the training process. These architectural innovations were intended to bring the performance of SNNs closer to that of modern deep learning models, combining the temporal efficiency of SNNs with advanced feature extraction capabilities.

Finally, the research explored the implementation of LSMs, a form of reservoir computing that is particularly well-suited for handling temporal pattern recognition tasks. LSMs were used to evaluate the capacity of SNNs to process time-varying inputs and generate high-dimensional

projections suitable for subsequent classification and decision-making tasks. This systematic progression—from reproducing established models to incorporating novel deep learning techniques — illustrates a comprehensive approach aimed at advancing the capabilities of SNNs in both depth and versatility.

3.2 Dataset and preprocessing

The datasets utilized in this study include MNIST [27], NORB [28], CIFAR-10 [29], a reduced version of Caltech [30] consisting only of the *face* and *motorbike* categories, and the Neuromorphic-MNIST (N-MNIST) dataset. In addition the Google's SOLI [31] dataset was used in the training of the LSM. These datasets were selected to provide a diverse range of visual challenges, encompassing simple handwritten digits, natural object recognition, radar-based gestures, and dynamic visual stimuli, thereby enabling a comprehensive evaluation of the SNN models across different tasks. In Figure 3.2 some samples of the datasets used.

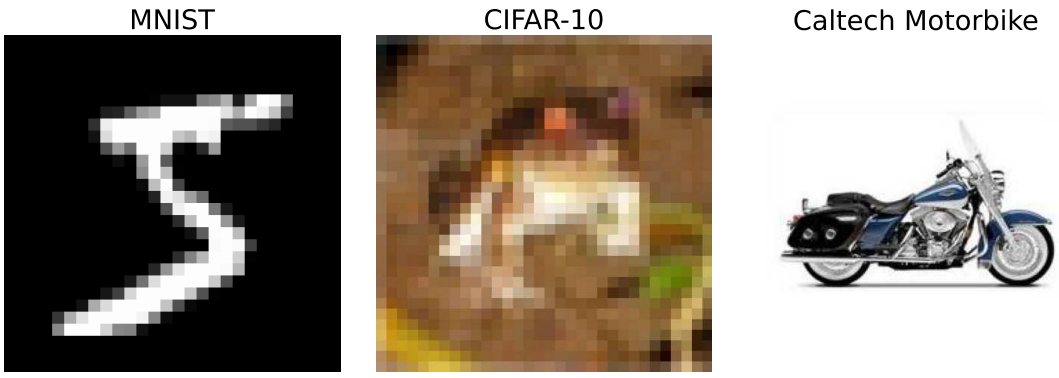


Figure 3.2: Example images from the datasets used in the experiments. The datasets include MNIST, CIFAR-10, and Caltech Face/Motorbike, each presenting distinct visual challenges and complexities for the classification tasks.

The N-MNIST dataset [32] is inherently a neuromorphic dataset, captured using an event-based camera to generate spiking data directly from dynamic visual scenes. This makes it a natural choice for evaluating the temporal processing capabilities of SNNs without requiring additional preprocessing. The N-MNIST dataset used in this study was obtained from the `Tonic` library[§] in `Python`, which provides convenient access to neuromorphic datasets. For the remaining datasets—MNIST, NORB, CIFAR-10, and the reduced version of Caltech—a preprocessing pipeline was applied to convert conventional image data into a suitable spike-

[§]The library can be accessed at `https://tonic.readthedocs.io/`

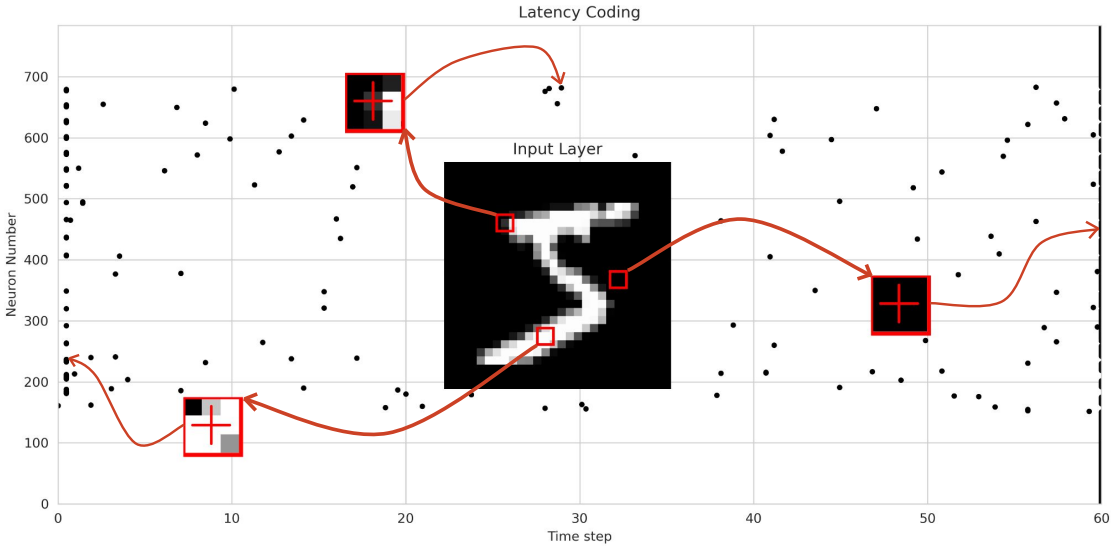
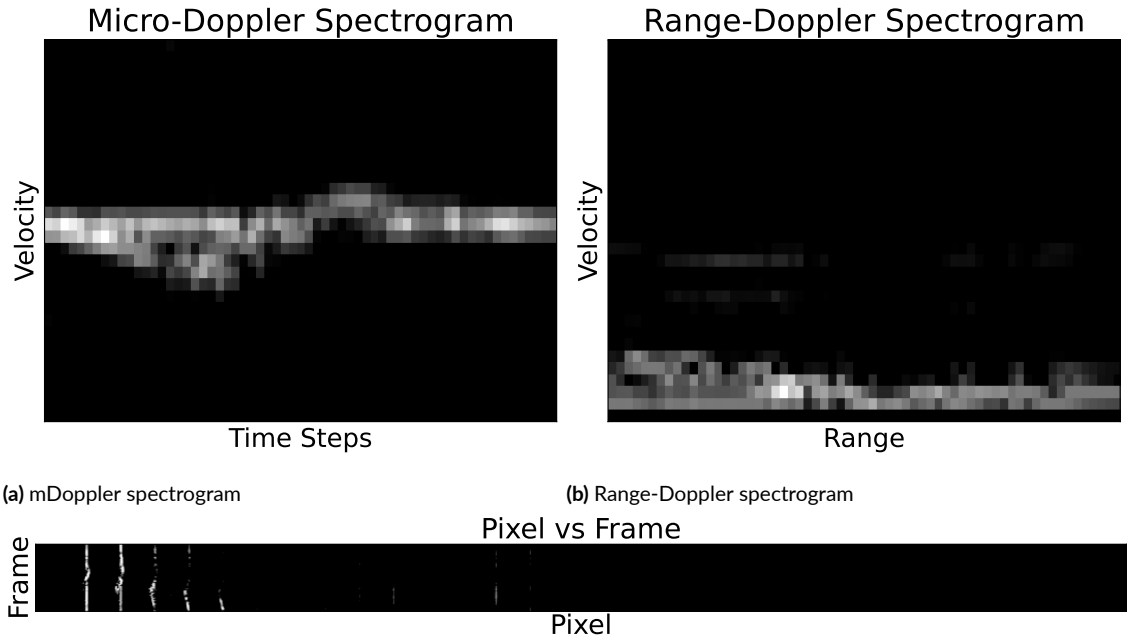


Figure 3.3: Illustration of Latency Coding in SNNs using Intensity-to-Latency Encoding. The central image represents an MNIST-like input sample, with three red-highlighted pixels. The zoomed-in boxes show enlarged regions of these selected pixels. Pixel intensity is inversely mapped to spike latency, meaning that brighter pixels (higher intensity) trigger spikes earlier in time. The scatter plot on the left displays the spike timings for all pixels, where the vertical axis represents individual neurons and the horizontal axis represents spike time steps.

based representation. The first step involved convolving the input images with Difference of Gaussian (DoG) filters at various scales to enhance edges and salient features. This filtering operation serves to highlight critical spatial information, mimicking the initial visual processing that occurs in biological vision systems. Following the DoG filtering, a temporal encoding scheme, specifically an intensity-to-latency encoding, was employed to convert the filtered images into spike trains. Using this encoding scheme, the obtained feature maps were converted into spike latencies that were inversely proportional to the saliency of the edges. In other words, the more salient the edge, the earlier the corresponding spike was propagated. This approach, as shown in Figure 3.3, generates a spike wave that captures the spatial structure of the input image while maintaining a temporal sequence that can be processed effectively by the SNN.

The intensity-to-latency encoding used in this study involved dividing the input into a series of time steps. During each time step, spikes were generated based on the intensity values, ensuring that regions with higher intensity values were represented earlier in the spike sequence. This approach provides a biologically inspired mechanism for encoding visual information, enabling the SNN to process spatio-temporal patterns in a manner similar to how biological neurons respond to stimuli. The generated spike wave was then propagated to the input layer of



(a) mDoppler spectrogram

(b) Range-Doppler spectrogram

(c) Pixel vs Frame representation

Figure 3.4: Visualization of Micro-Doppler and Range-Doppler Spectrograms for Hand Gesture Recognition. **(a)** micro-Doppler spectrogram, which captures velocity variations over time, highlighting the temporal evolution of a gesture. **(b)** range-Doppler spectrogram, representing the distribution of reflected radar signals across both range and velocity. **(c)** pixel vs. frame matrix, constructed by unrolling and stacking frames horizontally, providing a comprehensive view of the gesture sequence over time.

the SNN, allowing the network to leverage both spatial and temporal dynamics effectively for feature extraction and learning. This preprocessing approach ensures that the input data is appropriately transformed to take advantage of the temporal processing capabilities of SNNs, enabling effective representation and learning of both static and dynamic visual information.

The Soli dataset, collected using Google's 60 GHz FMCW millimeter-wave radar sensor [31] [33], was developed to enable human-computer interaction through radar-based sensing of micro-interactions. This dataset comprises 2D range-Doppler images of size 32×32, capturing 11 different hand gestures performed by ten users. Each user performed 25 instances of each gesture, leading to a total of 2750 samples. This dataset allows for the exploration of radar-based gesture recognition techniques. In Figure 3.4, visual illustrations of sample data from the Soli dataset are presented.

3.3 Network architecture design

The design of the network architectures in this study draws upon two foundational models presented in Mozafari et al.'s work, which serve as the starting point for further development and experimentation. These baseline models were selected due to their demonstrated effectiveness in SNN research, providing a solid foundation for evaluating and extending the capabilities of SNN architectures. A full description of our proposed models is provided in Table 3.10.

3.3.1 Baseline Model A: Shallow Convolutional SNN

The first baseline model, as described in Mozafari et al. (2018a) [2], is a shallow convolutional SNN designed for visual object recognition tasks, employing a temporal coding scheme where neurons fire based on the saliency of input features. The network is structured with multiple layers, starting from feature extraction to categorization based on first-spike latencies. A schematic representation of the network is shown in Figure 3.5.

Layer	Name	Description
S1	**Feature Extraction Layer**	Utilizes Gabor filters w. four different orientations to detect edges in input images. Outputs are encoded into spike latencies.
C1	**Pooling Layer**	Applies pooling to introduce invariance to small shifts. Neurons propagate the earliest spike in their receptive field, enhancing computational efficiency by reducing redundancy.
S2	**Decision-Making Layer**	Consists of IF neurons. Synaptic plasticity is governed by R-STDP, encouraging feature selectivity based on decision correctness.
C2	**Output Layer**	Contains neurons associated with different categories. The network's final decision is based on the first neuron in C2 to emit a spike, indicating the recognized category.

Table 3.1: Summary of the architecture of the shallow CNN and function of each layer.

A description of the architecture is provided in Table 3.1. The second complex layer (C2) in Mozafari et al.'s network is the decision-making layer, consisting of exactly n neurons, each associated with one of the neuronal grids in layer S2. The neurons in layer C2 perform a global

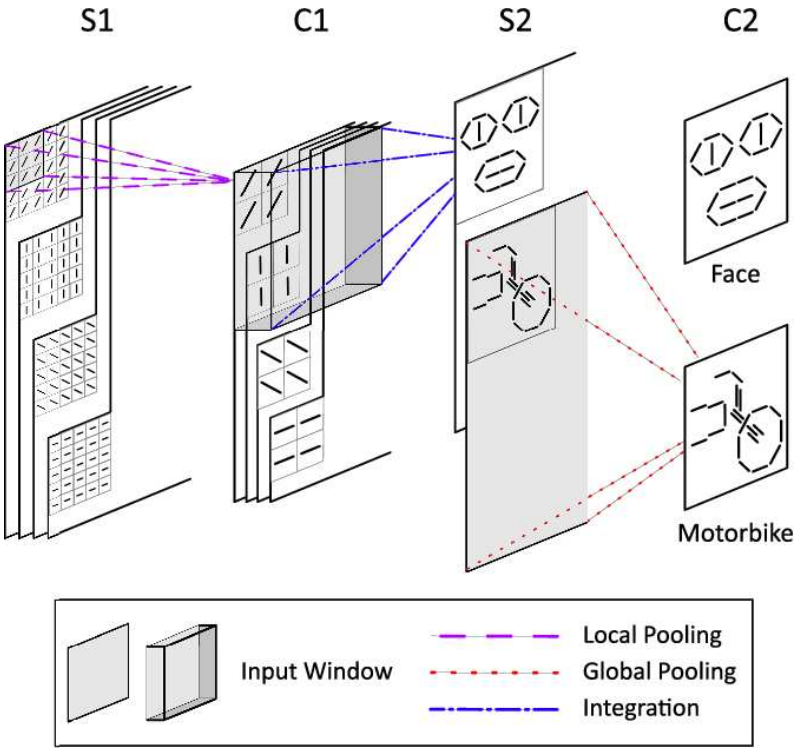


Figure 3.5: Overall structure of the proposed network with four retinotopically organized layers. The first layer (S1) extracts oriented edges from the input image by applying Gabor filters. A local max-pooling operation is applied by the cells in the subsequent layer (C1) to gain some degrees of position invariance. From here, spikes are propagated by the latencies that are inversely proportional to the maximum values. These spikes are the inputs for the IF neurons in the layer S2 that are equipped with the R-STDP learning rule. These neurons are encouraged/punished to learn/unlearn complex features. The activity of S2 neurons is used by C2 neurons for decision-making. These neurons are associated with class labels and the decision is made based on the neuron with the earliest spike [2].

pooling operation over the neurons in the respective S2 grid. Specifically, each C2 neuron only propagates the first spike received from the associated S2 neurons, effectively representing the minimum firing time from that group. Mathematically, the firing time of the i-th neuron in layer C2 is defined as:

$$t^f_{c2(i)} = \min_{j \in P_{c2}(i)} \left(t^f_{s2(j)} \right) \tag{3.1}$$

where $P_{c2}(i)$ is the set of all presynaptic neurons from layer S2 connected to the i-th neuron in layer C2, and $t^f_{s2(j)}$ represents the firing time of the j-th neuron in layer S2.

The decision of the network is made by dividing the neurons in C2 into several groups, with each group corresponding to a specific category of input stimuli. The network determines the category of the input by selecting the category whose neurons in C2 fire the earliest. Assuming

there are m distinct categories for input stimuli labeled from 1 to m, and n neuronal grids in layer S2, the neurons in C2 are divided into m groups. The decision D of the network is then given by:

$$F = \arg\min_i \left(t^f_{c2(i)} \mid 1 \leq i \leq n \right) \tag{3.2}$$

$$D = g(F) \tag{3.3}$$

where $g : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, m\}$ is a function that returns the group index of the neuron i, and $t^f_{c2(i)}$ represents the firing time of the i-th neuron in layer C2. The group corresponding to the earliest spike is assumed to be the correct classification for the given input.

3.3.2 Baseline Model B: Deep Convolutional SNN

The second baseline model, detailed in Mozafari et al. (2018b) [3], is a Deep Convolutional Spiking Neural Network (DCSNN) that integrates convolutional operations to enhance feature extraction, particularly for visual tasks with complex spatial structures. A schematic representation of the network is shown in Figure 3.6. The model utilizes both STDP and R-STDP learning rules across different layers to achieve efficient, biologically plausible learning.

In this architecture, summarized in Table 3.2, the lower convolutional layers (S1, S2) are trained using STDP, which enables the network to learn frequently occurring features. In contrast, the higher layer (S3) is trained with R-STDP, which reinforces features based on their relevance to the task, allowing the network to focus on features that are diagnostic for classification.

3.3.3 Extended DCSNN with Additional Convolutional Layer

To further enhance the capabilities of the baseline DCSNN, an additional convolutional layer was incorporated, which could be trained either with STDP or R-STDP. This extension aimed to evaluate the impact of deeper architectures on the ability of SNNs to learn more abstract and high-level features.

The network is described layer by layer in Table 3.3. In the final C4 layer, neurons are assigned to specific input categories, with each neuron representing a different class label. The decision of the network is determined based on either the earliest spike emitted by a neuron or the maximum internal potential, depending on the pooling layer configuration. A decision map is generated by comparing the internal potentials or spike times across all C4 neurons, and

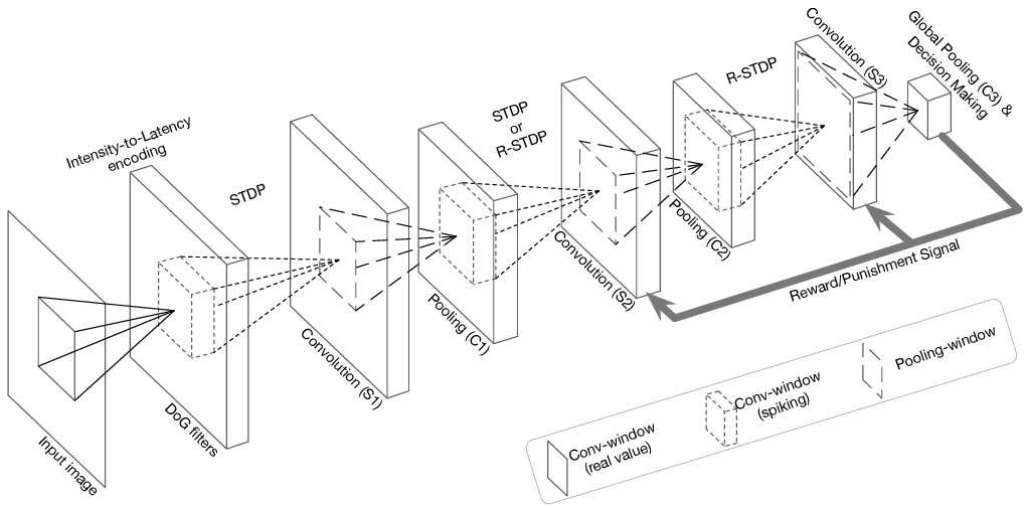


Figure 3.6: Overall structure of the proposed DCSNN for digit recognition. The input image is convolved by six DoG filters (on- and off-center in three different scales) and converted into spike latencies based on intensity-to-latency encoding scheme. Generated spikes are processed through three spiking convolution-then-pooling layers (S1 – C1,S2 – C2, and S3 – C3). All of the spiking convolution layers are trainable, employing either STDP or R-STDP learning rules. The network makes its decision in C3, where neurons are pre-assigned to digits, and the decision is the digit assigned to the neuron with either the maximum internal potential or the earliest spike time. Regarding the decision, a reward/punishment (reinforcement) signal will be generated globally, which is received by R-STDP-enabled layers. [3].

the class label corresponding to the neuron with the highest potential or earliest spike is selected as the network's output. If the decision matches the actual label, a reward signal is generated; otherwise, a punishment signal is applied. This reinforcement signal modulates synaptic plasticity in R-STDP-enabled layers, enhancing the network's ability to make accurate classifications.

