TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM School of Computation, Information and Technology

Optimizing the Conversion of Continuous-Valued Networks to Spiking Neural Networks

Etienne Müller

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:	Prof. Dr.-Ing. Alin Albu-Schäffer
Prüfer der Dissertation:	1. Prof. Dr.-Ing. habil Alois Christian Knoll
	2. Prof. Dr. rer. nat. Markus Diesmann

Die Dissertation wurde am 02.05.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 02.09.2024 angenommen.

Abstract

Deep learning has significantly advanced the field of AI in the past decade, with powerful parallel computing hardware and new approaches in ANNs enabling the creation of very deep architectures that frequently exhibit superhuman performance. Despite this success, energy consumption can be problematic in various contexts, such as large server applications with independent power generation, advanced driver assistance systems in cars with limited battery capacity, and small embedded systems with restricted power budgets.

One potential solution to this issue is neuromorphic computing, which is inspired by biological neurons and has the potential to reduce energy consumption through the use of SNNs. SNNs communicate with short all-or-nothing pulses rather than the continuous-valued activation functions used in traditional ANNs. While numerous approaches have been proposed for training SNNs, none have achieved performance comparable to ANNs trained with highly optimized gradient descent-based learning algorithms. As a result, the current state-of-the-art involves taking pre-trained ANNs and converting them into SNNs.

This thesis aims to analyze and compare the properties of ANNs and SNNs, review and optimize existing conversion approaches, and enable the development of more effective and efficient neuromorphic systems. We focus on the three fundamental encoding techniques: *rate coding*, *population coding*, and *temporal coding*. Through our analysis, we developed optimization techniques for increasing inference speed and approximating hyperparameters in rate-coded networks to construct the deepest SNN to date with more than 100 layers, providing the basis for a spiking transformer network. Furthermore, we propose a novel approach for sparse, low-energy computation in temporal-coded networks and demonstrate previously infeasible time-series processing through population coding by exploiting the inhomogeneities of analog neuromorphic hardware. By thoroughly exploring the possibilities for conversion between ANNs and SNNs, we hope to make significant progress toward enabling the widespread adoption of neuromorphic computing and reducing the energy consumption of AI systems.

Zusammenfassung

Deep Learning hat den Bereich der künstlichen Intelligenz (KI) in den letzten zehn Jahren erheblich vorangebracht. Leistungsstarke parallele Computerhardware und neue Ansätze bei künstlichen neuronalen Netzen (ANNs) ermöglichen die Erstellung von sehr tiefen Architekturen, die häufig übermenschliche Leistungen erbringen. Trotz dieses Erfolgs kann der Energieverbrauch in verschiedenen Kontexten problematisch sein, z. B. bei großen Serveranwendungen mit unabhängiger Stromerzeugung, bei fortschrittlichen Fahrerassistenzsystemen in Autos mit begrenzter Batteriekapazität und bei kleinen eingebetteten Systemen mit eingeschränktem Energiebudget.

Eine mögliche Lösung für dieses Problem sind neuromorphe Systeme, die von biologischen Neuronen inspiriert sind und das Potenzial haben, den Energieverbrauch durch den Einsatz von gepulsten neuronalen Netzen (SNNs) zu senken. SNNs kommunizieren mit kurzen Alles-oder-Nichts-Impulsen anstelle der in herkömmlichen ANNs verwendeten Aktivierungsfunktionen mit kontinuierlichen Werten. Es wurden zahlreiche Ansätze für das Training von SNNs vorgeschlagen, aber keiner hat bisher eine Leistung erreicht, die mit ANNs vergleichbar ist, die mit hoch optimierten, auf gradientenbasierenden Lernalgorithmen trainiert werden. Der derzeitige Stand der Technik besteht daher darin, vortrainierte ANNs in SNNs zu konvertieren.

Ziel dieser Arbeit ist es, die Eigenschaften von ANNs und SNNs zu analysieren und zu vergleichen, bestehende Konvertierungsansätze zu überprüfen und zu optimieren und die Entwicklung effektiverer und effizienterer neuromorpher Systeme zu ermöglichen. Wir konzentrieren uns auf die drei grundlegende Kodierungstechniken: *Ratenkodierung*, *Populationskodierung* und *zeitliche Kodierung*. Durch unsere Analyse haben wir Optimierungstechniken zur Erhöhung der Inferenzgeschwindigkeit und zur Annäherung von Hyperparatmetern in ratenkodierten Netzwerken entwickelt, um das zu der Zeit tiefste gepulste neuronale Netzwerk mit mehr als 100 Schichten zu konstruieren sowie die Grundlage für gepulste Transformer Netzwerk zu schaffen. Darüber hinaus schlagen wir einen neuartigen Ansatz für spärliche, energiesparende Berechnungen in zeitlich kodierten Netzwerken vor und demonstrieren die bisher nicht mögliche Verarbeitung von zeitlich kontinuierlichen Daten durch Populationskodierung, indem wir die Inhomogenitäten analoger neuromorpher Hardware ausnutzen. Durch die gründliche Erforschung der Möglichkeiten für die Umwandlung zwischen ANNs und SNNs hoffen wir, bedeutende Fortschritte auf dem Weg zu einer weit verbreiteten Einführung des neuromorphen Rechnens und zur Verringerung des Energieverbrauchs von Systemen der KI zu erzielen.

Acknowledgments

First of all, I would like to express my gratitude to Professor Alois Knoll, who made this research possible. He initiated this close collaboration between industry and academia, for which I am genuinely grateful. Additionally, he provided discussions and guidance throughout the whole project. Without that, a successful conclusion of the research project would not have been possible.

The research project was funded by Infineon Technologies AG. I am immensely grateful for this support. Representatively, I would like to thank Dr. André Roger; also for being my mentor and helping me during every stage of the project.

Next, I would like to thank the academic and administrative staff at the Chair of Robotics, Artificial Intelligence, and Real-time Systems: Dr. Alexander Lenz, Ute Lomp, and Amy Bücherl. They provided a positive and supportive working environment, no matter the circumstances or the kind of questions I had. My gratitude also goes to Dr. Sladjana Martens, who had my back during my last year at the chair and provided me with valuable support for my future steps.

Furthermore, especially during these times of a global pandemic, I would like to thank my colleagues, who created an enjoyable environment that promoted new ideas and the motivation to pursue further research directions. In particular, Daniel Auge, without whom I would have had a much tougher start in this new stage of life, and Emeç Ercelik, Sina Shafei and Christoph Segler, for the many discussions, tips and tricks, and assistance, mostly digital but sometimes in person at a good barbecue.

Lastly, I would like to thank my family: Maman, Uli, Mamie and especially my sister Sandrine, who inspired me to go down this path and stayed a motivation throughout.

My life – and this dissertation – would not be the same without you!

Contents

List of Figures **xiii**

List of Tables **xv**

List of Acronyms **xvii**

List of Symbols **xxi**

1 Introduction **1**
- 1.1 Research Questions and Scope of the Thesis 4
- 1.2 Structure 5
- 1.3 Contributions 8

2 Theoretical Background **11**
- 2.1 Neuron Models 12
  - 2.1.1 Biologically-plausible Models 14
  - 2.1.2 Biologically-inspired Models 16
  - 2.1.3 Integrate-and-Fire Models 16
- 2.2 Neuromorphic Hardware 21
  - 2.2.1 Software Simulation Environments 22
  - 2.2.2 Digital Implementations 22
  - 2.2.3 Analog and Mixed Analog/Digital Implementations 25
  - 2.2.4 Summary 27
- 2.3 Training of Spiking Networks 28
  - 2.3.1 Biologically Plausible Learning Algorithms 28
  - 2.3.2 Supervised Learning with Spikes 30
  - 2.3.3 Conversion Methods 30

3 Conversion Challenges and Opportunities **35**
- 3.1 Coding Schemes 36
  - 3.1.1 Rate Coding 38
  - 3.1.2 Population Coding 40
  - 3.1.3 Temporal Coding 41
  - 3.1.4 Conclusion 46

3.2 Properties of Neural Networks . . . 47
3.2.1 Tasks and Datasets . . . 48
3.2.2 Architectures . . . 49
3.3 Weight Normalization . . . 55
3.3.1 Basic Conversion . . . 55
3.3.2 Hyperparameter Search . . . 55
3.3.3 ReLU1 for Prototyping . . . 57