The inclusion of the additional convolutional layer allows the network to delve deeper into the hierarchical representation of the input data, potentially capturing more abstract features that are crucial for complex visual recognition tasks. By comparing the performance of the model with S_3 trained using STDP versus R-STDP, insights can be gained into how different learning mechanisms influence the network's ability to generalize and effectively represent high-level features.

3.3.4 Population Coding-Based Spiking Neural Network

Population coding is a neural representation strategy in which information is encoded by the activity of a group of neurons rather than by individual neurons. In biological systems, this approach is particularly useful for representing continuous variables, such as spatial orientation

Layer	Name	Description
	Input Preprocessing	Convolution with DoG filters at various scales to enhance edge features. Enhanced images are then encoded using an intensity-to-latency scheme.
S1	**Convolution Layer**	Extracts local features by convolving the input spike maps with learnable filters.
C1	**Pooling Layer**	Reduces spatial dimensions while retaining critical information, helping the network achieve translation invariance.
S2	**Convolution Layer**	Further processes the features, extracting more complex patterns.
C2	**Pooling Layer**	Reduces the feature map size, preserving the most salient features for higher-level processing.
S3	**Convolution Layer**	Integrates features extracted by previous layers.
C3	**Global Pooling Layer**	Aggregates the information across all spatial locations. Neurons in C3 are assigned to specific categories, and the decision of the network is based on the maximum internal potential or the earliest spike emitted by these neurons, depending on the pooling strategy employed [3].

Table 3.2: Summary of the architecture and function of each layer in the DCSNN network.

or motor control, by aggregating responses across a population of neurons. In the context of SNNs, population coding allows for the redundancy and robustness of information processing, as multiple neurons collectively contribute to the encoding of particular features, making the network less susceptible to noise and variations in individual neuron behaviour.

In this study, a population coding-based spiking neural network model was developed, inspired by the model presented in section 3.3.2. The architecture of this model is described in details in Table 3.4. It is similar to the DCSNN from the baseline, but with modifications in the decision-making layer to leverage the advantages of population coding. Specifically, the final layer of the model has been restructured to include three parallel branches, each ending in a decision layer that performs convolution operations with different kernels.

Mathematically, let $D_{i\in\{1,2,3\}}$ represent the decisions made by each of the three branches, with each decision based on the earliest spike time of the neurons in the corresponding decision map. The final decision D of the network is given by:

Layer	Name	Description
	Input Preprocessing	Convolution with DoG filters and encoded into spike latencies using an intensity-to-latency scheme.
S1	**Convolution Layer**	Extracts local features by convolving the input spike maps with learnable filters.
C1	**Pooling Layer**	Reduces spatial dimensions while retaining critical information, achieving translation invariance.
S2	**Convolution Layer**	Extracts more complex features, building upon those obtained in S1.
C2	**Pooling Layer**	Further reduces feature map dimensions, keeping only the most salient information.
S3	**Convolution Layer**	Integrates features extracted by the preceding layers for higher-level abstraction.
C3	**Pooling Layer**	Aggregates information across all spatial locations, aiding in the classification decision.
S4	**Additional Convolution Layer**	Refines feature representations by learning more specialized patterns.
C4	**Global Pooling Layer**	Aggregates critical information. Depending on the training scenario, S4 can be trained with either STDP or R-STDP, each influencing feature abstraction differently.

Table 3.3: Summary of the extended architecture and the function of each layer of the extended DCSNN.

$$D = \text{mode}(D_1, D_2, D_3) \tag{3.4}$$

where mode$(\cdot)$ represents the majority voting function that selects the class label that occurs most frequently among the three branches.

The addition of the three-branch decision layer allows the network to leverage population coding by aggregating information from different spatial scales, leading to a more comprehensive and resilient classification process. This approach aims to enhance the network's ability to generalize across various visual tasks, effectively capturing both fine-grained and large-scale spatial features through the parallel decision pathways.

3.3.5 Inception-Like Spiking Neural Network

In classical deep learning, inception blocks are architectural elements designed to improve feature extraction by employing multiple convolutional operations with different kernel sizes in

Layer	Name	Description
	Input Preprocessing	Convolution with DoG filters and encoding using an intensity-to-latency scheme.
S1	**Convolution Layer**	Extracts local features from the input spike maps using learnable filters.
C1	**Pooling Layer**	Reduces the dimensionality while retaining critical spatial information.
S2	**Convolution Layer**	Continues to extract more complex features.
C2	**Pooling Layer**	Reduces feature map dimensions and helps achieve translation invariance.
S3	**Branching - Convolution Layers**	Three parallel convolutional layers, each employing different kernel sizes (small, medium, large) to capture fine, intermediate, and broad spatial features, respectively.
C3	**Decision making - Majority Voting**	Decision by majority voting across branches, enhancing robustness by leveraging multi-scale information and reducing the likelihood of incorrect classification.

Table 3.4: Summary of the architecture with a three-branch decision-making layer, detailing the convolution and pooling layers, as well as the decision-making strategy.

parallel. The inception module allows the network to capture features at various scales, enhancing its ability to recognize complex patterns in the input data. By using different kernel sizes, the inception block ensures that both fine-grained and broad contextual information are processed simultaneously, making the network more robust and versatile in its representation capabilities.

Inspired by the inception concept, this study introduces an inception-like SNN model that incorporates branching and multi-scale feature extraction into a spiking architecture. The model extends the DCSNN presented in 3.3.3 by including a branching structure in the penultimate layer, which captures different spatial features at multiple scales. The spiking maps from these branches are then concatenated before being processed by the final layer, enabling a richer representation of the input data.

The architecture of the intecption-like SNN is specified in Table 3.5. Mathematically, let $X_{i\in\{1,2,3\}}$ represent the feature maps generated by the three branches in layer S3, where each X_i corresponds to a different kernel size. The concatenated feature map X_{concat} is given by:

$$X_{\text{concat}} = [X_1, X_2, X_3] \tag{3.5}$$

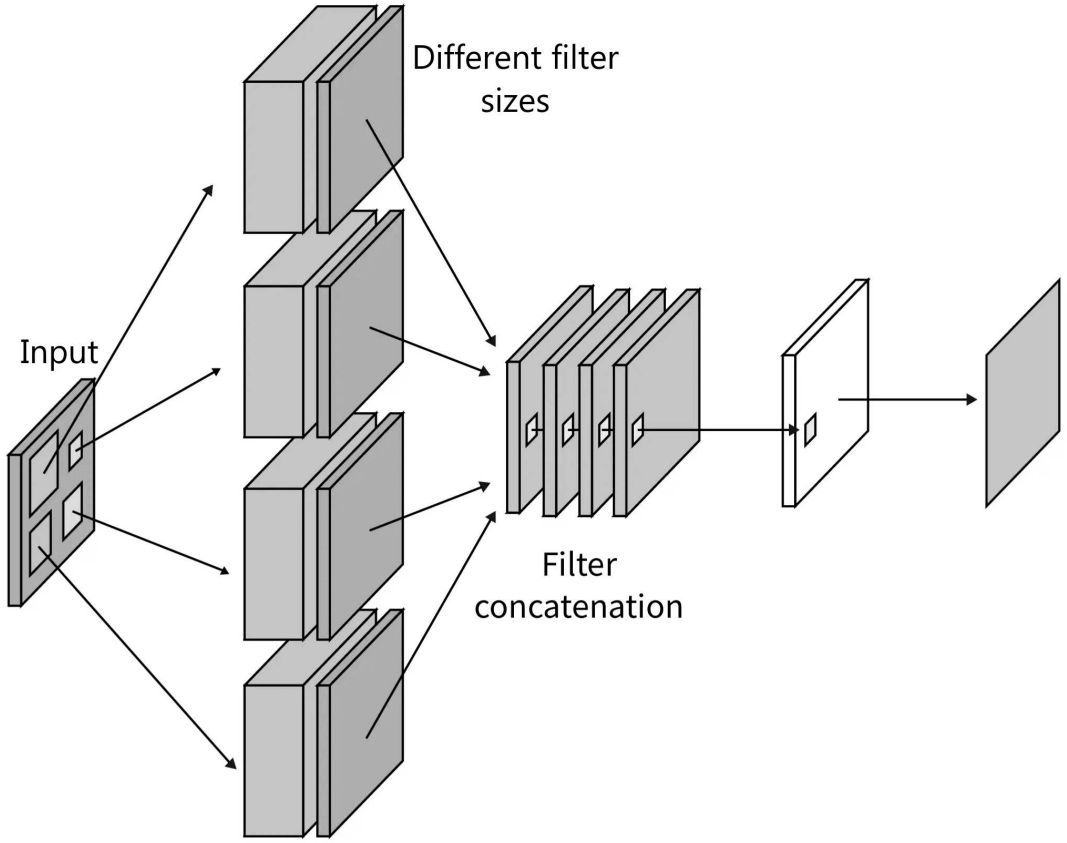


Figure 3.7: Schematic representation of the inception block structure used.

where $[X_1, X_2, X_3]$ denotes the concatenation operation along the channel dimension. The final convolutional layer (C_4) processes X_{concat} to generate the output spike trains, which are then used for classification. A visualization of the inception block is reported in Figure 3.7.

The inception-like structure introduced in this model allows the network to leverage multi-scale feature extraction by processing the input data through different convolutional kernels in parallel. This approach enriches the feature representation and improves the network's ability to recognize patterns at different levels of granularity. By concatenating the spiking maps from each branch, the network gains a more holistic view of the input, ultimately enhancing its classification accuracy and robustness in handling complex visual tasks.

Layer	Name	Description
	Input Preprocessing	Convolution with DoG filters and intensity-to-latency encoding scheme.
S1	**Convolution Layer**	Extracts local features from the input spike maps using learnable filters.
C1	**Pooling Layer**	Reduces the dimensionality of the feature maps while retaining important spatial information.
S2	**Convolution Layer**	Extracts more complex features from the input data.
C2	**Pooling Layer**	Reduces feature map size, ensuring translation invariance.
S3-Branch	**Branching Convolutional Layers**	The network splits into three parallel convolutional pathways with distinct kernel sizes to capture features at different scales: small for fine details, medium for intermediate features, and large for broader context. The resulting feature maps are concatenated along the channel dimension.
C4	**Final Convolutional Layer**	Concatenated feature maps are processed to integrate the multi-scale information obtained from the branching pathways.
S4	**Output Layer**	Determines the class label based on the spiking activity of the neurons in the final convolutional layer.

Table 3.5: Summary of the inception-like architecture with branching convolutional layers, capturing the multi-scale feature extraction approach and classification strategy.

3.3.6 Residual Spiking Neural Network

In classical deep learning, residual networks (ResNets) introduced skip connections, with the scheme reported in Figure 3.8, to mitigate the problem of vanishing gradients and to ease the training of deep networks by enabling a more effective flow of information through layers. By adding skip connections that bypass certain layers, ResNets facilitate the learning of residual functions, allowing gradients to be propagated more directly and improving the network's ability to learn deep representations. This concept has been adapted to SNNs in the current study, with the aim of preserving spike timings and minimizing signal degradation, in line with the critical brain hypothesis, which suggests that neural systems may operate at a critical state for optimal computational efficiency.

The residual SNN model developed in this study is described in details in Table 3.6. It consists of four layers, with a skip connection from the output of the second layer to the output of the third layer. This architecture aims to preserve temporal information by directly propagating spikes from an earlier layer, thereby potentially enhancing the ability of the network to retain relevant features over time and improving the stability of the spiking signals.

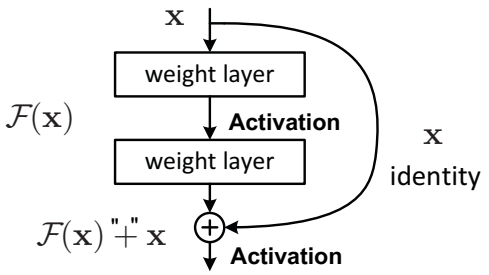


Figure 3.8: Illustration of a residual block structure in a neural network, showing the skip connection that allows the input x to bypass the weight layers and combine with the output $F(x)$. Source: [34].

Layer	Name	Description
	Input Preprocessing	Convolution with DoG filters and encoded using an intensity-to-latency scheme.
S1	**Convolution Layer**	Extracts local features from the input spike maps using a set of learnable filters.
S2	**Convolution Layer**	Refines the features learned in S1 to extract more complex spatial and temporal patterns.
S3	**Convolution Layer**	Processes the spiking map from S2, integrating the bypassed information from the skip connection to maintain temporal integrity while enabling further feature extraction.
	Skip Connection	A skip connection bypasses S3, taking the output spiking map from S2 and directly connecting to the output of S3, preserving spike timings and reducing signal degradation.
	Combining Skip and S3 Outputs	Outputs combined: Logical OR, Logical AND, or First Spike Selection.
S4	**Final Convolution Layer**	Integrates the combined spiking map, extracting high-level features and preparing the data for classification.
	Output Layer	Determines the class label based on the spiking activity from S4.

Table 3.6: Summary of the residual SNN architecture, detailing the function of each layer, including skip connections and the methods used to combine outputs from different layers.

The skip connection in this model serves to address the issue of temporal degradation by providing a direct pathway for spikes, thereby pre-

serving spike timings that are critical for effective classification. By applying different methods to combine the spiking maps, such as logical operations and first spike selection, the model aims to retain the temporal precision of the spiking information, which is hypothesized to be beneficial for optimal information processing.

3.3.7 Surrogate Gradient-Based Spiking Neural Network

In traditional deep learning, the training process benefits significantly from backpropagation, which relies on the differentiability of activation functions to compute gradients. However, in SNNs, the discrete and non-differentiable nature of spikes makes direct backpropagation inapplicable. To address this challenge, surrogate gradient methods have been introduced. Surrogate gradient learning involves approximating the non-differentiable spiking function with a smooth, surrogate function during the backward pass, which allows for the calculation of gradients and the use of gradient-based optimization techniques.

In surrogate gradient methods, the forward pass still employs the standard spiking mechanism, such as the Heaviside step function, while the backward pass uses a differentiable surrogate function. A common choice for the surrogate gradient is the gradient of a sigmoid or a piecewise linear function that approximates the spiking function in a way that is computationally efficient and provides useful gradient information. By employing these surrogate gradients, SNNs can benefit from gradient descent-based optimization, enabling more effective training of deeper architectures and achieving performance closer to that of traditional artificial neural networks.

The model used in this study to explore the surrogate gradient method consists of a simple convolutional architecture designed for the MNIST dataset. The model is structured as in Table 3.7.

The surrogate gradient used in this model was a piecewise linear function that approximates the Heaviside step function during backpropagation. Specifically, the gradient is defined as:

$$\frac{\partial S}{\partial u} = \gamma \max(0, 1 - |u|) \tag{3.6}$$

where S represents the spike output, u is the membrane potential, and γ is a scaling factor that controls the steepness of the surrogate function. This approach provides a non-zero gradient when the membrane potential u is close to the threshold, allowing for efficient weight updates during training.

By utilizing the surrogate gradient approach, this model was able to train effectively despite

Layer	Name	Description
Input Layer		The input to the network is a spike-encoded version of the MNIST images, with dimensions $2 \times 28 \times 28$, (2 represents the number of channels).
Conv1	**Convolutional Layer 1**	Applies 12 filters of size 5×5 to the input spike maps, extracting local features such as edges and textures, providing a foundational representation of the input data.
Conv2	**Convolutional Layer 2**	64 filters, size 5×5, building upon the features extracted by the first layer to learn more complex and abstract representations.
Flatten	**Flattening Layer**	Flattening into a one-dimensional vector to prepare it for the fully connected layer.
Dense	**Dense Output Layer**	The final dense layer consists of 10 neurons, each representing one of the possible classes in the MNIST dataset.

Table 3.7: Summary of the SNN architecture, detailing each layer's function from input preprocessing to the final dense output layer.

the non-differentiable nature of spiking neurons. The combination of convolutional layers and surrogate gradient-based learning enabled the network to learn useful feature representations for the classification of MNIST digits, bridging the gap between biologically plausible spiking dynamics and effective deep learning training methodologies.

3.3.8 Liquid State Machine Model

Liquid State Machines, the scheme of which is reported in Figure 3.9, are a type of recurrent SNN inspired by the biological structure of the brain cortex. They are based on the concept of reservoir computing, where a fixed, recurrent reservoir of spiking neurons processes temporal data, and a readout layer performs the classification. The LSM model used in this study follows the architecture described by Biswas et al. (2023) [8], which explores the use of LSMs for multimodal data, including vision and speech classification.

The key components of the LSM model designed are as follows:

- **Input Layer**: The input layer receives preprocessed spiking data. For visual data such as MNIST, the images are normalized and converted into spiking input using a Poisson random spike generator. This layer establishes the spiking representation of the input stimuli, which is then fed into the reservoir.