4 Rate-Coded Conversion 59
4.1 Optimization Methods . . . 60
4.1.1 Related Work: Voltage Clamping . . . 60
4.1.2 Related Work: Channel-Wise Normalization . . . 61
4.1.3 Unbiased Quantization . . . 63
4.2 Deep Spiking CNN for Classification . . . 67
4.2.1 Training and Conversion . . . 67
4.2.2 Conversion Evaluation . . . 68
4.3 Spiking FPN for Object Detection . . . 73
4.3.1 Training of RetinaNets . . . 73
4.3.2 Evaluation of the Results . . . 74
4.4 Spiking Transformer Networks . . . 77
4.4.1 Conversion Approach . . . 77
4.4.2 NLP Evaluation . . . 78
4.4.3 Image Classification Evaluation . . . 80

5 Population-Coded Conversion 85
5.1 Related Work: Conversion of Elman RNNs . . . 86
5.2 Methodology . . . 86
5.2.1 Spiking Sigmoid . . . 87
5.2.2 Spiking Tanh . . . 88
5.3 Conversion Evaluation . . . 91
5.3.1 Dataset . . . 91
5.3.2 Network Architecture . . . 91
5.3.3 Conversion of the LSTM's Gates and States . . . 92
5.3.4 Conversion of the Entire LSTM Cell . . . 94
5.4 Discussion of the Results . . . 95

6 Temporal-Coded Conversion 97
6.1 Related Work: TTFS-coded Conversion . . . 98
6.2 Methods . . . 99
6.2.1 Neural Oscillations . . . 100

6.2.2 Spike Amplitude Adaption 103
6.2.3 Pulse Duration Adaption 103
6.3 Conversion Evaluation 104
6.3.1 Phase Coding Compared to TTFS Coding 104
6.3.2 Phase Coding Compared to Rate Coding 105
6.4 Discussion of the Results 106

7 Summary and Conclusion 109
7.1 Theoretical Background 109
7.2 Conversion Challenges and Opportunities 110
7.3 Methods and Evaluation 111
7.3.1 Rate-Coded Conversion 111
7.3.2 Population-Coded Conversion 112
7.3.3 Temporal-Coded Conversion 113
7.4 Research Questions 113
7.5 Conclusion 115

8 Outlook 117
8.1 Rapid Prototyping of Rate-Coded Spiking Networks 117
8.2 Population-Coded Converted Networks 118
8.3 Temporal-Coded Converted Networks 118

Bibliography 121

List of Figures

1.1 Exponential growth of neural network parameters 2
1.2 Structure of the thesis 6

2.1 Chapter structure 11
2.2 Schematic diagram for the Hodgkin-Huxley model 15
2.3 Electrical equivalent circuit of the LIF neuron 17
2.4 Voltage courses of Hodgkin-Huxley, LIF and IF neurons 18
2.5 Membrane potential of leaky and non-leaky IF neurons 20
2.6 Conversion approach 32

3.1 Chapter structure 35
3.2 Exemplary coding schemes for a sequence of images over time 38
3.3 Visualization of rate coding techniques 39
3.4 Visualization of population coding techniques 40
3.5 Taxonomy of temporal coding techniques 41
3.6 Visualization of temporal coding techniques 42
3.7 Architecture of Resnet18 50
3.8 Architecture of the RetinaNet Feature Pyramid Network 51
3.9 Simple recurrent unit and LSTM block 52
3.10 Vision transformer architecture 53
3.11 Transformer architecture and attention mechanism 54
3.12 Relation between the conversion loss of test set and synthetic set 56

4.1 Chapter structure 59
4.2 Error-centering of spiking neurons 64
4.3 Biased and unbiased quantization 66
4.4 Inference of the different optimization methods for different-sized ResNets 69
4.5 Accuracy of the three optimization methods on the CIFAR-10 dataset . . 72
4.6 mAP of the three optimization methods 74
4.7 Architecture of the converted spiking NLP transformer network 79
4.8 Results of NLP transformer conversion 80
4.9 Architecture of the converted spiking vision transformer network 81
4.10 Results of vision transformer conversion 82

5.1 Chapter structure 85
5.2 Population-coded spiking LSTM 88
5.3 Example of an approximated sigmoid function with a population of neurons 89
5.4 Example of an approximated tanh function with two joint populations of neurons 90
5.5 Accuracy of the population-coded gates 93
5.6 Accuracy of the population-coded states 94

6.1 Chapter structure 97
6.2 Spike generation with TTFS spike encoding 99
6.3 Bias lane for temporal-coded conversion 102
6.4 Neural oscillation method with pulse duration adaption 107
6.5 Neural oscillation method with pulse duration adapation 108

List of Tables

2.1 Digital neuromorphic hardware solutions 23
2.2 Analog and mixed analog/digital neuromorphic hardware solutions 25

4.1 Accuracy for different optimization methods 70
4.2 Resulting accuracies for the original ANN 72
4.3 Overview of Conversion Losses 76

5.1 Results of the individual gate and state conversion 92
5.2 Results of the LSTM conversion 95

6.1 Comparison to previous temporal coding approaches 105
6.2 Comparison to rate coded approaches 106

List of Acronyms

AEIF Adaptive Exponential Integrate-and-Fire

AI Artificial Intelligence

ANN Artificial Neural Network

ASIC Application-Specific Integrated Circuit

BSA Ben's Spiker Algorithm

CNN Convolutional Neural Network

COCO Common Objects in Context

DAC Digital-to-Analog Converter

DBN Deep Belief Network

FF Feedforward

FPAA Field-Programmable Analog Array

FPGA Field-Programmable Gate Array

FPN Feature Pyramid Network

FPNA Field-Programmable Neural Array

GRU Gated Recurrent Unit

GUI Graphical User Interface

HBP Human Brain Project

HICANN High Input Count Analog Neural Network

HICANN-DLS High Input Count Analog Neural Network with Digital Learning System

HSA Hough Spiker Algorithm

IF Integrate-and-Fire

IMDb Internet Movie Database

ISI Interspike Interval

LIF Leaky Integrate-and-Fire

LSTM Long Short-Term Memory

mAP Mean Average Precision

MIM Metal-Insulator-Metal

ML Machine Learning

MNIST Modified National Institute of Standards and Technology Database

MW Moving-Window

NAS Neural Architecture Search

NEF Neural Engineering Framework

NLP Natural Language Processing

NN Neural Network

PSTH Peri-Stimulus-Time Histogram

QIF Quadratic Integrate-and-Fire

ReLU Rectified Linear Unit

ResNet Residual Neural Network

RetinaNet Retina Neural Network

RF Resonate-and-Fire

RNN Recurrent Neural Network

ROC Rank-Order Coding

RPN	Region Proposal Networks
RQ	Research Question
SDR	Sparse Distributed Representation
SF	Step-Forward
SGD	Stochastic Gradient Descent
SNN	Spiking Neural Network
STDP	Spike-Timing-Dependent Plasticity
TBR	Threshold-Based Representation
TC	Temporal Contrast
TTFS	Time-to-First-Spike
VLSI	Very-Large-Scale Integration

List of Symbols

Notation

a	A scalar (real or integer)
$\underline{a}$	A scalar (complex)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$a(t)$	A temporally changing variable (continuous time)
$a[t]$	A temporally changing variable (discrete time)
a_i	The i-th element of vector $\boldsymbol{a}$, with the index starting at 1
$a_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$a^{(i)}$	Element within the i-th layer of a network
$a^{(i,j)}$	Element with contributions from layer i to layer j
a_{descr}	A variable with distinct description
$f(a)$	A function of a
$\text{Re}(\underline{a})$	Real part of complex variable $\underline{a}$
$\text{Im}(\underline{a})$	Imaginary part of complex variable $\underline{a}$