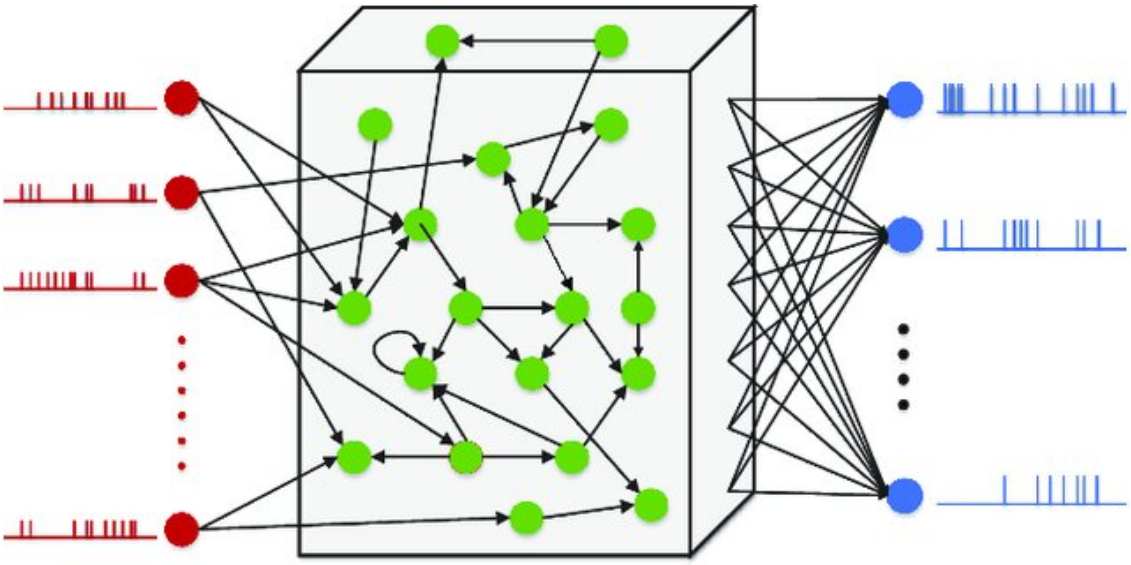


Figure 3.9: Schematic representation of the LSM structure used. Source: [35]

- **Reservoir**: The reservoir is a three-dimensional network composed of 1000 spiking neurons arranged in a $10 \times 10 \times 10$ grid. The neurons in the reservoir are a mix of excitatory and inhibitory types, following an $80\%/20\%$ ratio. Connections between neurons are probabilistic and depend on the Euclidean distance between neurons i and j, defined by the following equation:

$$\text{Prob}(i, j) = C \exp\left(-\left(\frac{D(i, j)}{\lambda}\right)^2\right) \tag{3.7}$$

  where $D(i, j)$ represents the distance between neurons i and j, and C is a constant that varies depending on the type of connection. The value of C is set based on the type of neurons involved: excitatory-excitatory, excitatory-inhibitory, inhibitory-excitatory, or inhibitory-inhibitory. The outgoing weights are set to w_{LSM} for excitatory neurons and $-w_{LSM}$ for inhibitory neurons. The reservoir's recurrent structure enables it to generate a high-dimensional, temporal representation of the input data, which can then be used by the readout layer for classification. This approach is similar to how the human cortex processes complex sensory data.

- **Readout Layer**: The readout layer is a fully connected layer that takes the spike trains from the reservoir and uses them to classify the input data. The spike trains are converted to firing rates, which serve as the input features for the readout layer. The readout layer in this study is a simple linear classifier that predicts the class label based on the spiking activity of the reservoir.

- **Time Partitioning**: In some experiments, the spike trains from the reservoir were partitioned into several time segments, and the spike rates were calculated independently for each segment. This time-partitioning technique allowed for better representation of temporal information and improved the classification performance for time-varying datasets.

The reservoir's neurons are modeled as LIF neurons, where the membrane potential $v_i(t)$ of the i-th neuron evolves according to the following differential equation:

$$\frac{dv_i}{dt} = -\frac{v_i(t)}{\tau_v} + u_i(t) - \theta\sigma_i(t) \tag{3.8}$$

where τ_v is the decay time constant, $u_i(t)$ represents the synaptic current input, θ is the spiking threshold, and $\sigma_i(t)$ represents the spike-train of the neuron. The synaptic current $u_i(t)$ is calculated as:

$$u_i(t) = \sum_{jeqi} w_{j,i}(\sigma_j * \alpha_u)(t) \tag{3.9}$$

where $w_{j,i}$ is the weight of the connection from neuron j to neuron i, and

$$\left(\alpha_u(t) = \frac{1}{\tau_u} e^{-t/\tau_u} H(t)\right) \tag{3.10}$$

is the synaptic filter, with $H(t)$ being the Heaviside function.

The Liquid State Machine's reservoir provides a rich representation of the input data that can be used effectively for classification tasks by a simple linear readout layer. The use of a fixed reservoir with varying input modalities allows for efficient processing across different datasets, such as MNIST, Neuromorphic MNIST and Soli without requiring changes to the reservoir structure itself. The inclusion of time-partitioning further enhances the model's performance by allowing it to capture more detailed temporal features of the input data.

3.4 Training Techniques and Parameter Settings

The training of SNNs involves unique complexities compared to traditional artificial neural networks due to their event-driven nature and the discrete spikes used for information transmission. Unlike ANNs, where backpropagation can be applied directly due to differentiable activation functions, SNNs require specialized training approaches to handle the non-differentiable

spike events. In this section, we discuss the different training methods used for the SNNs developed in this work. We explore biologically inspired learning rules, such as STDP and R-STDP, alongside more classic approaches like Surrogate Gradient Methods that allow gradient-based optimization for spiking models. Additionally, we describe the specific parameter settings used to optimize network performance, including hyperparameter tuning and the rationale for selecting specific values.

3.4.1 STDP and R-STDP

Spike-Timing-Dependent Plasticity is a biologically inspired, unsupervised learning rule that plays a fundamental role in training SNNs. It allows synaptic weights to be modified according to the precise timing of spikes between presynaptic and postsynaptic neurons, enabling the network to learn temporal patterns in the input data. Specifically, the synaptic weight change (ΔW) depends on the time difference (Δt) between the presynaptic and postsynaptic spikes. When a presynaptic neuron fires shortly before a postsynaptic neuron, the synapse is potentiated (strengthened), indicating a causal relationship. Conversely, if the postsynaptic neuron fires before the presynaptic neuron, the synapse is weakened. This learning rule is given by:

$$\Delta W_{ij} = \begin{cases} A_+ e^{-\Delta t/\tau_+}, & \text{if } \Delta t > 0 \text{ (potentiation)}, \\ -A_- e^{\Delta t/\tau_-}, & \text{if } \Delta t < 0 \text{ (depression)}, \end{cases} \tag{3.11}$$

where A_+ and A_- are learning rates for potentiation and depression, respectively, and τ_+ and τ_- represent the time constants that determine the rate at which potentiation or depression decays over time. This temporal learning mechanism enables the network to capture causal relationships in spiking activity, essential for processing temporal sequences and dynamic environments [1, 2].

In our implementation the synaptic weight updates are bounded to prevent runaway synaptic growth. The weight change is influenced by a stabilizer term that ensures the weights remain within biologically plausible limits, given by:

$$\Delta W_{ij} = A_+ \times (W_{ij} - LB) \times (UB - W_{ij}), \quad \text{if } \Delta t > 0 \tag{3.12}$$

where LB and UB are the lower and upper bounds of synaptic weights, respectively, ensuring that synaptic modifications remain stable during learning.

Reward-Modulated Spike-Timing-Dependent Plasticity extends STDP by integrating a re-

ward signal, inspired by biological neuromodulation mechanisms such as dopamine processes. This form of learning combines temporal learning with reinforcement, allowing the network to adjust synaptic weights based on both the spike timing and the outcome of network activity. R-STDP is particularly useful in reinforcement learning scenarios where the network must learn sequences of actions that lead to a reward.

In R-STDP, an eligibility trace is used to accumulate information based on the timing of presynaptic and postsynaptic spikes. When a reward signal is introduced, this eligibility trace determines the extent of the synaptic weight change. The eligibility trace represents a decaying memory of recent synaptic activity, given by:

$$E_{ij}(t) = E_{ij}(t-1)e^{-\Delta t/\tau_e} + \text{(contribution from recent spike activity)} \tag{3.13}$$

where τ_e is the decay constant, controlling how quickly past activities lose their influence on synaptic changes. The weight update rule for R-STDP is then given by:

$$\Delta W_{ij} = \eta R E_{ij} \tag{3.14}$$

where η is the learning rate, R is the reward signal (positive for reinforcement, negative for punishment), and E_{ij} is the eligibility trace between the presynaptic neuron j and postsynaptic neuron i [2].

In our implementation, R-STDP has been used to train convolutional SNNs, allowing the network to learn spatial and temporal features through reinforcement learning. The eligibility trace mechanism ensures that synaptic changes are informed by both recent neural activity and reward signals, providing a biologically plausible way to link neural activity to behavioral outcomes. This combination of STDP and R-STDP facilitates both simple associative learning and more complex, goal-oriented behaviors, making these methods suitable for real-time applications and neuromorphic systems.

Furthermore, a generalized formula has been introduced to accommodate both STDP and R-STDP rules, capturing the direction and magnitude of synaptic changes. This generalized form can be expressed as:

$$\delta_{ij} = \begin{cases} \alpha\phi_r a_r^+ + \beta\phi_p a_p^-, & \text{if } t_j - t_i \leq 0, \\ \alpha\phi_r a_r^- + \beta\phi_p a_p^+, & \text{if } t_j - t_i > 0 \text{ or neuron j never fires,} \end{cases} \tag{3.15}$$

$$W_{ij} = \delta_{ij}(W_{ij})(1 - W_{ij}) \tag{3.16}$$

where i and j refer to the post- and pre-synaptic neurons, respectively, and W_{ij} represents the amount of weight change for the synapse connecting these two neurons. The parameters a_r^+, a_r^-, a_p^+, and a_p^- scale the magnitude of weight changes, with a_r^+ and a_p^+ being non-negative (indicating potentiation) and a_r^- and a_p^- being non-positive (indicating depression). The controlling factors ϕ_r and ϕ_p are updated after each batch of inputs, based on classification performance metrics N_{hit} and N_{miss}, which denote the number of correctly and incorrectly classified samples, respectively.

Additionally, a k-winner-take-all mechanism is employed during plasticity, where only k neurons are eligible for synaptic changes. The neurons are selected based on their earliest spike times and internal potentials. This mechanism ensures competition between neurons and prevents over-representation of any specific feature, thus fostering diverse feature extraction. To further enhance stability, the term $W_{ij}(1 - W_{ij})$ is included, ensuring that synaptic weights remain within a bounded range $[0, 1]$, and stabilizing weight changes as they converge.

Two types of lateral inhibition mechanisms are also employed to propagate more salient information across the network. The first type ensures that if a neuron located at position (x, y) in one feature map fires, the other neurons at the same spatial location but in different feature maps are inhibited from firing. This helps to prevent redundant information being processed across multiple orientations or channels. The second type of lateral inhibition increases the spike latencies of nearby neurons in the same feature map by a factor proportional to their Euclidean distance from the firing neuron. Specifically, inhibition is applied within a radius of 1 to 5 pixels, with inhibition factors of 15%, 12%, 10%, 7%, and 5%, respectively. These mechanisms enhance the network's ability to focus on the most salient features, improving the efficiency of information propagation and reducing redundancy in feature representation.

3.4.2 Comparison of R-STDP and STDP

In this work, we compare the performance of our model—trained using R-STDP—with a version trained using conventional STDP for object recognition tasks. The comparison aims to evaluate the advantages and limitations of incorporating reinforcement learning signals into SNNs versus purely timing-based plasticity. The process begins with training the network using STDP, allowing the network to extract relevant features from the input data in an unsupervised manner. Specifically, final layers are targeted for feature extraction, producing feature

vectors that are subsequently used for classification tasks. Three distinct types of feature vectors are generated for comparison:

1. **First-Spike Vector**: This binary vector contains zeros for all values except the entry corresponding to the neuronal grid with the earliest spike, thereby encoding the spatial location of the earliest spiking neuron. This vector serves to capture temporal priority among neurons.
2. **Spike-Count Vector**: This feature vector records the total number of spikes emitted by neurons within each grid during the input presentation window. It captures the firing frequency of neurons, effectively representing rate coding across spatial dimensions.
3. **Potential Vector**: This vector retains the maximum membrane potential reached by each neuron in the grid, irrespective of the firing threshold. This allows for the representation of sub-threshold neuronal dynamics, which can provide additional information not captured by spike timing alone.

After extracting the feature vectors from both training and testing sets, two machine learning classifiers—K-nearest neighbors (KNN) and Support Vector Machines (SVM)—are employed to evaluate the discriminative capability of the learned features. This evaluation process highlights the potential benefits of utilizing temporal features derived from both STDP and R-STDP training. To maintain a fair comparison between STDP and R-STDP, identical hyperparameters, including learning rates and synaptic plasticity rules, were employed, with the exception of the magnitude of LTP and LTD for STDP. The values for these parameters were explored empirically to ensure optimal performance under STDP, while the R-STDP model adjusted synaptic plasticity dynamically based on reward and punishment signals. The use of R-STDP integrates environmental feedback through the release of neuromodulators, such as dopamine and acetylcholine, which modulate synaptic plasticity based on external success or failure signals. In contrast, STDP operates solely based on spike timing, without consideration of feedback signals. The key hypothesis tested in this comparison is whether the incorporation of reinforcement signals in R-STDP can lead to superior feature extraction and classification performance by guiding the network towards behavior that maximizes the expected reward over time.

3.4.3 Surrogate Gradient Method

The Surrogate Gradient Method addresses one of the fundamental challenges in training SNNs: the non-differentiability of the spiking function, which makes conventional gradient-based op-

timization like backpropagation difficult. In a typical SNN, spikes are generated by a discontinuous thresholding function, which does not have a gradient that can be used for error backpropagation. The surrogate gradient method solves this issue by introducing a differentiable approximation of the spiking function, enabling gradient-based training of SNNs.

In this approach, as described in 3.3.7, a surrogate gradient approximates the derivative of the spiking function during backpropagation, enabling the training of deep SNNs using standard optimization techniques Mathematically, let $V(t)$ represent the membrane potential of a neuron at time t, and let $S(t)$ denote the spike output, where $S(t) = H(V(t) - V_{\text{th}})$, with $H(\cdot)$ being the Heaviside step function and V_{th} the firing threshold. The Heaviside function is non-differentiable, which prevents the direct application of gradient-based optimization. In the surrogate gradient approach, a smooth function $\sigma(V)$ is used to approximate $H(V)$ during the backward pass:

$$\frac{\partial S(t)}{\partial V(t)} \approx \sigma'(V(t) - V_{\text{th}}) \tag{3.17}$$

Common choices for $\sigma(V)$ include the sigmoid function:

$$\sigma(V) = \frac{1}{1 + e^{-\lambda V}} \tag{3.18}$$

where λ controls the steepness of the sigmoid, or a piecewise linear function that approximates the gradient within a certain range around the threshold. This enables the use of gradient descent to adjust synaptic weights, effectively training the network in a manner similar to traditional artificial neural networks.

In our implementation, we used the surrogate gradient approach provided by the `snnTorch` library [36]. This library, built on top of `PyTorch`, allows for the seamless integration of surrogate gradient training within the broader deep learning ecosystem. By leveraging `PyTorch`'s automatic differentiation capabilities, `snnTorch` simplifies the implementation of surrogate gradients, enabling efficient training of spiking neurons. The surrogate gradient method has proven to be effective in training deeper SNN architectures, overcoming the limitations imposed by the non-differentiable nature of spike activations. By approximating the gradient of the spiking function, surrogate gradients provide a practical solution for training SNNs using backpropagation, making them more competitive with traditional deep learning models while maintaining the energy efficiency and temporal processing capabilities of spiking neurons, but sacrificing, on the other hand, biological plausibility.

3.4.4 Training techniques for LSM

In LSMs, the liquid itself—composed of a recurrent reservoir of spiking neurons—is not directly trained. Instead, the liquid serves as a dynamic transformation module that projects the input signals into a high-dimensional and complex internal state space. The role of the liquid is to create a rich repertoire of spatiotemporal features that can be utilized for decision-making without adjusting the internal synaptic weights.

The actual training occurs at the readout layer, which is a simple linear or non-linear classifier attached to the output of the liquid. The readout layer takes the transformed states produced by the liquid and learns to map them to the desired output classes. In our experiments, we employed two distinct training techniques for the readout layer:

- **Classical ANN Training**: The readout layer was trained as a traditional ANN, treating the outputs from the liquid as fixed feature vectors. This approach makes use of standard backpropagation techniques to update the weights of the readout, optimizing a loss function (e.g., cross-entropy loss) to minimize classification error.
- **Surrogate Gradient SNN Training**: We also explored training the readout layer using surrogate gradient methods suitable for SNNs. This approach preserves the spiking nature throughout the readout, maintaining consistency with the rest of the spiking architecture.

By using these two training approaches, we compared the performance and effectiveness of conventional ANN training against SNN-based readout, which takes advantage of the temporal dynamics inherent in SNNs. Each approach has its benefits: the ANN readout provides a straightforward and computationally efficient solution, while the SNN readout leverages the temporal information encoded within spike timings, potentially improving performance on tasks requiring precise temporal relationships.

3.5 Overfitting Avoidance Mechanisms

In reinforcement learning settings, the risk of overfitting is high, as the network may become overly specialized to maximize rewards on the training examples, resulting in poor generalization to unseen data. To prevent the network from being trapped in local optima or overfitting, two mechanisms were employed during the training phase. These approaches aim to ensure that the network can explore other solutions that generalize well to both seen and unseen samples.

3.5.1 Adaptive Learning Rate

The adaptive learning rate mechanism aims to balance the effects of rewards and punishments received during training. At the beginning of training, the weights are initialized randomly, often leading to high misclassification rates, as the network has yet to learn discriminative features effectively. During this initial stage, the ratio of correctly classified samples to misclassified ones is low, resulting in more punishment signals being applied. This can cause rapid weakening of synaptic weights, leading to "dead" or highly selective neurons that respond to only a few inputs. Conversely, as the training progresses, a higher rate of correct classifications may lead to more reward signals, causing the network to become increasingly selective and effectively ignore difficult samples, which may also lead to overfitting.

To counteract this, an adjustment factor is used to modulate weight changes, balancing the impact of both correct and incorrect samples throughout the training iterations. Specifically, if N represents the total number of training samples, and N_{hit} and N_{miss} represent the number of correctly and incorrectly classified samples, respectively, in the last iteration, the weight changes for the current iteration are adjusted as follows:

$$\Delta W_{ij} = \Delta W_{ij} \times \begin{cases} \frac{N_{\text{miss}}}{N} & \text{if a reward is received,} \\ \frac{N_{\text{hit}}}{N} & \text{otherwise,} \end{cases}$$

where W_{ij} represents the weight of the connection between neurons i and j. This mechanism helps in mitigating the imbalance between rewards and punishments and ensures that the network is encouraged to explore a wider set of possible solutions.

Learning Rate Scheduling for Surrogate Gradient and LSM Models

For the surrogate gradient models and the Liquid State Machine LSM models, an additional mechanism—Learning Rate Scheduling—was employed to facilitate smoother convergence. Specifically, the Reduce on Plateau strategy was used to adapt the learning rate during training. This method was employed to counteract the challenge of vanishing or diverging gradients often encountered in training deep spiking networks with surrogate gradients.

The Reduce on Plateau scheduler monitors the model's performance (typically the validation loss) and reduces the learning rate when the performance stagnates. The motivation for this approach lies in the dynamic nature of SNN training, where initial large learning rates facilitate exploration of the parameter space, while smaller learning rates in later stages enable

fine-tuning for better local optima. The learning rate update rule used in the Reduce on Plateau strategy can be described as:

$$\eta_{t+1} = \begin{cases} \eta_t \times \gamma, & \text{if } M(t) \geq M(t - \Delta t), \quad \forall \Delta t = 1, \ldots, k \\ \eta_t, & \text{otherwise} \end{cases} \tag{3.19}$$

where η_t represents the learning rate at epoch t, $\gamma \in (0, 1)$ is the reduction factor, and $M(t)$ denotes the monitored metric, such as validation loss. The learning rate is reduced by multiplying it with γ if the validation metric does not improve over k consecutive epochs. For the surrogate gradient models and LSM implementations, γ was empirically set to 0.1, and k was set to 5 epochs. This dynamic adjustment of the learning rate allows the model to escape from potential saddle points or plateaus in the loss landscape and helps fine-tune the learning in the later stages of training, thereby leading to better convergence. For SNNs trained with surrogate gradients, this approach is crucial in overcoming the challenges related to the non-linear and spiking nature of the networks, which often complicates the optimization process compared to traditional neural networks.

3.5.2 Dropout

Dropout is another technique employed to mitigate overfitting during training. In RL scenarios, the goal is to maximize the expected reward, but the network may become dependent on a small subset of features that are sufficient to classify most training examples correctly. This can lead to severe overfitting, especially in complex tasks, as the network may leave certain neurons untrained. Untrained neurons often contribute to the degradation of network performance by randomly firing, reducing accuracy on test samples.

To address this issue, dropout is employed to deactivate individual neurons at random during training with a certain probability p_{drop}. Specifically, neurons were temporarily switched off with the probability p_{drop}, ensuring that each neuron actively contributed to feature extraction throughout training. By enforcing the involvement of more neurons, dropout increases the likelihood of discovering more discriminative features, while simultaneously reducing the tendency of neurons to engage in blind firing. This, in turn, improves the generalization capacity of the model and reduces overfitting.

3.5.3 Training Schedulers

The training process of SNNs presents unique challenges, such as long training times and difficulty in convergence due to the non-differentiable nature of spikes and the high sensitivity of hyperparameters. To address these issues and optimize the training phase, different schedulers were employed to prevent overfitting, improve convergence, and balance learning progress efficiently. Specifically, an early stopping mechanism was used to terminate training when progress stalled, while learning rate scheduling was utilized to adapt the learning rate dynamically based on training performance.

Early Stopping Mechanism

In order to prevent overfitting and reduce computational resources, an Early Stopping technique was integrated into the training pipeline. Early stopping aims to halt training if the model's performance does not improve over a predetermined number of epochs, thereby mitigating the risk of overfitting and conserving computational power. In the current study, early stopping was set to trigger if no significant improvement in the validation metric was observed for a maximum of 10 consecutive epochs.

Mathematically, if $M(t)$ represents the metric (e.g., validation accuracy or loss) at epoch t, the stopping condition is met when:

$$M(t) \leq M(t-1), \quad \forall t = t_0, t_0+1, \ldots, t_0+9 \tag{3.20}$$

where t_0 is the epoch at which the metric ceased to improve. This approach effectively prevents the model from continuing to learn from training data when no further generalization improvements are achievable. Early stopping is particularly advantageous for SNNs, where prolonged training without convergence can lead to the saturation of synaptic weights and overfitting to noise.

3.5.4 Data Augmentation

Data augmentation is a crucial technique used to enhance the generalization capabilities of neural networks by artificially expanding the training dataset. By generating diverse variations of existing training examples, data augmentation helps prevent overfitting, where a model learns to perform exceptionally well on the training data but fails to generalize to unseen examples. In the context of SNNs, overfitting can be especially problematic due to the potential lack of

diversity in the spike-encoded inputs. Thus, data augmentation is employed to provide a richer and more varied set of input patterns that force the network to learn generalizable features.

In our experiments, several data augmentation methods were employed, including random rotations, random horizontal and vertical flips, Gaussian noise addition, and masking of rows and columns. Random rotations were applied to training images to create new perspectives of each sample, making the network more robust to changes in orientation. Random horizontal and vertical flips were also used to increase robustness to symmetry variations. Gaussian noise was added to simulate realistic sensory noise, which helps in training a network that can handle noisy or corrupted inputs effectively. Additionally, masking entire rows and columns serves to occlude portions of the input, thereby simulating scenarios where some features may be missing or occluded. This encourages the network to rely on broader feature sets rather than focusing on any single specific feature, thus mitigating overfitting by discouraging overly specific feature representation. By employing these augmentation techniques, we aimed to increase both the diversity of the input samples and the network's ability to generalize beyond the training dataset. In Figure 3.10 we report some samples obtained by augmentation of the original one.

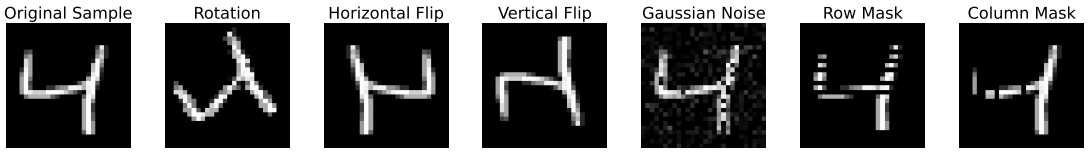


Figure 3.10: Visualization of different augmentations applied to an MNIST sample, including random rotation, horizontal flip, vertical flip, added Gaussian noise, row mask, and column mask, along with the original image.

Data augmentation serves as a practical strategy for reducing overfitting in SNNs, promoting more robust learning and improved performance on unseen data. The combined use of these augmentation methods ensures that the network is exposed to a wide variety of possible input variations, helping it develop more generalizable features.

3.6 Hardware and Computational Environment

In this study, the development and training of SNNs were conducted using conventional computing resources rather than specialized neuromorphic hardware. Although already cited neuromorphic platforms are optimized for spike-based computation and energy-efficient processing, they were not available for use in this research. As a next step, the methods and models

developed in this work will be adapted for deployment on these neuromorphic systems, with the expectation of achieving significant improvements in power efficiency and scalability.

The SNN models were trained and tested on a personal computing system with the key specifications reported in Table 3.9.

Software/Library	Version
`Python`	`3.11.9`
`PyTorch`	`2.3.0`
`snnTorch`	`0.9.1`
`Spyketorch`	–
`NumPy`	`2.1.2`
`Matplotlib`	`3.9.2`
`SciPy`	`1.14.1`
`Scikit-learn`	`1.5.2`

Table 3.8: Software Environment Details

Component	Specification
Processor	13th Gen Intel(R) Core(TM) i9-13900H @ 2.60 GHz
Installed RAM	32.0 GB (31.6 GB usable)
GPU	NVIDIA GeForce RTX 4070 Laptop GPU
CUDA Version	12.1.105

Table 3.9: Key Hardware Specifications of the Training System

The high-performance CPU with multiple cores and 32 GB of RAM enabled efficient handling of the computational demands for training SNNs. The NVIDIA GeForce RTX 4070 GPU significantly reduced training time through GPU-based parallelism, especially for methods like surrogate gradient approaches involving approximate gradients over multiple layers. Although traditional computing lacks the efficiency of neuromorphic systems, it offered a practical platform for developing and validating our models. The GPU was crucial in managing the complexity of training SNNs, which involve intricate temporal dynamics and spike-based computations.

An overview of the software and libraries used is provided in Table 3.8.

Model	S-Layer Parameters				C-Layer Parameters				Parameters for Synaptic Plasticity						
	Layer	no. Feature Maps	Input Window	Threshold	Layer	Input Window	Stride	Type	Layer	a_r^+	a_r^-	a_p^+	a_p^-	k	r
3.3.1	S2	features * n. classes	(w, h, 4)	42	C2	(1, 1)	0	Spike	S2	0.01	-0.0035	-0.01	0.0006	5	3
3.3.2	S1	30	(5, 5, 6)	15	C1	(2, 2)	2	Spike	S1	0.004	-0.003	0	0	5	3
	S2	250	(3, 3, 30)	10	C2	(3, 3)	3	Spike	S2	0.004	-0.003	0	0	8	1
	S3	200	(5, 5, 250)	∞	C3	(5, 5)	0	Pot.	S3	0.004	-0.003	0.0005	-0.004	1	0
3.3.3	S1	30	(5, 5, 6)	15	C1	(2, 2)	2	Spike	S1	0.004	-0.003	0	0	5	3
	S2	150	(3, 3, 30)	10	C2	(2, 2)	2	Spike	S2	0.004	-0.003	0	0	8	1
	S3	250	(3, 3, 150)	50	C3	(3, 3)	3	Spike	S3	0.004	-0.003	-/0.0005	-/-0.004	15	1
	S4	200	(5, 5, 250)	∞	C3	(3, 3)	0	Pot.	S4	0.004	-0.003	0.0005	-0.004	1	1
3.3.5	S1	30	(5, 5, 6)	15	C1	(2, 2)	2	Spike	S1	0.004	-0.003	0	0	5	3
	S2	75	(3, 3, 30)	10	C2	(2, 2)	2	Spike	S2	0.004	-0.003	0	0	8	1
	S3_a	100	(3, 3, 75)	50	C3	(3, 3)	3	Spike	S3	0.004	-0.003	0.0005	-0.004	15	1
	S3_b	100	(5, 5, 75)	50	C3	(3, 3)	0	Pot.	S4	0.004	-0.003	0.0005	-0.004	1	1
	S3_c	100	(7, 7, 75)	50											
	S4	200	(5, 5, 300)	∞											
3.3.4	S1	30	(5, 5, 6)	15	C1	(2, 2)	2	Spike	S1	0.004	-0.003	0	0	5	3
	S2	250	(3, 3, 30)	10	C2	(3, 3)	3	Spike	S2	0.004	-0.003	0	0	8	2
	S3_a	200	(3, 3, 250)	∞	C3_a	(5, 5)	0	Pot.	S3	0.004	-0.003	0.0005	-0.004	1	0
	S3_b	200	(5, 5, 250)	∞	C3_b	(5, 5)	0	Pot.							
	S3_c	200	(7, 7, 250)	∞	C3_c	(5, 5)	0	Pot.							
3.3.6	S1	30	(5, 5, 6)	15	C1	(2, 2)	2	Spike	S1	0.004	-0.003	0	0	5	3
	S2	150	(3, 3, 30)	10	C2	(2, 2)	2	Spike	S2	0.004	-0.003	0	0	8	1
	S3	250	(3, 3, 150)	50	C3	(3, 3)	3	Spike	S3	0.004	-/0.0005	-/-0.004	15	1	
	S4	200	(5, 5, 250)	∞	C3	(3, 3)	0	Pot.	S4	0.004	-0.003	0.0005	-0.004	1	1

Table 3.10: Comprehensive overview of the parameters used for training the different models across all experiments. The table is divided into three main sections: (1) **S-Layer Parameters**, which specify the number of feature maps, the size of the input windows, and the spike thresholds used for each layer; (2) **C-Layer Parameters**, which define the convolutional properties such as input window size, stride length, and type (Spike-based or Potential-based); and (3) **Synaptic Plasticity Parameters**, including the values of synaptic scaling factors (a_r^+, a_r^-, a_p^+, a_p^-), the number of winning neurons (k), and the radius of lateral inhibition (r) used in the experiments. Each row describes specific configurations for different models or tasks, covering baseline, deep, inception-inspired, population coding, and residual network variants, allowing for an exhaustive comparison of each configuration's role in network performance.

4

Experiments and Results

This chapter provides a comprehensive examination of the experimental setup, methodologies, and results that are crucial to understanding the application and performance of SNNs in diverse computational scenarios. The goal of this chapter is to detail the process of validating and extending the methods developed throughout this research. A significant emphasis is placed on demonstrating both the capabilities and limitations of SNNs under varying architectural and algorithmic modifications. The experiments span a range of tasks, focusing on assessing network efficiency, adaptability, and effectiveness in solving classification problems, while also exploring the energy efficiency benefits inherent to neuromorphic computing. This chapter is divided into detailed discussions on the experimental setup, the training and testing phases, specific parameter settings, and the evaluation metrics that are vital in analyzing the outcomes of the experiments. The outcomes are presented alongside discussions that frame these findings in the context of current advancements in the field, thus providing a well-rounded understanding of the implications of this work.

4.1 Experimental Setup

The experiments were conducted using a personal laptop, as neuromorphic hardware was unavailable. This limitation suggests that a natural progression for this work would involve implementation on neuromorphic hardware, which is better suited to capitalize on the energy-efficient capabilities of SNNs. The hardware and software specifications are reported in 3.6. Each reward-modulated layer was trained for 100 epochs using the early stopping methods previously discussed in 3.5. The dataset was divided into batches of 1000 samples each, following

a mini-batch training approach. In mini-batch training, instead of updating the model after each individual sample or waiting for the entire dataset, the model is updated after processing each batch of samples. This approach balances the benefits of both stochastic gradient descent and full-batch training, allowing for faster convergence and reduced variance in weight updates. The mini-batch strategy helped stabilize the training process, especially given the complexity of the spiking network models and the reinforcement learning mechanisms employed.

The evaluation metrics used in these experiments focused on classical metrics suitable for classification tasks, which include:

- **Accuracy**: Accuracy represents the ratio of correctly predicted samples to the total number of samples. It can be expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

  where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. Accuracy provides an overall measure of how well the model performs on both positive and negative samples.

- **Precision**: Precision measures the ratio of correctly predicted positive observations to the total predicted positives. It is defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

  Precision is particularly useful in scenarios where false positives are more significant and need to be minimized.

- **Recall (Sensitivity)**: Recall, also known as sensitivity, measures the ratio of correctly predicted positive observations to all the actual positives:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

  Recall is crucial in situations where it is essential to minimize false negatives, such as medical diagnosis.

- **F1 Score**: The F1 Score is the harmonic mean of precision and recall, providing a balance between the two, especially when there is an uneven class distribution. It is calculated as:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

  The F1 score is useful when both precision and recall are important and there is a need to find a balance between false positives and false negatives.

The parameters for each experiment, such as STDP learning rates, local inhibition radius, and firing thresholds, were adapted to the specific requirements of each scenario. These specific values were presented and discussed in detail in the Section 3.3.

4.2 Baseline Models

The experiments began with reproducing the results of two foundational studies by Mozafari et al. [2, 3]. These studies serve as the baseline for evaluating the performance of SNNs trained using Spike-Timing-Dependent Plasticity and Reward-Modulated Spike-Timing-Dependent Plasticity. Each experiment was conducted independently, targeting different aspects of learning in SNNs, and aimed to validate the reliability of these methods in a classification context.

4.2.1 Experiment 1: Implementing a shallow CSNN

In this experiment, we evaluated the proposed SNN architecture and learning strategy using several well-known object recognition benchmarks. The datasets used for evaluation included Caltech face/motorbike, ETH-80, and small NORB, each presenting different levels of complexity. The Caltech dataset, containing two categories (face and motorbike), served as an introductory benchmark, whereas the ETH-80 and small NORB datasets presented more challenging object recognition tasks with multiple categories and varying viewpoints.

The network was configured to extract specific features from each dataset, with an emphasis on reinforced selectivity. For the Caltech dataset, we employed ten neuronal grids for each category, which were trained to become selective to their target categories. Reinforcement was used to ensure that neurons from specific grids were responsible for recognizing either faces or motorbikes. During the early training iterations, synaptic weights were adjusted rapidly, leading to initially unstable network behavior, but the neurons gradually became more selective as the reward and punishment signals guided the learning process. After 200 iterations, the network achieved robust selectivity for training samples, and convergence continued for approximately 300 more iterations before achieving stability on testing samples. This behavior indicated that the network required more iterations to generalize features from training to testing data effectively, aided by adaptive learning rates to sustain convergence.
We conducted 30 repetitions of the experiment with different initial random weights and training samples, achieving achieving a median performance of 98.1% with an interquartile range (IQR) of $[97.6\%, 98.7\%]$. When the network was trained with STDP alone, the maximum performance achieved was 97.2%, indicating a significant improvement when using reward-modulated learning. Figure 4.1 shows the be

The same the networnetwork he training itstk over trucherations.aviour of ture was then evaluated on the ETH-80 and small NORB datasets. For the ETH-80 dataset, which includes images of eight categories from different viewpoints, the network extracted ten features per cat-