Symbols

t	Time
Δt	Time interval
t^f	Firing times of a neuron
$v(t)$, $\underline{v}(t)$, $v[t]$	Membrane voltage of a neuron
v_{th}	Threshold voltage of a neuron
v_{reset}	Reset voltage of a neuron
C	Membrane capacitance of a neuron
R	Resistance of a neuron
$\delta(t)$, $z[t]$	Spike event
$i(t)$, $i[t]$	Input current of a neuron
τ	Time constant of a neuron
λ	Damping constant of a neuron
$\boldsymbol{W}$	Weight matrix containing the synaptic connection weights
f	Frequency
ω	Angular frequency
η	Learning rate
$\mathcal{L}$	Loss function
ψ_{lin}, ψ_{sig}	Pseudo gradient
s_{i}, s_{f}	Sparsity level
$\boldsymbol{x}$	Input vector
$\boldsymbol{y}$	Output vector
$\hat{\boldsymbol{y}}$	Label vector
$c_{\omega=\omega_n}(t)$	Envelope function of a neuron's response with $\omega = \omega_n$
$\sigma(x)$	Softmax function
$S(x)$	Logistic sigmoid function
N_{xxx}	Number of xxx
E_{xxx}	Energy consumption of xxx

1 Introduction

Artificial Intelligence (AI) has undergone remarkable progress in recent years, with a particular focus on Neural Networks (NNs). Despite the inception of computational approaches for NNs as far back as the 1950s [1], they only began to receive widespread recognition in the last decade. DanNet [2] was acknowledged in 2011 as the first NN to achieve superhuman performance on a pattern recognition dataset. This breakthrough marked the beginning of the disruption that AI-based approaches would bring to numerous industries.

Recent advancements in AI have led to a significant surge in transformative technologies across multiple disciplines. The medical field has made significant progress in the automated detection of diseases in imaging methods-based diagnosis [3], [4]. Similarly, the justice sector has used AI to identify issues in legal contracts [5]. Not least because NNs can optimize the training of NNs themselves better than humans can [6].

AI has also enabled solutions that traditional algorithms are unable to address. In 2016, an NN defeated the human champion in the game of Go [7]. Furthermore, the prediction of protein folding, a complex process that had eluded researchers for over 60 years, has now become feasible through the use of NNs [8]. This breakthrough has opened up new possibilities in the field of drug discovery, particularly for the treatments for diseases such as Alzheimer's and Parkinson's.

However, the high energy consumption of NNs is a significant issue that must be addressed. Despite this challenge, increasing the number of layers in a deep NN can improve performance by a fraction of a percent [10], leading to some models containing over 10,000 layers in depth [11]. The largest language model in 2020, GPT-3 [12], had more than 175 billion trainable parameters and required a staggering 190,000 kWh of energy for its training, resulting in the emissions of 85 tonnes of CO_2 equivalents [13].

This energy consumption assumption only accounts for the final training phase with optimal hyperparameter settings. Algorithms such as Neural Architecture Search (NAS) [14] can help determine these hyperparameters, resulting in multiple iterations and significantly increasing emissions, potentially by a factor of over 3,000 [15]. As Machine Learning (ML) capabilities continue to advance, so too does the need for increasingly powerful computational resources to support the training of sophisticated Artificial Neural Networks (ANNs). Recent studies suggest that this computational power doubles approximately every 3.4 months [16] (see Figure 1.1), compared to the computation per