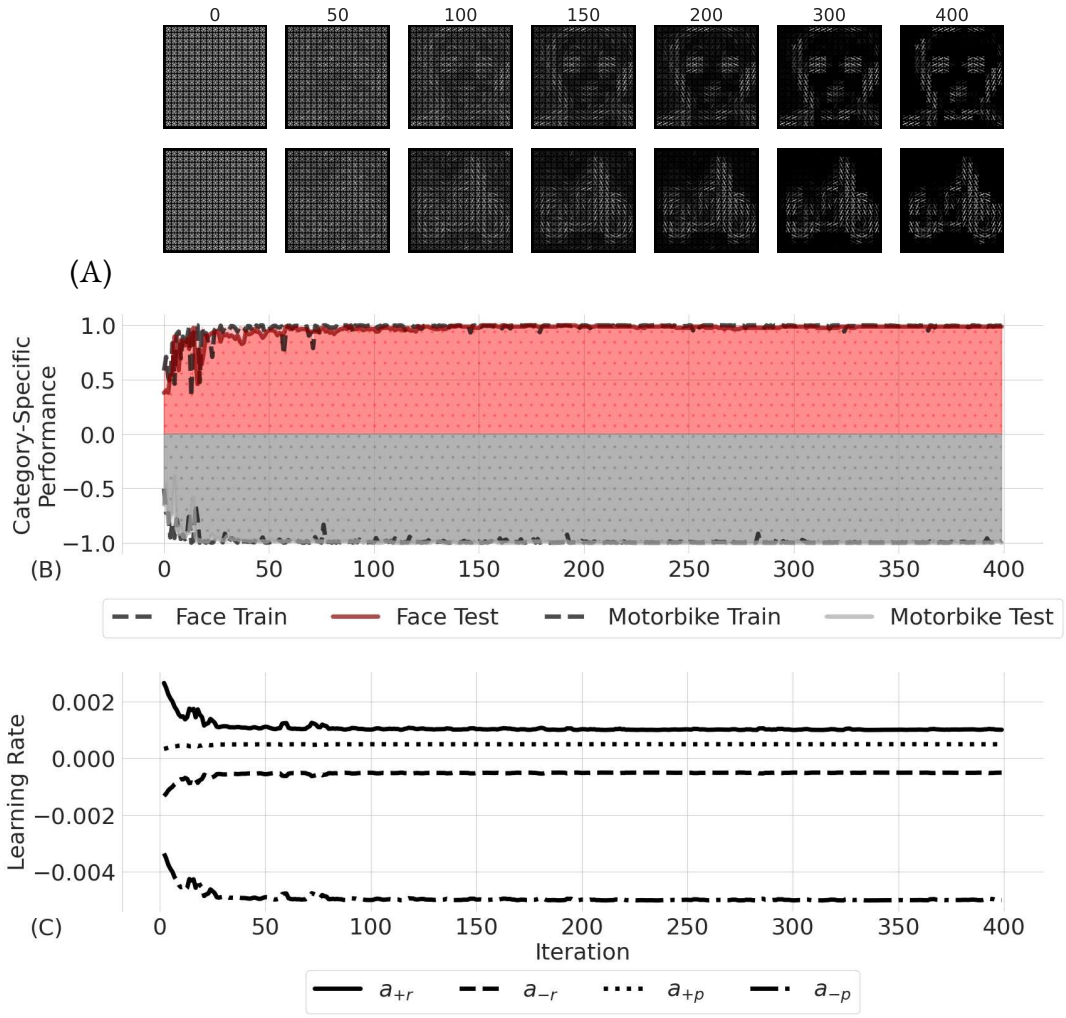


Figure 4.1: Training of the network on the Caltech face/motorbike dataset. (A) Displays the progression of two different features (from a total of 20) that were extracted by the network. The top plot represents face-specific neurons, while the bottom plot represents motorbike-specific neurons. (B) Depicts the hit rate for neurons in each category. Gray (red) filled curves indicate the proportion of times that face (motorbike) neurons emit the earliest spike in response to their target stimuli. Note that the motorbike neuron curves are inverted vertically for enhanced clarity, while hit rates over the testing dataset are represented with a dotted pattern. (C) Shows the trajectory of changes in learning rate based on the number of correct (N_{hit}) and incorrect (N_{miss}) categorizations.

egory, totaling 80 features. We presented nine instances per category as training samples, with the remaining used for testing. After 250 training iterations, the network achieved an average performance of 88.7% with an IQR of $[87.2\%, 90.3\%]$. Comparatively, applying STDP with Support Vector Machines or k-Nearest Neighbors as classifiers achieved an accuracy of 81.1%, highlighting the superior feature extraction capability of the reward-modulated approach. The

small NORB dataset, known for its complexity, was used to further validate the proposed network. Here, neurons from ten grids were employed to recognize five object categories. The network achieved an average testing accuracy of 67.4% with an IQR of $[67.0\%, 67.7\%]$, whereas STDP alone achieved a maximum of 49%. Analysis of confusion matrices revealed that both methods had difficulty distinguishing between similar classes, such as four-legged animals and humans or cars and trucks, but the reinforcement learning approach produced a more balanced error distribution and reduced confusion rates.

4.2.2 Experiment 2: R-STDP in DCSNN

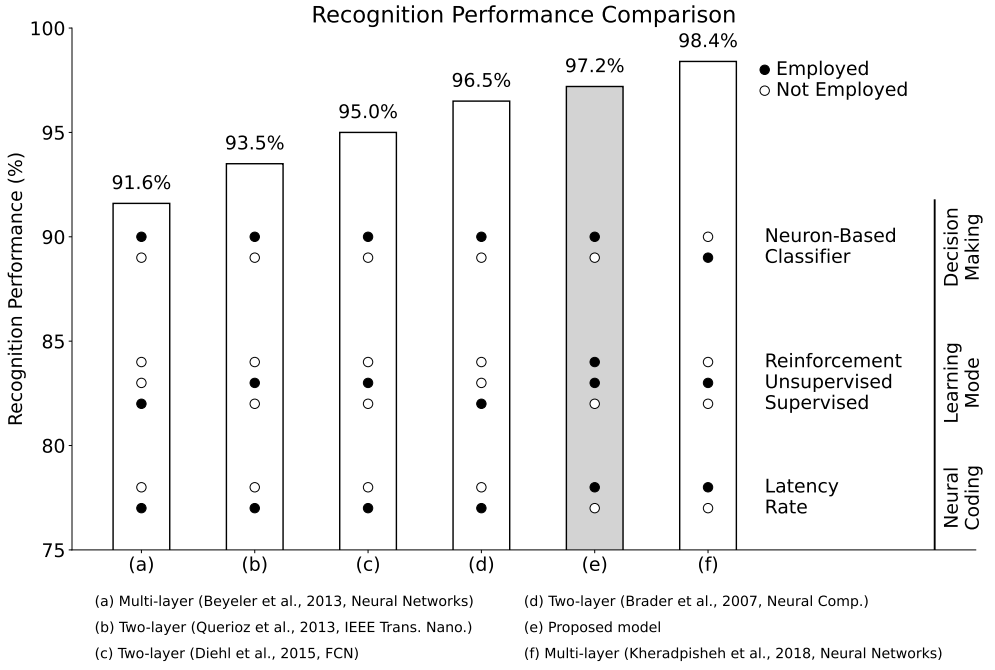


Figure 4.2: Performance comparison of various existing SNNs, trained using STDP-based learning rules, on the MNIST dataset, alongside the proposed network.

The proposed deep convolutional SNN was evaluated on the MNIST dataset, which consists of 60,000 training images and 10,000 testing images of handwritten digits. The goal was to train the network to classify handwritten digits into ten distinct classes and subsequently evaluate its recognition performance on the unseen test samples. The network's input layer contained six feature maps, each representing one of the six Difference of Gaussian filters that were applied at three different scales with both on- and off-center polarities. These feature maps conveyed spike latencies derived from the input images. The architecture of the network included three spiking layers (S1, S2, S3), each followed by a pooling layer (C1, C2, C3). The first two spiking layers were set with specific thresholds, while the third layer had an infinite

threshold, meaning that neurons in this layer never fired but accumulated the incoming spikes as internal potentials. When all spikes were accumulated, a manual spike time greater than previous spikes was assigned to the neurons in S3. The pooling layers C1 and C2 utilized spike-based local pooling, while C3 neurons applied potential-based global pooling, consistent with the behavior of S3 neurons.

The final decision was based on the C3 neuron with the highest accumulated potential, and the corresponding label was assigned to the input image. During this task, the weight stabilizer term was removed from the plasticity equation for the S3 layer, and the synaptic weights were manually limited to values between 0.2 and 0.8, which improved synaptic plasticity in later iterations of training based on empirical observations.

The network was trained layer by layer. The spiking layers S1 and S2 were trained using STDP for 4 and 8 epochs, respectively, with an image from the MNIST dataset fed into the network at each iteration. To accelerate training, the learning rates were multiplied by two after every 500 iterations until the reinforcement signal reached a value of 0.15. Training of the third spiking layer (S3) was based on R-STDP, using reinforcement signals from C3. S3 was trained for 100 epochs, using mini-batches of 1000 samples, and the recognition performance was evaluated after every epoch.

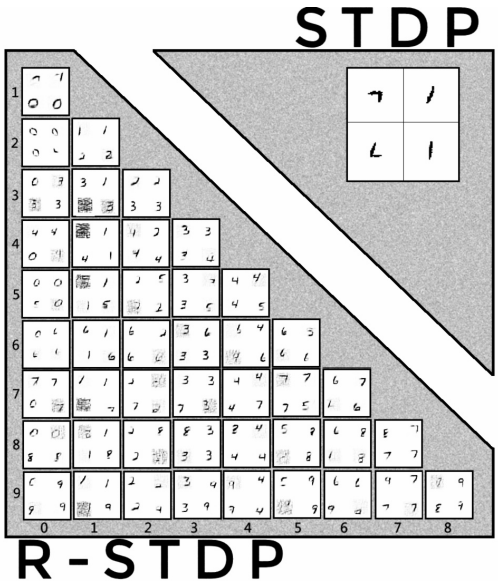


Figure 4.3: This figure illustrates the reconstruction of features extracted by S2 neurons during digit distinguishing task, under the condition where the number of feature maps is limited to four. The lower portion of the figure demonstrates the features extracted by applying R-STDP to S2 afferents, addressing each pairwise digit recognition task (indicated by row and column indices). The occurrence of untrained or partially trained feature maps (depicted as dark or gray squares) results from the dominance of more representative features, which win the plasticity competition across all images, thereby restricting other features from learning or converging. In the upper portion, the four features obtained through STDP applied to S2 neurons are presented.

The network achieved an accuracy of 97.2% on the MNIST dataset, which outperformed most previous SNNs trained with STDP alone. The training curves can be consulted in Figure 4.8. This performance was notable because no external classifier was used, thereby enhancing both the biological plausibility and hardware efficiency of the network. The experiment also evaluated the application of reward-modulated plasticity to the penultimate spiking layer (S2), which did not yield improvements in this task. Further analysis suggested that applying reward-modulation at S2 could be beneficial in scenarios with frequent distractors and limited computational resources. Another experiment involved training on the CIFAR-10 dataset, achieving an accuracy of 67.1%, still far from the state-of-art results, but proving the methods used also for more complex samples. In Figure 4.2 a performance comparison with various SNNs alongside the

proposed model on the MNIST dataset.

An additional experiment aimed to distinguish pairs of digits from the MNIST dataset while treating other digits as distractors. A smaller network was used to evaluate the application of R-STDP across multiple layers. The network was trained on 10,000 samples for each digit pair and tested on 200 samples per pair. S1 was trained with STDP, while S2 was tested with both STDP and R-STDP, and S3 always used R-STDP. The results, summarized in Figure 4.3, show that R-STDP consistently outperformed STDP when fewer feature maps were available in S2. As the number of feature maps increased, STDP narrowed the performance gap. Fig. 6 highlights that R-STDP enabled S2 neurons to extract target-specific features, whereas STDP failed to differentiate between targets.

4.3 Extended SNN Architectures

The extension of the baseline models aimed to explore the role of increased network depth in SNNs. Specifically, the architecture was extended by adding additional layers trained using either STDP or R-STDP. This provided insight into how deeper structures could enhance learning capabilities and also revealed challenges, particularly in coordinating the training of multiple R-STDP layers.

In the case of adding multiple R-STDP layers, a primary challenge emerged: how to propagate the final reinforcement signal to train more than one layer simultaneously. Two approaches were employed to address this issue: (1) updating all synapses together, based on the final output, and (2) a greedy training approach where the weights of one layer were fixed while the weights of another were updated. Both approaches presented distinct advantages and limitations. The training of multiple R-STDP layers simultaneously revealed a critical issue regarding learning stability. If the final layer modified its synaptic weights at the same rate as the penultimate layer, it could destabilize the learning process. Changes in the penultimate layer could turn a previously optimal configuration of the final layer into a suboptimal one, resulting in a negative feedback loop that disrupted the entire network's stability. This instability necessitated careful tuning of parameters and slower adjustments to prevent adverse interactions between layers. Additionally, the original networks were not constrained by strict, hard-coded parameter rules, adding complexity to finding an optimal solution.

For the STDP-trained additional layer, tuning the threshold parameters yielded meaningful results, suggesting that even in a deeper architecture, STDP could effectively guide learning by extracting important temporal features. In contrast, training multiple R-STDP layers required an alternate strategy. Initializing the weights appropriately was critical to avoid instability. This was achieved by training the network with traditional STDP for several epochs before transitioning to R-STDP training. This sequential approach helped to stabilize the synaptic weights, providing a robust starting point for reward-based modulation. Figure 4.4 illustrates examples of filters learned during these training processes, highlighting the distinct patterns and progression achieved using the STDP and R-STDP learning rules.

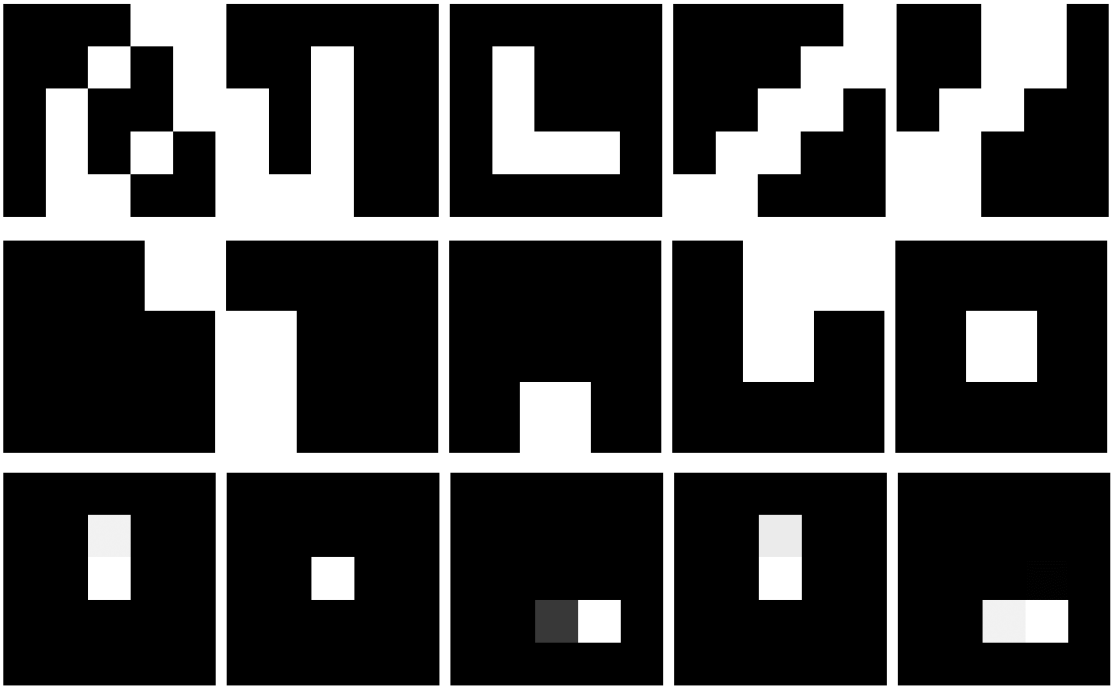


Figure 4.4: Filters from convolutional layers of the SNN, illustrating synaptic weight patterns used for detecting specific spatial features in the input data. Lighter regions indicate stronger synaptic weights, which correspond to features that the network considers important for encoding information. The filters demonstrate a hierarchical progression of feature complexity across layers, capturing increasingly abstract patterns.

When training multiple R-STDP layers, the penultimate layer was initially trained with STDP for several epochs to establish a meaningful initialization of synaptic weights. This preliminary training phase provided the network with a stable set of synaptic strengths, capturing the basic features necessary for effective classification. Following this initialization, R-STDP training was applied to the penultimate layer, which allowed it to effectively integrate reward-based synaptic modifications. This two-step process helped stabilize the training dynamics and improved the overall accuracy of the network, mitigating the potential instability that can arise from training multiple reward-modulated layers simultaneously without prior structure.

When training all R-STDP layers concurrently, the results were found to be unstable and generally degraded the performance compared to the baseline models. However, the greedy training approach showed more promising outcomes, with performance comparable to the baseline results. This suggested that by isolating the training of different R-STDP layers, the network could maintain stability and perform effective reward-based learning.

Additional experiments were conducted using the benchmark dataset CIFAR-10, to further evaluate the performance of the extended architectures, in particular considering input RGB images. These datasets introduced greater complexity and diversity, allowing for a comprehensive assessment of the network's ability to generalize across different types of visual data. The

performances achieved are reported in Table 4.1.

An additional experiment was conducted by incorporating yet another layer into the architecture. However, this extension resulted in catastrophic failure, likely due to the exacerbation of the problems associated with parameter tuning and the stability of learning dynamics across multiple layers. The attempt to add more depth beyond the extended architecture emphasized the sensitivity of such networks to parameter settings and indicated that simply adding layers does not guarantee improved performance.

4.4 Advanced SNN Architectures

The experiments in this chapter focus on evaluating and extending the capabilities of SNN models by implementing various advanced architectures. Building upon the baseline models, these experiments investigate the impact of incorporating more sophisticated features and structures, such as population coding, inception-inspired modules, residual connections, and liquid state machines. Each of these advanced architectures aims to address specific challenges related to feature extraction, model depth, temporal dynamics, and overall network robustness.

The objective is to explore the extent to which these modifications can enhance learning performance, classification accuracy, and stability across diverse benchmark datasets. The experiments span a range of datasets with varying complexity, including MNIST, CIFAR-10, allowing us to evaluate the versatility and generalization capabilities of these architectures under different scenarios.

In the subsequent sections, we provide a detailed analysis of the results obtained from each experiment, highlighting improvements, challenges, and insights gained through the integration of these advanced strategies.

4.4.1 Population Coding SNN

In this experiment, we extended the baseline SNN architecture by incorporating a population coding scheme within the decision-making layer. The architecture was inspired by a branching mechanism, similar to the approach utilized in Random Forest models, where multiple independent decision pathways are employed to improve classification robustness and accuracy. Specifically, the decision-making layer of the network was divided into three branches, each responsible for processing spatial features at different scales. The branches utilized increasing kernel sizes to capture both fine and broad spatial features effectively, providing a diverse representation of the input data.

The intuition behind using this branching structure is analogous to the way Random Forests operate: by combining the outputs of multiple branches—each focusing on different aspects of the input—the network can generate a more robust overall decision. The first branch was assigned a smaller kernel size, focusing on fine spatial details, while the other branches used

progressively larger kernels to capture broader spatial relationships within the input. This diversity in spatial analysis allows the model to adapt to varying patterns across different images, enabling better overall performance.

The layers in this population coding model were trained for 4, 8, and 100 epochs, respectively, with early stopping mechanisms implemented to prevent overfitting and reduce unnecessary computations. This training approach ensured that the features learned by each branch were relevant and could generalize well across the testing datasets.