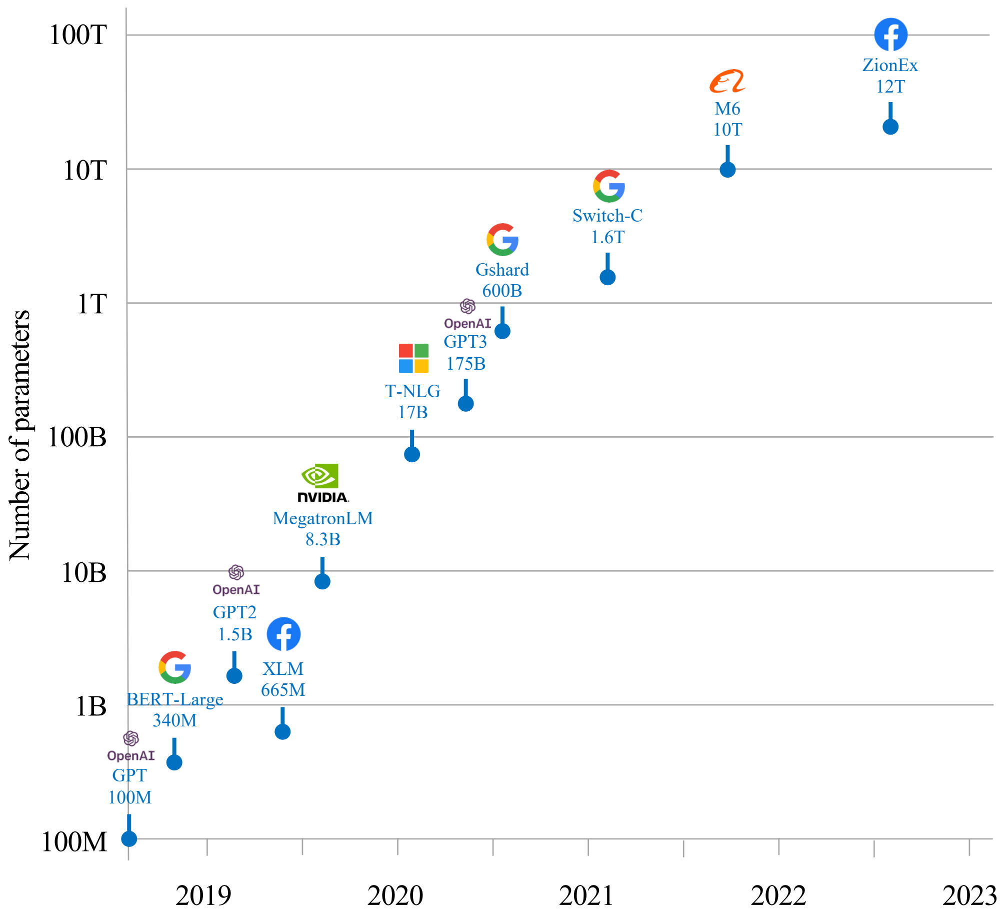


Figure 1.1: Exponential growth of neural network parameters. Parameter counts of several recently released pre-trained large language models (Figure updated from [9])

Joule doubling only every 2.6 years [17]. This exponential growth will be unsustainable in the coming years.

Researchers are exploring strategies to reduce the energy consumption of NNs, including (1) optimizing the information flow within the networks through techniques such as sparsity [18]–[20], (2) developing alternative network architectures such as MobileNet [21] and ShuffleNet [22], (3) improving training algorithms [23], and (4) utilizing hardware accelerators that are optimized for NN computations [24].

A further approach is neuromorphic computing, representing a promising paradigm based on the idea of mimicking the behavior of cortical neurons in biology. The core idea is to replicate the response of biological neurons to incoming stimuli in silicon hardware [25]. This approach has shown significant potential for energy consumption reductions compared to current hardware [26]. As a result, neuromorphic computing has garnered considerable attention from various research organizations and companies, including Intel [27], IBM [28], and Infineon [29].

Neuromorphic computing represents a continuation of the evolutionary trajectory of NNs, starting with the McCulloch-Pitts neurons [30] that were based on Boolean activation functions and followed by the development of continuously activated ANNs, sometimes also referred to as analog NNs. Recent advances in neural network design have led to the development of Spiking Neural Networks (SNNs), which leverage the temporal dynamics of spike events to encode and process information [31], making them computationally more powerful compared to preceding generations [32].

Despite the benefits of SNNs, the non-differentiability of pulses, typically represented as Dirac delta functions, presents a challenge for applying traditional training algorithms based on stochastic gradient descent and error backpropagation. Thus, new training approaches are required to overcome this challenge.

Training of SNNs falls into three categories: (1) biologically plausible learning rules at the synapses, for example, Spike-Timing-Dependent Plasticity (STDP) [33], [34], (2) supervised learning with spikes as a spiking variation of gradient-descent-based error backpropagation [35]–[37], and (3) the conversion of ANNs trained with backpropagation to SNNs, where the architecture and parameters of pre-trained ANNs are directly transferred to SNNs.

An ANN is trained with backpropagation to adjust the network's weights in the training approach. Following this training, the ANN is converted into a SNN through the replacement of the real-valued activation functions with spiking neurons. The activity level in the original ANN corresponds to the frequency of spikes generated by the spiking neuron and therefore encodes the information in the firing rate, accordingly referred to as rate-coded conversion. While rate-coded conversion is the most commonly used approach, researchers have proposed other methods that employ different coding techniques [38]–[40]. These conversion techniques have yielded the highest-performing SNNs to date, as reported in various literature sources. A comprehensive overview of the available performances can be found in [41].

An additional advantage is that an external party can handle the conversion process without requiring the creator of the original ANN to publish or share their dataset.

In this thesis, we evaluate encoding schemes and determine which are best suited for conversion approaches. We explore the available options, and when not available, we develop our approaches to fill the missing gaps. The rate-coded approach is the most common and most researched one. We scale it to very deep NNs with over 100 layers and present optimization algorithms for increasing inference speed and approximating hyperparameters.

For the other information encodings suited for conversion, we present a population-coded approach that shows high potential on subthreshold analog neuromorphic hardware and a sparsely activated temporal-coded approach, which can reduce the energy consumption even further than the low-powered rate-coded approach many times over. Our study provides a comprehensive analysis of the encoding schemes for SNNs and provides insights into the advantages and limitations of each approach. Our findings can guide researchers and hardware manufacturers in selecting the most suitable approach for their specific needs.

1.1 Research Questions and Scope of the Thesis

The overall question which guides this research work is as follows:

> *How can ANN-to-SNN conversion approaches be used for faster development of energy-efficient, state-of-the-art spiking networks?*

Answering this overarching question raises several issues that require careful consideration. One of the most significant benefits of spiking neurons is their ability to consume energy only when transmitting information, thus necessitating neurons that are sparsely activated for highly energy-efficient computation. The Rectified Linear Unit (ReLU) activation function is commonly used in ANNs. This activation function can be seen as the firing rate of a neuron when averaging the number of output spikes during a given interval. Because of that most conversion approaches adopt rate-coded spiking neurons to convert the ReLU activation function. Researchers have proposed several alternative coding schemes, but there is a lack of a comprehensive overview of those that could be utilized in the conversion process, which motivates the first Research Question (RQ):

Research Question 1 (Spike Encodings)
Which spike encodings can be used for mapping the activation function of ANNs to spiking neurons?

After analyzing the different viable encoding schemes, we investigate the state-of-the-art network architectures that can benefit from the ANN-to-SNN conversion approaches. While many conversion approaches focus on simple network architectures such as feedforward networks or basic Convolutional Neural Networks (CNNs), real-world applications typically require more optimized architectures for specialized tasks with higher performance, such as RetinaNet or transformer networks. Consequently, the second RQ arises:

Research Question 2 (State-of-the-Art Architectures)
Which state-of-the-art neural network architectures and operators are missing a conversion approach?

Prior to the deployment on neuromorphic hardware, it is necessary to convert the networks and run simulations to evaluate and optimize the conversion approaches. However, running SNN simulations is computationally expensive and time-consuming compared to the matrix multiplications of ANNs on optimized hardware. The converted spiking networks usually take a long time to converge to their highest accuracy; therefore various optimization approaches have been presented over time. These approaches introduce a speed-accuracy-tradeoff, with the optimal choice for the newly introduced hyperparameters usually involving trial and error to obtain the best results. Determining these hyperparameters before running a simulation of an SNN could significantly improve the speed of development, which motivates the third RQ:

Research Question 3 (Rapid Prototyping)
Can the accuracy of the converted networks already be estimated before the conversion process?

Many networks that solve real-world applications comprise deep structures with a large number of layers. Accordingly, for converted SNNs to be viable, they must be able to handle very deep networks on large datasets with high accuracy. Small errors can accumulate per layer, resulting in unusable performance in very deep networks. Additionally, many of the current approaches used small datasets such as MNIST or CIFAR-10, which do not represent real-world datasets. Therefore, we raise the following RQ:

Research Question 4 (Scalability and Optimization)
Do the existing approaches scale to very deep NNs with large datasets and can they be further optimized?

1.2 Structure

The present work is organized into eight chapters, aimed at addressing the conversion of ANNs to SNNs. A schematic representation of the structure of this thesis is illustrated in Figure 1.2.

Chapter 1 serves as an introduction, in which the motivation for the research is presented, and the problem statement is introduced. Furthermore, the chapter formulates the research questions that focus on bridging the knowledge gap. The chapter concludes with a list of publications that form the basis of the work presented in this thesis or are directly related to it.