The experimental results indicate that incorporating population coding yielded consistent improvements over the baseline model in terms of classification accuracy. Specifically, the model achieved an improvement of approximately 1% to 2% in accuracy on respectively the MNIST and CIFAR-10 datasets. This enhancement in performance demonstrates the effectiveness of a diversified decision-making approach, which benefits from integrating information across multiple spatial scales. The improved accuracies for both MNIST and CIFAR-10 datasets demonstrate the robustness of the population coding approach in the decision-making process. By allowing the network to leverage different feature scales concurrently, this architecture effectively enhances both spatial representation and decision-making capabilities. In Figure 4.8 the training curve compared with the other models.

4.4.2 Inception-like SNN

In this experiment, we evaluated an Inception-like SNN designed to enhance performance by incorporating multiple spatial scales. The network was trained on the same datasets as the baseline model, specifically MNIST and CIFAR-10, with a similar training setup, including early stopping and adaptive learning rates. The key architectural distinction was in the decision-making layer, where a branching structure, analogous to an inception module, allowed the network to process input at three different spatial resolutions. Each branch used progressively larger kernel sizes to capture features of increasing complexity. The Inception-like network was trained over 4, 8, and 100 epochs for the different branches, respectively, with early stopping applied based on validation performance. This training strategy helped in avoiding overfitting while ensuring that each branch could adequately learn the temporal features it was responsible for capturing.

The Inception-like model demonstrated a slight improvement in recognition accuracy over the baseline, achieving approximately 1% better performance on both MNIST and CIFAR-10 datasets. The CIFAR-10 dataset, being more challenging due to its varied object viewpoints and higher complexity, benefited the most from the inception-inspired spatial processing, suggesting that the model's ability to aggregate information from multiple spatial perspectives played a crucial role in improving classification.

The performance improvements were attributed to the capacity of the network to analyze data at different spatial scales, effectively capturing distinct features that could not be easily extracted by a single kernel. This approach draws parallels to ensemble methods, such as ran-

dom forests, where combining multiple models leads to greater robustness and accuracy. The branches acted as specialized paths that processed the input differently, enabling more nuanced decision-making. In Figure 4.8 the training curve compared with the other models.

4.4.3 ResNet-like SNN

In this experiment, we evaluated the ResNet-inspired SNN architecture to explore the benefits of incorporating skip connections for signal preservation and improved learning in deep spiking networks. The key motivation was to address the challenge of signal degradation observed in deeper SNNs and verify whether adding skip connections could improve feature extraction and classification performance.

The ResNet-like SNN architecture builds upon the hierarchical structure of traditional SNNs by introducing skip connections to mitigate the degradation of the signal in deeper layers. The network comprises multiple spiking layers, interleaved with pooling layers, and includes direct connections between non-consecutive layers to maintain information flow. To integrate information between the original and skip pathways, three different methods were explored: logical AND, logical OR, and taking the first spike between the two maps. The best results were obtained using the first spike approach, which provided a clear decision mechanism while preserving the temporal integrity of the spikes. On the other hand, the logical AND and logical OR approaches resulted in unstable and less accurate outcomes, likely due to the noisy and conflicting contributions from both pathways, which disrupted the network's ability to effectively learn and distinguish features.

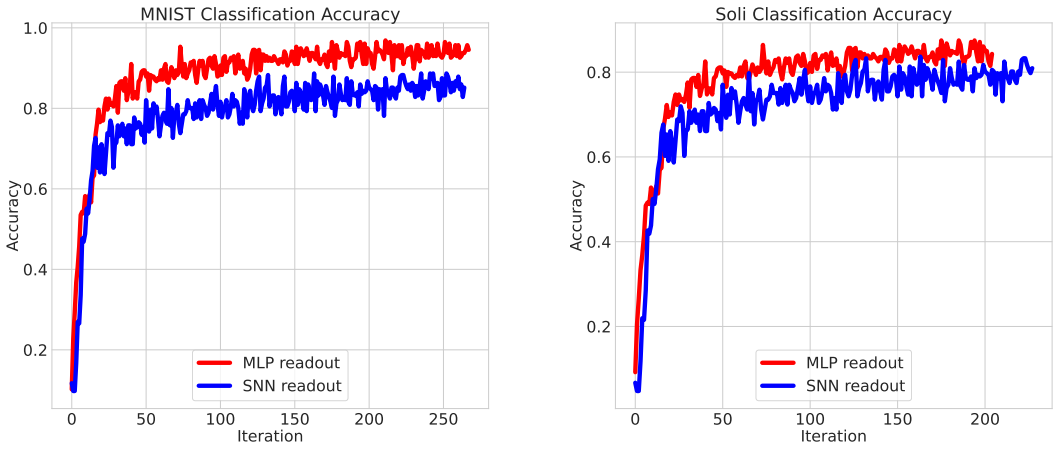
The performance of the ResNet-like SNN was evaluated across the CIFAR-10 and MNIST datasets, comparing its recognition accuracy to baseline spiking networks without skip connections. For CIFAR-10, the addition of skip connections resulted in improved convergence rates and higher accuracy, demonstrating better feature extraction capabilities for this complex dataset. However, the performance on the MNIST dataset showed a slight decrease, possibly due to the simpler nature of the dataset, where the additional complexity introduced by skip connections might not have been necessary. The training curves for different datasets, showing the effect of skip connections on convergence rates, are presented in Figure 4.8.

Training the ResNet-like SNN presented certain challenges, including parameter tuning for the skip connections and balancing the influence of STDP and R-STDP. The skip connections proved beneficial in overcoming the vanishing spike issue, but the optimal configuration required extensive experimentation, particularly for adjusting learning rates and thresholds. Despite these challenges, the ResNet-like architecture demonstrated significant improvements compared to baseline SNN models without skip connections, achieving superior performance for the CIFAR-10 dataset.

4.4.4 Liquid State Machine

The proposed SNN was evaluated on two distinct tasks: digit recognition using the neuromorphic version NMNIST and gesture recognition using the radar-based Soli dataset. The experiments were aimed at understanding the effectiveness of the proposed approach in diverse application domains, ranging from simple digit classification to more complex radar-based gesture recognition.

For the task of digit recognition, the network achieved a notable accuracy of 94.5%. This result indicates a significant improvement over a single shallow multi-layer perceptron (MLP) on the same task, showcasing the ability of the SNN to capture more intricate features from the input data. The improved performance can be attributed to the temporal processing capabilities of the SNN, which effectively encoded the digit information through its spiking dynamics, thereby providing enhanced classification results.



(a) MNIST training dataset

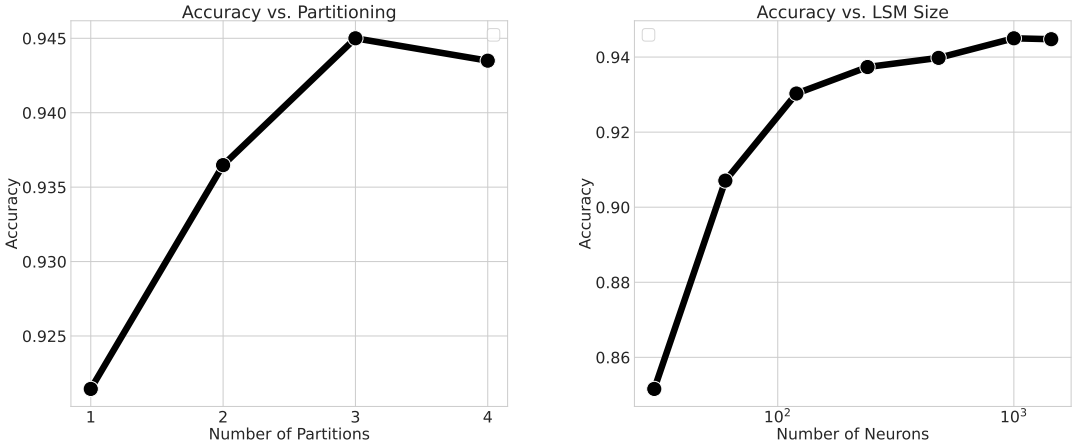
(b) Soli training dataset

Figure 4.5: Comparison of classification accuracy during training for MNIST and Soli datasets using two different readout mechanisms: MLP readout (red) and SNN readout (blue). **(a)** Shows the accuracy evolution for the MNIST dataset, with the MLP readout achieving a higher final accuracy compared to the SNN readout. **(b)** Displays the training results for the Soli dataset, where a similar trend is observed, with the MLP readout maintaining a consistent advantage over the SNN readout.

In the radar-based gesture recognition task, we utilized the Soli dataset to train and evaluate the network's ability to identify different hand gestures based on radar signals. The network achieved an accuracy of 84.3% at plateau, which, while promising, did not reach the high accuracy levels reported by Tsang et al [19]. The lower performance might be due to the challenging nature of radar-based data, which inherently involves complex temporal and spatial correlations that are difficult to capture with the current model configuration. Despite these challenges, the network demonstrated reasonable generalization, indicating its potential utility in gesture recognition applications. Both training curvers are detailed in Figure 4.5.

Furthermore, we tested the use of an external SVM as a classifier to replace the final decision-making layer of the network. The results obtained with the SVM classifier were slightly worse, achieving an accuracy of 79.1%. This decrease in performance highlights the importance of an end-to-end spiking neural architecture, where each component of the network is jointly optimized to achieve the best overall performance, rather than relying on a separate classifier that lacks the temporal dynamics inherent to the rest of the network.

An additional experiment was performed for the evaluation on the effect of two key architectural variations on the performance of the Liquid State Machine. Specifically, we investigated the impact of partitioning the reservoir and scaling the reservoir size. As shown in Figure 4.6, increasing the number of partitions up to three led to a notable improvement in accuracy, while further partitioning resulted in a slight decrease. This suggests that the optimal balance is achieved when sufficient partition complexity exists without overly disrupting inter-region interactions. Similarly, expanding the reservoir size yielded increasing gains in accuracy, with performance leveling off beyond a certain number of neurons, indicating a saturation of the model's representational capacity.



(a) Dependecy on the number of partitions.

(b) Dependency on the size of the reservoir.

Figure 4.6: Impact of reservoir partitioning and size on Liquid State Machine performance. (a) Classification accuracy as a function of the number of partitions within the LSM reservoir. The accuracy peaks at three partitions, highlighting an optimal balance between partition complexity and feature extraction. (b) Relationship between the number of neurons in the LSM and the overall classification accuracy. Larger reservoirs result in improved performance, which stabilizes beyond a certain size, indicating a saturation point in the system's capacity for learning.

These experiments demonstrate that SNNs can be effective for various recognition tasks, with their performance being largely influenced by the temporal characteristics of the data and the nature of the decision-making layer. The results of the gesture recognition experiment suggest potential areas for improvement, such as better handling of complex temporal dynamics, which may require further architectural or methodological advancements.

4.5 Surrogate Gradient-Based Training

In this experiment, we explored the effectiveness of surrogate gradient techniques in training SNNs. Surrogate gradient methods were employed to address the non-differentiability of spiking neurons, enabling gradient-based optimization that closely resembles backpropagation used in traditional artificial neural networks.

The network was trained on the MNIST and CIFAR-10 datasets, following the same training regimen as previous experiments, including early stopping and adaptive learning rates. The optimizer used for training was Adam, with an initial learning rate of 0.01, which was adaptively adjusted using the Reduce on Plateau strategy. No weight decay was applied. The surrogate gradient approach allowed the model to learn from error signals in a more direct and effective manner, improving convergence speed and final performance. Two loss functions were employed during training:

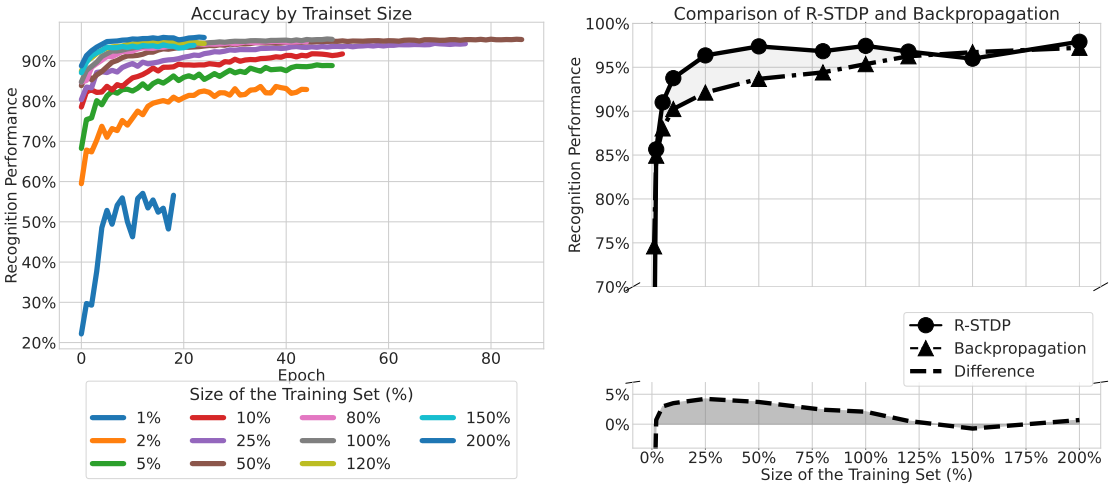
- The cross-entropy loss was used for classification tasks, where the loss was computed based on the discrepancy between predicted and true classes, taking into account the average firing rate of the output neurons. In this context, the cross-entropy loss was adapted to spiking networks by considering the output as the spike rate over a given time window, thereby encouraging correct class prediction through spike activity.
- The MSE loss, on the other hand, was used to optimize spike count behavior, aiming to minimize the difference between the actual and target number of spikes over a defined time frame. This approach effectively trained the network to match desired spike behavior, ensuring that the spiking activity accurately represented the intended output patterns. The use of MSE in this way allowed for fine-tuning of the spiking response, promoting more stable and consistent neuron firing behavior.

The results demonstrated that the surrogate gradient-trained SNN achieved competitive accuracy on both datasets, with notable improvements in convergence rates compared to traditional spiking models trained with unsupervised learning rules. On the MNIST dataset, the model achieved an accuracy of 95.3%.. For CIFAR-10, the model reached an accuracy of 68.3%, indicating that the surrogate gradient method was particularly effective in handling more complex datasets.

4.5.1 Comparison of STDP and Backpropagation

This section compares models trained using STDP and traditional BP, focusing on their performance across different training set sizes. Experiments were conducted using datasets ranging from 1% to 200% of the original size, with the latter achieved through data augmentation techniques. One notable advantage of using STDP in the initial layers of SNNs lies in its ability to effectively extract general features, reducing the risk of overfitting. In contrast, backpropagation

tends to be more data-hungry, requiring larger amounts of training data to achieve satisfactory results. To illustrate this, both models were trained with varying amounts of data. The STDP-based model exhibited a remarkable ability to generalize even when trained with significantly reduced dataset sizes. The results, depicted in Figure 4.7, show how the network regained almost all of its generalization power even with only 20% of the original training samples. On the other hand, the supervised network trained with backpropagation showed a dependency on the quantity of training data, achieving its highest accuracy only when the full dataset was used. This highlights the efficiency of STDP, as it performs well even under data scarcity, making it a more biologically plausible approach. These findings suggest that STDP offers a more data-efficient learning mechanism, drawing parallels to how humans can learn from a limited number of examples. The ability of STDP to retain performance even with small training sets underscores its potential as a viable alternative for training SNNs, particularly in situations where data availability is constrained.



(a) Impact of the size of the trainset on the performances. **(b)** Comparison of STDP and BP depending on the trainset size.

Figure 4.7: Comparison between STDP and BP dependency on the number of training samples. **(a)** Training curves of the models using spike-timing-dependent plasticity and backpropagation for each training set size, demonstrating the resilience of spike-timing-dependent plasticity to data size reduction. **(b)** Comparison of recognition performance between the spike-timing-dependent plasticity-based network and a supervised SNN trained with backpropagation, across restricted training set sizes. The spike-timing-dependent plasticity model exhibits superior generalization capabilities, even with limited training data, as opposed to the backpropagation model.

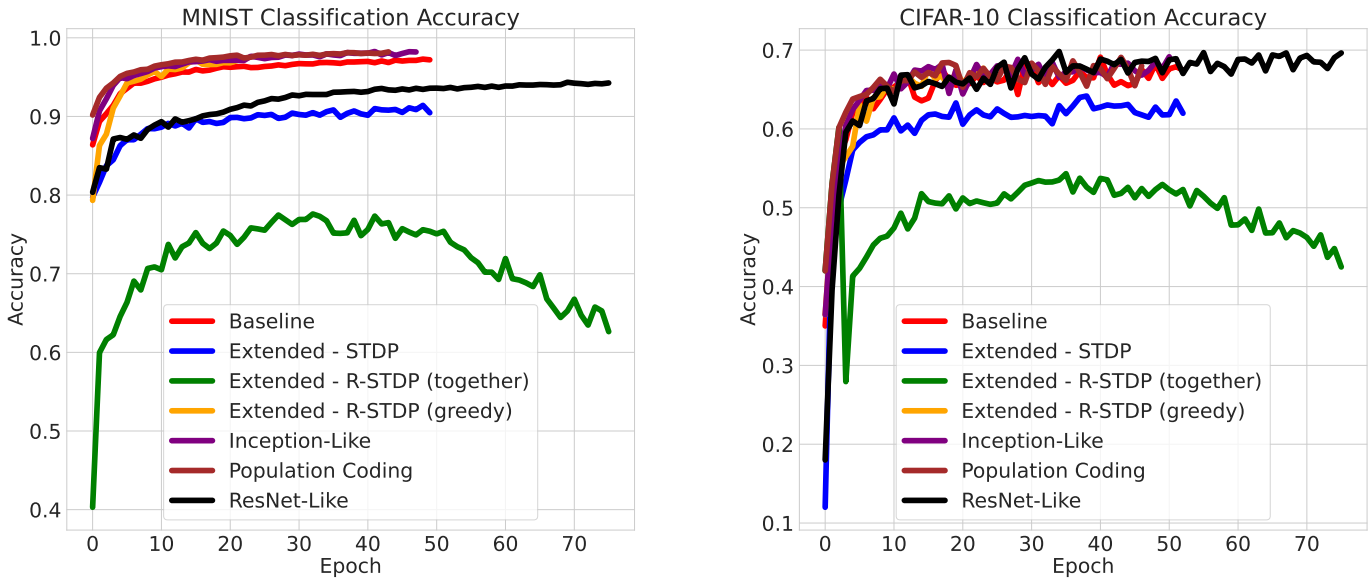
4.6 Summary of the results

This section provides a comprehensive summary of the experimental findings presented in Sections 4.2.2, 4.3, 4.4. Key insights are encapsulated through the training curves in Figure 4.8,

illustrating the convergence behaviour of the proposed model during training, and Table 4.1, highlighting the quantitative performance metrics achieved under varying experimental conditions.

	MNIST				CIFAR-10			
Model	**Accuracy**	**Recall**	**Precision**	**F1**	**Accuracy**	**Recall**	**Precision**	**F1**
Baseline	0.97	0.97	0.97	0.97	0.67	0.68	0.66	0.67
DeepSNN (STDP)	0.91	0.91	0.90	0.91	0.63	0.65	0.62	0.63
DeepSNN (R-STDP - Together)	0.76	0.75	0.76	0.76	0.53	0.48	0.61	0.54
DeepSNN (R-STDP - Greedy)	0.97	0.98	0.97	0.97	0.68	0.70	0.64	0.69
Population coding	**0.98**	**0.97**	**0.98**	**0.98**	0.68	0.69	0.68	0.68
Inception-like	0.98	0.97	0.98	0.97	0.69	0.63	0.72	0.68
ResNet-Like	0.94	0.94	0.93	0.94	**0.70**	**0.71**	**0.70**	**0.70**

Table 4.1: Performance Metrics for Different SNN Architectures on MNIST and CIFAR-10 Datasets



(a) MNIST

(b) CIFAR-10

Figure 4.8: Classification accuracy of the proposed models.

5 Discussion

Mammals exhibit remarkable speed and accuracy in visual object recognition, facilitated by the hierarchical processing in their visual cortex. This architecture progressively enhances the complexity of neuronal responses, resulting in robust and invariant object recognition capabilities. Computational models inspired by this mammalian visual system have been studied extensively to create biologically plausible models that can provide a low-cost platform for hypothesis testing and human-like vision for artificial intelligence.

Deep convolutional neural networks have emerged as highly effective models in this domain. These networks draw inspiration from the hierarchical structure of the visual cortex and have achieved impressive results in visual object recognition. However, their reliance on supervised learning imposes significant limitations, as supervised learning restricts the model to information provided by a teacher and requires substantial data. Furthermore, DCNNs utilize continuous activation functions, which is energy-intensive and resource-consuming, making them less suitable for energy-efficient deployment. Recently, deep reinforcement learning has been proposed to address some of these challenges. In deep reinforcement learning, DCNNs are employed as powerful function approximators to estimate value functions or policies, allowing the agent to effectively map observations to optimal actions. By combining reinforcement learning principles with the representational power of DCNNs, models have been developed that achieve super-human performance, solving problems beyond the capabilities of traditional supervised learning approaches [37, 38]. Despite these advancements, DCNNs still face challenges regarding biological plausibility and hardware efficiency. On the other hand, SNNs, which operate with biologically realistic spiking neurons, provide an alternative to DCNNs that is both energy-efficient and hardware-friendly. SNNs have the potential to close the performance gap with DCNNs, thanks to their rich spatio-temporal capabilities. These net-

works encode information using spike timing rather than firing rates, allowing them to operate with significantly lower energy consumption. Moreover, learning rules such as spike-timing-dependent plasticity and reward-modulated STDP offer biologically inspired mechanisms for synaptic adjustment, enabling SNNs to be implemented efficiently in real-time, low-power hardware. Unlike DCNNs, which rely on high-precision backpropagation for weight updates, SNNs leverage local synaptic modifications driven by spike-timing, making them particularly well-suited for online, on-chip learning where data is dynamic and not available a priori.

Although SNNs have not yet matched the accuracy of DCNNs in many tasks, they exhibit unique advantages in terms of biological plausibility and adaptability. By using STDP to extract regularities from input data and R-STDP to make decisions based on rewarding behavior, SNNs can achieve competitive performance while maintaining energy efficiency.

5.1 Role of Reward-Modulated STDP in Hierarchical SNN Performance

In our experiments, we tackled the visual object recognition task using a hierarchical spiking neural network equipped with a reinforcement learning rule, R-STDP. Reinforcement learning has been shown to be a plausible mechanism for decision-making in the brain, making it well-suited for training neurons to respond selectively to specific classes of input. By employing R-STDP, we developed a biologically plausible model capable of performing visual categorization autonomously. The proposed network operates in the temporal domain, encoding information via spike times. The input image is first convolved with oriented Gabor filters, producing a spike train through a latency-to-intensity coding scheme. The generated spikes are then propagated through the feature extraction layer.

The network effectively discovered task-specific features by using R-STDP to assign specific neurons to particular classes. Each neuron was preassigned to a class, with its intended behavior being to fire early in response to stimuli from that class. To further reduce computational costs, each neuron was restricted to firing at most once per input, with the latency of its spike serving as a measure of its stimulus preference. This approach allowed the earliest spike to determine the network's decision, meaning that when a class-specific neuron fired first, that class was selected as the output.

Comparative experiments between R-STDP and traditional STDP highlighted the efficiency of R-STDP in terms of computational resource utilization. R-STDP successfully addressed a hand-designed discrimination task where the temporal order of spikes was the only distinguishing feature, achieving this with a minimal number of neurons, synapses, and thresholds. In contrast, STDP required more neurons, synapses, and higher thresholds due to its tendency to extract statistically frequent features rather than diagnostic ones. Consequently, using STDP necessitates a greater number of neurons or synapses to eventually identify the critical features. By informing neurons about their outcomes, R-STDP proved capable of achieving effective

performance with fewer computational resources.

To further evaluate the capabilities of R-STDP in conjunction with hierarchical SNNs, we assessed the proposed network against a similar model using STDP and a CNN with the same structure. The evaluations were conducted on three datasets of natural images: Caltech Face / Motorbike, ETH-80 and small NORB, which presented increasing complexity with images taken from multiple viewpoints. The results demonstrated that R-STDP consistently outperformed both STDP and the CNN with a similar structure trained with surrogate gradient. In particular, the proposed network achieved superior performance relying solely on first-spike latencies, even outperforming models that utilized powerful classifiers like support vector machines and error backpropagation.

This comparison between R-STDP and STDP showed that while both networks used similar parameter values, R-STDP provided substantial advantages in terms of efficiency and performance. Although one could potentially use STDP with more neurons and fine-tuned thresholds to improve feature extraction, R-STDP demonstrated a more efficient utilization of resources, making it particularly suitable for practical applications requiring computational efficiency.

The key features of the proposed network include robust object recognition in natural images, the use of a single spike per neuron to significantly reduce energy consumption, and decision-making based on first-spike latencies instead of relying on complex classifiers, thereby enhancing the biological plausibility of the model. Additionally, synaptic plasticity is governed by reinforcement learning via the R-STDP rule, allowing the network to effectively extract highly diagnostic features. These attributes make the proposed network appealing for neuromorphic engineering, given its compatibility with energy-efficient hardware.

5.2 Insights into the Integration of STDP and R-STDP for Improved Classification

We explored the capability of the proposed spiking neural network in solving a 10-class digit recognition problem using the MNIST dataset. The network was designed to utilize STDP in the lower layers to extract regularities in the input, while the higher layers employed R-STDP to achieve decision-making capabilities. This combination allowed the network to autonomously classify input samples without relying on complex external classifiers, thus enhancing biological plausibility and hardware efficiency.

The input images were processed through DoG filters, generating spike latencies using an intensity-to-latency encoding scheme. Spike propagation, as well as synaptic plasticity, occurred layer-wise, with R-STDP-enabled layers being trained concurrently. When the spike wave reached the decision-making layer, the class label was determined based on the maximum potential or the earliest spike time of the neurons. Comparing this decision to the original label allowed the generation of a reinforcement signal, which modulated the plasticity in the R-

STDP-enabled layers. This approach led to a recognition accuracy of 97.2%, surpassing most previously proposed STDP-based SNNs.

The proposed network retained the core advantages of SNNs, such as biological plausibility, energy efficiency, and hardware compatibility. It demonstrated two key features that elevated its energy efficiency compared to traditional DCNNs: first, the use of a spike/no-spike regime allowed energy-efficient accumulation of incoming spikes, second, the use of at most one spike per neuron resulted in minimal energy consumption, as most neurons remained silent during any given input. This event-driven implementation further reduced power requirements, making the proposed network well-suited for practical, low-energy applications. These characteristics also underscore the suitability of the network for neuromorphic hardware implementations where efficiency is paramount.

5.3 Signal Degradation in Deep Architectures

In our experiments, we observed that increasing the depth of SNNs led to a degradation of signal quality, rendering the information less informative at the decision-making layer. This phenomenon was particularly evident in datasets characterized by a lower initial spike rate. We hypothesize that this degradation is related to the principles outlined in the critical brain hypothesis, which posits that neural systems operate near a critical phase transition to optimize information processing capabilities [39].

The critical brain hypothesis suggests that neural networks function optimally at the edge of a phase transition, balancing between order and disorder to maximize computational efficiency and adaptability [41]. In this context, a critical state represents a dynamic regime where the system is neither completely stable nor entirely chaotic, but rather poised between these two extremes. This balance allows for the most efficient information transmission, providing both sensitivity to external stimuli and robustness in response. Operating at criticality enables neural systems to maximize their dynamic range, allowing them to respond to a wide variety of inputs while maintaining the ability to process information effectively. When a network operates in a subcritical regime, characterized by diminished activity propagation, the system may fail to sustain and transmit information effectively. This subcritical state can lead to the attenuation of signals as they traverse through successive layers, resulting in reduced informativeness at the output stage. Conversely, if the network enters a supercritical regime, excessive activity can lead to chaotic behavior, causing a loss of meaningful information due to over-excitation.

As illustrated in Figure 5.1, the Ising model demonstrates the three key phases of system dynamics: the subcritical phase, characterized by diminished activity propagation and insufficient signal transmission; the critical point, where fractal-like patterns emerge and the system achieves optimal sensitivity and information processing; and the supercritical phase, marked by chaotic behavior due to excessive activity. The spatial arrangements of nodes in Figure 5.1(B) clearly depict these differences, while the probability distributions (Figure 5.1(C)) and correlation lengths (Figure 5.1(D)) further emphasize the unique properties of each regime.

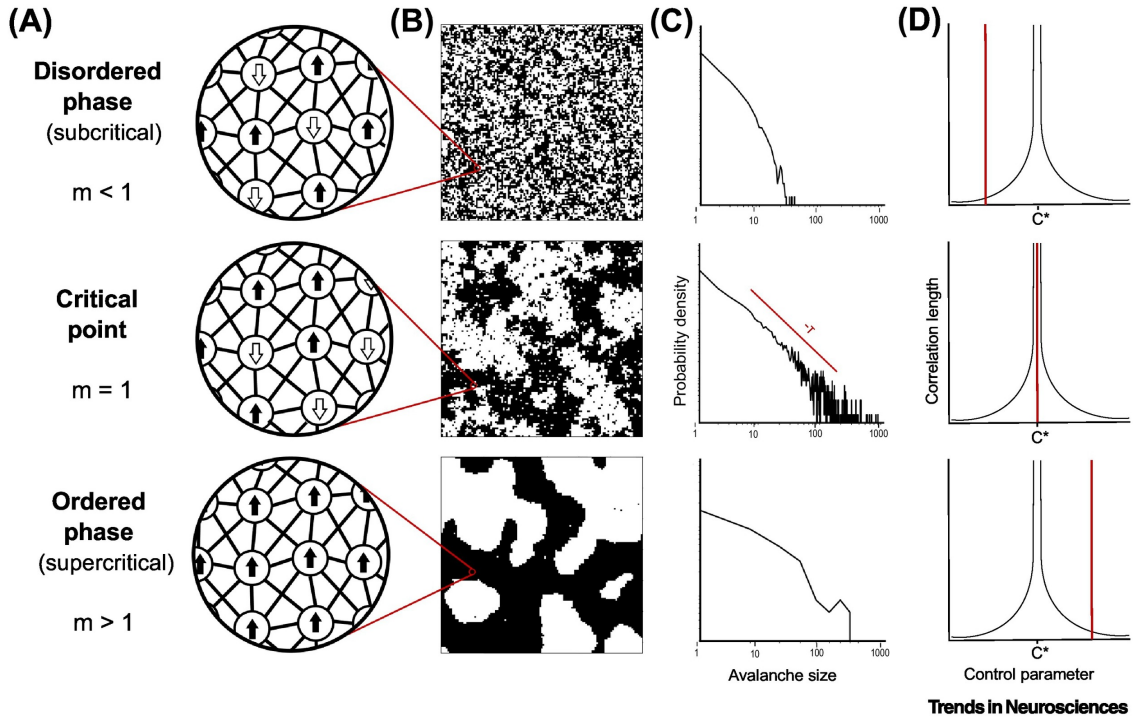


Figure 5.1: The Ising model illustrating different phases of system dynamics. **(A)** The model comprises a network of nodes that can be in one of two states (spin-up or spin-down), representing neural activity. Three distinct phases are shown: disordered (subcritical), critical, and ordered (supercritical). **(B)** The spatial arrangement of node states changes depending on the phase, with the critical point yielding fractal-like patterns of activity. **(C)** Probability distributions of avalanche sizes, illustrating small, large, and power-law-distributed cascades corresponding to disordered, critical, and ordered phases, respectively. **(D)** As the system reaches the critical point, correlation length peaks, indicating maximum sensitivity and optimal information processing. Source: [40]

Our findings align with studies indicating that SNNs not operating near criticality exhibit compromised performance due to insufficient signal propagation and processing [42]. Specifically, in deeper architectures, the cumulative effect of subcritical dynamics may exacerbate signal degradation, as each additional layer introduces potential for further attenuation. This effect is compounded in datasets with inherently low spike rates, where the initial paucity of activity is insufficient to maintain robust signal transmission through the network's depth.

To mitigate these issues, it is imperative to design SNNs that operate near the critical regime, ensuring optimal signal propagation and processing. This approach may involve adjusting network parameters, such as synaptic weights and neuronal thresholds, to promote dynamics that support sustained and efficient information flow across all layers. By aligning network operation with the principles of the critical brain hypothesis, we can enhance the performance and robustness of deep spiking neural network architectures.

5.4 Role of sophisticated architectures

In our experiments, we investigated a range of sophisticated architectural designs inspired by traditional artificial neural networks, including inception modules and residual connections. These designs aimed to enhance the baseline models by incorporating deeper structures and more intricate pathways for information flow. While the advanced models did achieve slight improvements over the baseline, the overall gains were relatively modest, especially compared to the transformative impact such modifications have in conventional artificial neural networks.

In traditional ANNs, increasing depth and adding sophisticated components, such as inception blocks or residual connections, often result in substantial improvements in model performance, as can be seen in Figure 5.2, transforming shallow networks into highly capable models for solving complex tasks. However, our findings indicate that, for SNNs, these architectural enhancements do not deliver the same level of transformative improvement.

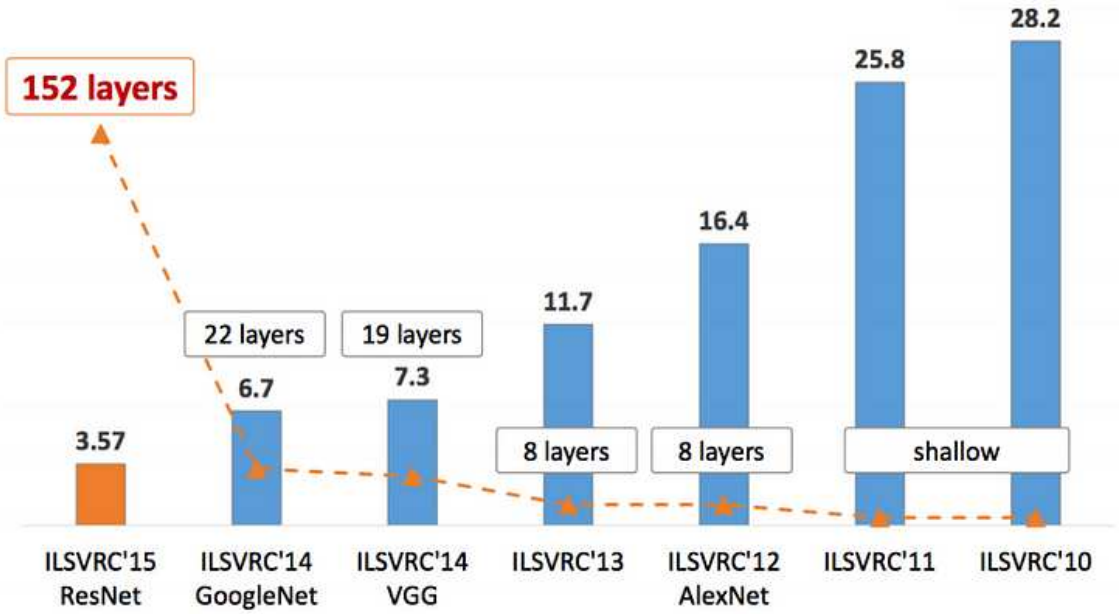


Figure 5.2: Progression of ImageNet Challenge Top-5 Error Rates Over Time: This chart illustrates the evolution of neural network architectures from 2010 to 2015, showcasing significant improvements in accuracy as models grew deeper. Starting with shallow architectures in 2010, advancements through AlexNet (2012) [43], VGG (2014) [44], GoogleNet (2014) [45], and culminating with the 152-layer ResNet (2015) [46] reveal a trend toward increased depth, leading to a dramatic reduction in error rates, with ResNet achieving a top-5 error rate of only 3.57%. Source: [47]

One possible explanation for this disparity lies in the fundamental differences in training methodologies. Unlike backpropagation, which provides informative gradient updates to all layers of a conventional ANN, SNNs typically rely on unsupervised or semi-supervised training mechanisms. These training approaches do not facilitate the same degree of inter-layer information exchange, which is crucial for coherent and effective feature extraction. Consequently,

the addition of more layers or sophisticated structures does not yield the dramatic performance gains observed in traditional deep learning approaches.

Instead, we observed that the performance gains achieved by adding more sophisticated structures in spiking networks were incremental, with diminishing returns as the model complexity increased. This outcome suggests that, for SNNs, simply increasing depth may not be the optimal strategy for significantly enhancing performance. Given the unsupervised or reinforcement-based nature of the training, it appears more likely that the bottleneck lies in the training mechanisms themselves, particularly in how learning signals are propagated and how different layers communicate.

As a result, future work might benefit from exploring alternative strategies beyond increasing depth, such as improving synaptic plasticity mechanisms, enhancing neuromodulatory signals, or optimizing inter-layer communication. These approaches may prove more effective in enabling SNNs to reach their full potential, rather than focusing solely on the depth and sophistication of the architecture.

5.5 On the catastrophic forgetting issue

Catastrophic forgetting is a well-known issue in traditional artificial neural networks, where the model loses previously learned knowledge upon encountering new data, especially in sequential learning scenarios. In classical ANNs, this problem arises due to the way backpropagation updates weights globally across the network, often overwriting important information learned during earlier tasks.

In contrast, SNNs trained with STDP/R-STDP exhibit increased resilience to catastrophic forgetting. This can be attributed to the localized and activity-dependent nature of these learning rules. STDP adjusts synaptic strengths based on the precise timing of pre- and post-synaptic spikes, effectively capturing and retaining temporal patterns in the data. By focusing on localized synaptic changes that depend on specific neural activities, STDP helps to prevent the overwriting of previously acquired features, thereby preserving prior learning even when new tasks are introduced.

R-STDP further extends this resilience by incorporating a reinforcement learning component, which guides synaptic changes based on external reward signals. This mechanism allows the network to adapt more flexibly, selectively reinforcing useful synaptic modifications that are beneficial for task performance while reducing the likelihood of unintentional interference with previously learned knowledge. This selective reinforcement process helps the network retain important information over time, particularly in scenarios where multiple, distinct tasks are learned sequentially.

Another factor contributing to the robustness of STDP and R-STDP against catastrophic forgetting is the inherent sparsity in spiking networks. In SNNs, neurons are typically sparsely activated, meaning that only a subset of neurons and synapses are engaged at any given moment. This sparse activation helps limit the extent of synaptic updates during learning, reducing the

risk of overwriting critical information stored across the network. As a result, different tasks are more likely to be encoded by distinct sets of synapses, further mitigating the effects of catastrophic forgetting.

In our experiments, both STDP and R-STDP demonstrated superior performance in retaining learned information across multiple tasks compared to trained classical ANN models.

5.6 Challenges in Hyperparameter Selection

One of the primary challenges we faced throughout our experiments was the difficulty in determining appropriate hyperparameters. This is a pervasive issue in the design and training of SNNs, which are inherently more sensitive to the choice of hyperparameters compared to traditional artificial neural networks. Unlike classical ANNs, where hyperparameter selection may be more forgiving and incremental changes often result in gradual performance differences, SNNs exhibit a far more brittle nature. Small deviations in certain parameters can lead to dramatic swings in performance, sometimes resulting in catastrophic failure.

This sensitivity complicates the training process significantly. Parameters such as the firing thresholds, the lateral inhibition radii, the number of synapses eligible for modification, and the learning rates for both STDP and R-STDP are particularly crucial. During our experiments, we observed that even slight adjustments in these hyperparameters could lead to instability in the network. For instance, if the firing thresholds was not correctly calibrated, as discussed in 5.3, neurons could either become inactive—failing to respond to input patterns—or enter a state of hyperactivity, where their firing becomes uncoordinated and no meaningful information can be extracted. Similarly, tuning the lateral inhibition radius and the number of synapses eligible for plastic changes was essential to ensure that the network retained useful selectivity without collapsing into over-generalized or over-specialized states. The learning rates of STDP and R-STDP also played a critical role in network stability. A learning rate too high would often result in oscillatory or divergent behavior, while a rate too low would cause the learning process to stagnate, failing to adapt the network effectively to the input patterns. These hyperparameters require precise adjustment to prevent the network from degrading, highlighting the challenging balance between network stability and adaptability.

In addition, the inherent characteristics of SNNs make generalization particularly challenging. The networks tend to be highly sensitive to input conditions, resulting in behavior that may not generalize well across different tasks or datasets without considerable manual tuning. This user-unfriendliness contrasts sharply with classical ANNs, which, at least for relatively simple tasks, can often be configured with reasonable default hyperparameters and still achieve satisfactory performance. The ability of classical ANNs to generalize well with default settings makes them accessible to a broader audience, whereas the steep learning curve associated with SNNs demands an in-depth understanding of both the network dynamics and the biological mechanisms being modeled.

The difficulties in hyperparameter selection suggest that more work is required to develop systematic approaches for configuring SNNs. In particular, the development of more robust theoretical frameworks could provide a foundation for setting these parameters. Drawing inspiration from biological systems may offer a pathway forward—certain "rigid" rules derived from biological observations could help guide the selection process, reducing reliance on trial-and-error and improving the stability of the networks. For example, biological neurons operate under well-defined firing thresholds and synaptic plasticity rules that have evolved to balance stability with learning flexibility. Adopting such biologically inspired constraints could mitigate the extreme sensitivity to parameter settings that currently limits the practical application of SNNs.

5.7 Reservoir Computation and Versatility

The Liquid State Machine architecture demonstrates a remarkable ability to project input data into a high-dimensional space, in accordance with reservoir computing theory. The fundamental idea behind reservoir computing is that a fixed, recurrent network—the reservoir—acts as a dynamic, nonlinear projection unit that enriches the representation of incoming data. This transformation enables simpler, linear readout mechanisms to perform tasks such as classification or regression with high accuracy, making liquid state machines an effective approach for solving complex problems without explicit training of the reservoir itself.

One of the most notable characteristics of this model is the absence of a need for training of the reservoir. Unlike other neural architectures that require detailed parameter tuning across multiple layers, liquid state machines leverage randomly initialized connections within the reservoir, effectively bypassing the challenges associated with intricate hyperparameter optimization. This feature contributes to the model being capable of solving a wide variety of tasks with minimal tuning requirements. The fixed reservoir, combined with the versatility of the readout layer, allows for efficient adaptability across tasks with very different data types, ranging from time-series data to image processing. The liquid's dynamic response to input data captures both spatial and temporal information, which is critical for handling sequential and real-time data. As spikes propagate through the reservoir, they interact in a complex, nonlinear manner, creating a diverse set of high-dimensional features. This richness of features directly contributes to the efficacy of the readout layer, which can then be trained using classical methods. This temporal representation helps the system maintain memory of previous states, thereby enhancing its suitability for time-sensitive applications, such as gesture recognition or speech processing.

One of the key advantages of liquid state machines lies in their universality and flexibility. Since the reservoir operates as a fixed, complex dynamic system, it is capable of mapping inputs into distinct trajectories through its high-dimensional state space. The ability to work across a wide spectrum of domains makes this approach versatile, especially when compared to traditional neural networks, where the architectural design and hyperparameter tuning often need

to be highly specialized for each task. Additionally, due to the inherent richness of the internal states produced by the reservoir, liquid state machines are capable of managing task diversity without substantial structural changes.

While the readout layer is the only trainable component, its ability to leverage the high-dimensional states is directly influenced by the quality of the reservoir's dynamics. This dependency implies that the initial design of the reservoir, including its connectivity, sparsity, and overall structure, still plays a crucial role in the model's success. Although liquid state machines can generalize across various tasks, designing the reservoir to ensure a suitable balance between memory capacity and computational power is essential to achieve optimal results. Poorly designed reservoirs may yield underwhelming feature richness, resulting in a readout layer that struggles to generalize effectively.

Despite these challenges, liquid state machines have proven to be robust for tasks involving temporal data, showing resilience to minor input variations due to the stochastic nature of their connectivity. This robustness, alongside the absence of complex backpropagation-based training through recurrent layers, gives liquid state machines a distinct edge in terms of computational efficiency. The event-driven nature of SNNs, where activations are based on discrete spikes, further contributes to the energy efficiency of these models, making them suitable for low-power, neuromorphic hardware implementations.

The experiments conducted on liquid state machines in this work highlight their generalization capabilities, which stem from projecting the input into a richer feature space without specific training of the internal dynamics. The ability of these models to adapt and provide a meaningful separation of input features for classification, as evidenced by the performance metrics, supports their application to a diverse set of tasks. However, it is also evident that while liquid state machines provide flexibility and efficiency, the lack of trainable weights within the reservoir may limit their adaptability compared to more sophisticated deep learning models, which can leverage end-to-end learning to optimize performance.

6 Conclusion

The objective of this thesis was to explore and evaluate the use of SNNs for complex recognition tasks, employing biologically inspired learning rules such as STDP and reward-modulated STDP. Through extensive experiments, we examined the baseline models and extended their capabilities with more sophisticated architectures, including population coding, inception-like, and ResNet-like structures, alongside the use of Liquid State Machines.

Our key findings demonstrate that the baseline models with spike timing dependent plasticity and reward-modulated STDP achieved competent performance on visual object recognition tasks, including digit recognition (MNIST), object recognition (CIFAR-10) and gesture classification (Soli). The use of reward-modulated STDP, particularly, allowed the network to learn with minimal supervision, providing a biologically plausible alternative to classical supervised methods. The performance improvement compared to spike timing dependent plasticity highlights the effectiveness of reinforcement-modulated learning in reducing computational resource usage while ensuring robust feature extraction.

The introduction of advanced architectures, such as population coding, inception-like, and ResNet-like structures, yielded moderate performance gains. Unlike in traditional artificial neural networks, where adding depth and sophistication often leads to radical performance improvements, these modifications did not produce transformative results. This suggests that, in the context of unsupervised or semi-supervised SNNs, increasing depth may not be the optimal strategy. Instead, alternative approaches, such as improved inter-layer communication or hybrid training mechanisms, could hold more promise.

The Liquid State Machine model showcased its capability to project input data into a higher-dimensional space effectively, aligning well with the reservoir computing theory. Its untrained reservoir structure allows it to solve a wide range of tasks without requiring extensive parame-

ter tuning, highlighting its robustness. However, the overall performance of Liquid State Machines still falls short compared to classical methods when dealing with highly complex tasks, indicating room for further exploration in optimizing their reservoir dynamics.

One of the significant challenges encountered during this research was the sensitivity of SNNs to hyperparameter selection. Factors such as firing rate, lateral inhibition radius, synaptic modification constraints, and learning rates for spike timing dependent plasticity and reward-modulated STDP had a profound impact on the network's performance. Minor changes to these parameters could lead to drastic variations in results, including catastrophic forgetting. This sensitivity makes generalization difficult and highlights the need for more rigid theoretical frameworks to guide parameter selection, potentially inspired by biological systems.

Our experiments demonstrated that SNNs, through spike timing dependent plasticity and reward-modulated STDP, are less susceptible to catastrophic forgetting compared to classical artificial neural networks. The localized learning mechanism of spike timing dependent plasticity and the reinforcement-guided modulation of reward-modulated STDP allow individual layers to retain learned features without the need for continuous feedback across layers. This attribute contrasts sharply with classical backpropagation-based methods, where continuous weight updates can result in the overwriting of previously learned information.

This work has contributed to the understanding of SNNs and their potential applications in practical AI tasks. While SNNs offer distinct advantages in terms of biological plausibility, energy efficiency, and robustness to catastrophic forgetting, several challenges remain. The complexity of hyperparameter tuning and the limited gains from deeper architectures suggest that future research should focus on developing new training mechanisms and leveraging hybrid approaches. By combining the strengths of biologically inspired learning and modern AI techniques, there is potential to unlock new capabilities in neuromorphic computing, making it an exciting field for future exploration.

There are several promising directions for future exploration in the field of SNNs. First, we aim to further advance SNN architectures by leveraging key findings from neuroscience and biology. This could help transition from a trial-and-error approach to one based on more solid theoretical underpinnings, ultimately improving the stability and performance of SNNs. Moreover, we plan to explore more sophisticated architectural enhancements. This includes mechanisms such as neurogenesis (the creation of new neurons) and synaptic pruning (the removal of weak synapses), both of which could dynamically adjust the network's structure to improve efficiency and learning. Additionally, multi-scale temporal coding is another avenue we wish to explore, by employing different time windows for different sets of neurons to capture both fast and slow temporal dynamics, ultimately combining the outputs from these neurons to form a comprehensive representation. We aim to experiment with different reinforcement learning strategies and investigate the use of spiking attention mechanisms to enhance the decision-making capabilities of SNNs. A key aspect of our future research will also focus on the development of interpretability tools, designed to quantify the importance of individual neurons or synapses in decision-making processes, thereby providing insights into how

information is processed within the network.

We also aspire to integrate some of our work into established libraries, such as snnTorch, which would facilitate more robust and accessible frameworks for future research. Finally, implementing these models in neuromorphic hardware is an essential step towards practical, real-world deployment of SNNs as it would fully exploit the spiking nature of these networks and yield energy-efficient implementations, ultimately bridging the gap between biological plausibility and modern AI applications.

References

[1] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in Neural Circuits*, vol. 9, 2016. [Online]. Available: https://www.frontiersin.org/journals/neural-circuits/articles/10.3389/fncir.2015.00085

[2] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike-based visual categorization using reward-modulated stdp," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 6178–6190, Dec 2018.

[3] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, "Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks," *Pattern Recognition*, vol. 94, pp. 87–95, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320319301906

[4] K. Yamazaki, V.-K. Vo-Ho, D. Bulsara, and N. Le, "Spiking neural networks and their applications: A review," *Brain Sciences*, vol. 12, no. 7, 2022. [Online]. Available: https://www.mdpi.com/2076-3425/12/7/863

[5] J. D. Nunes, M. Carvalho, D. Carneiro, and J. S. Cardoso, "Spiking neural networks: A survey," *IEEE Access*, vol. 10, pp. 60 738–60 764, 2022.

[6] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[7] E. R. Kandel, J. D. Koester, S. H. Mack, and S. A. Siegelbaum, *The Brain and Behavior*. New York, NY: McGraw Hill, 2021. [Online]. Available: neurology.mhmedical.com/content.aspx?aid=1180639279

[8] A. Biswas, N. S Nambiar, K. Kejriwal, and U. Ganguly, "Madapter: A multimodal adapter for liquid state machines configures the input layer for the same reservoir to enable vision and speech classification," in *2023 International Joint Conference on Neural Networks (IJCNN)*, 2023, pp. 1–6.

[9] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. USA: Cambridge University Press, 2014.

[10] M. C. W. van Rossum, G. Q. Bi, and G. G. Turrigiano, "Stable hebbian learning from spike timing-dependent plasticity," *Journal of Neuroscience*, vol. 20, no. 23, pp. 8812–8821, 2000. [Online]. Available: https://www.jneurosci.org/content/20/23/8812

[11] C. Lee, P. Panda, G. Srinivasan, and K. Roy, "Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning," *Frontiers in Neuroscience*,

vol. 12, 2018. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00435

[12] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[13] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, "Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron," *Frontiers in Neuroscience*, vol. 13, 2019. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2019.00625

[14] H. Irmak, F. Corradi, P. Detterer, N. Alachiotis, and D. Ziener, "A dynamic reconfigurable architecture for hybrid spiking and convolutional fpga-based neural network designs," *Journal of Low Power Electronics and Applications*, vol. 11, p. 32, 08 2021.

[15] W. Maass, T. Natschläger, and H. Markram, "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 11 2002. [Online]. Available: https://doi.org/10.1162/089976602760407955

[16] W. Maass, *Liquid State Machines: Motivation, Theory, and Applications*. World Scientific Publishing Co Pte Ltd, 2011, pp. 275–296. [Online]. Available: https://worldscientific.com/doi/abs/10.1142/9781848162778_0008

[17] W. Cai, F. Ellinger, and R. Tetzlaff, "Neuronal synapse as a memristor: Modeling pair- and triplet-based stdp rule," *IEEE transactions on biomedical circuits and systems*, vol. 9, 06 2014.

[18] D. Izzo, A. Hadjiivanov, D. Dold, G. Meoni, and E. Blazquez, "Neuromorphic computing and sensing in space," 2022. [Online]. Available: https://arxiv.org/abs/2212.05236

[19] I. J. Tsang, F. Corradi, M. Sifalakis, W. Van Leekwijck, and S. Latré, "Radar-based hand gesture recognition using spiking neural networks," *Electronics*, vol. 10, no. 12, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/12/1405

[20] A. Patino-Saucedo, H. Rostro-Gonzalez, T. Serrano-Gotarredona, and B. Linares-Barranco, "Liquid state machine on spinnaker for spatio-temporal classification tasks," *Frontiers in Neuroscience*, vol. 16, p. 819063, 03 2022.

[21] R. Legenstein, D. Pecevski, and W. Maass, "A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback," *PLOS Computational Biology*, vol. 4, no. 10, pp. 1–27, 10 2008. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1000180

[22] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:53086859

[23] P. Joshi and W. Maass, "Movement generation and control with generic neural microcircuits," in *Biologically Inspired Approaches to Advanced Information Technology*, vol. 3141, 01 2004, pp. 258–273.

[24] 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, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1254642

[25] E. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov 2003.

[26] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.

[27] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[28] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. II–104, 2004.

[29] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18268744

[30] F.-F. Li, M. Andreeto, M. Ranzato, and P. Perona, "Caltech 101," Apr 2022.

[31] J. Lien, N. Gillian, M. E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev, "Soli: ubiquitous gesture sensing with millimeter wave radar," *ACM Trans. Graph.*, vol. 35, no. 4, Jul. 2016. [Online]. Available: https://doi.org/10.1145/2897824.2925953

[32] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.

[33] K. Wiggers, "Google unveils pixel 4 and pixel 4 xl with gesture recognition and dual rear cameras," https : / / venturebeat . com / 2019 / 10 / 15 / google-unveils-the-pixel-4-and-pixel-4-xl-with-gesture-recognition-and-dual-rear-cameras, 2019, accessed: 2024-11-07.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[35] Y. Liu, S. Yenamachintala, and P. Li, "Energy-efficient fpga spiking neural accelerators with supervised and unsupervised spike-timing-dependent-plasticity," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 15, pp. 1–19, 05 2019.

[36] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," 2023. [Online]. Available: https://arxiv.org/abs/2109.12894

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:205242740

[38] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:515925

[39] J. Beggs, "The criticality hypothesis: How local cortical networks might optimize information processing," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 366, pp. 329–43, 08 2007.

[40] J. O'Byrne and K. Jerbi, "How critical is brain criticality?" *Trends in Neurosciences*, vol. 45, no. 11, pp. 820–837, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166223622001643

[41] D. R. Chialvo, "Emergent complex neural dynamics," *Nature Physics*, vol. 6, no. 10, p. 744–750, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1038/nphys1803

[42] O. Kinouchi and M. Copelli, "Optimal dynamical range of excitable networks at criticality," *Nature Physics*, vol. 2, no. 5, p. 348–351, Apr. 2006. [Online]. Available: http://dx.doi.org/10.1038/nphys289

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[45] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

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

[47] Z. Jiang, "A novel crop weed recognition method based on transfer learning from vgg16 implemented by keras," *IOP Conference Series: Materials Science and Engineering*, vol. 677, p. 032073, 12 2019.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Prof. Michele Rossi, whose guidance throughout the course of my thesis work has been invaluable. I am also deeply thankful to my co-supervisors Riccardo Mazzieri and Eleonora Cicciarella for their consistent support, insight, and encouragement throughout the various stages of this project.

I extend my gratitude to Michele Allegra, my co-examiner, for his valuable feedback and constructive comments that significantly improved the quality of this work.

Lastly, I would like to thank my family and friends for their unwavering support and motivation during this challenging journey.