Chapter 2 focuses on the background and related work concerning the neuron models used in this thesis, with Section 2.1 introducing the relevant models. Section 2.2 provides an overview of the research landscape of neuromorphic hardware accelerators, while

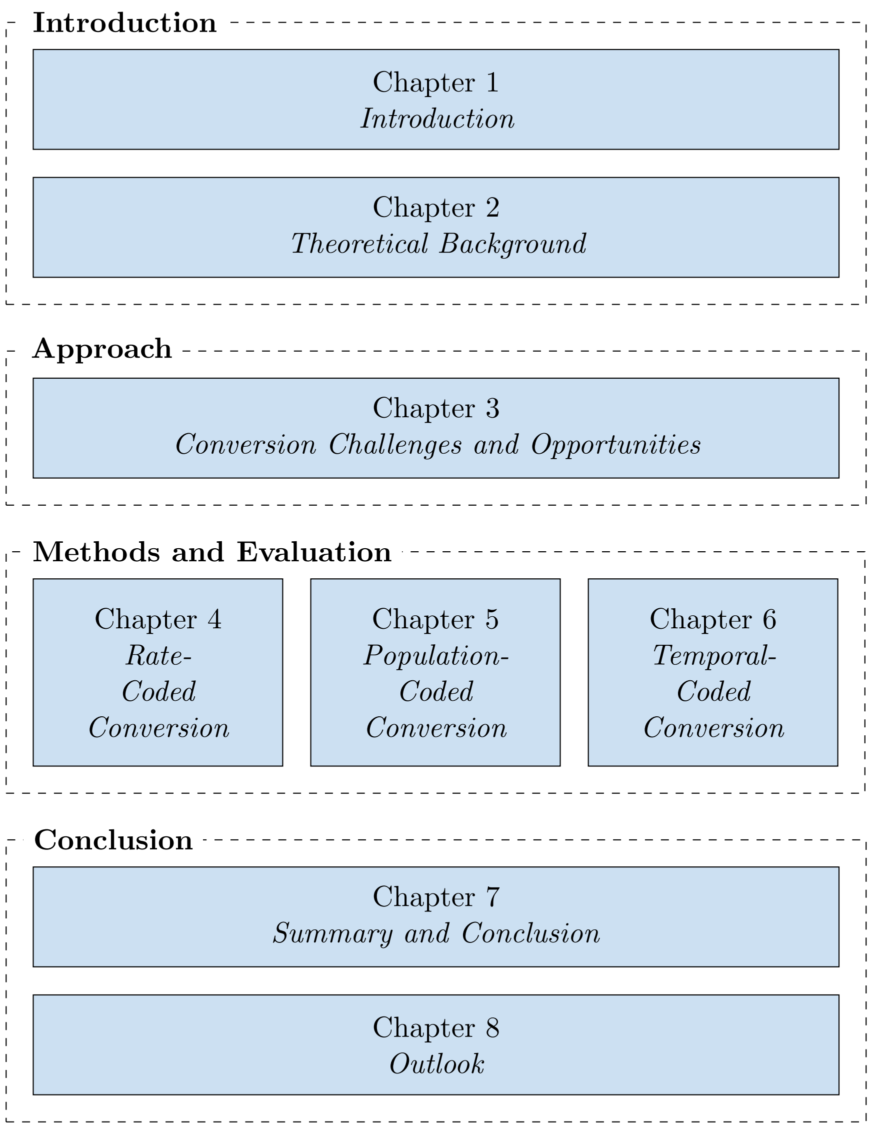


Figure 1.2: Structure of the thesis.

Section 2.3 highlights the different approaches for the training of SNNs, including bio-inspired, artificial, and conversion methods.

Chapter 3 describes the approach of the present work, beginning with an overview of the various encoding schemes that are compatible to encode information in a spike-based format. It evaluates which encoding schemes are best suited for conversions and which are not, aiming to answer the first research question. Next, it discusses the most important network architectures available, highlighting which ones have been, can, and need to be converted. Lastly, in Section 3.3, we discuss the weight normalization for the ANN-to-SNN conversion and how it can be approximated for faster development of converted SNN, which aims to answer the third research question.

The following three chapters (Chapters 4 to 6) are grouped by the coding mechanisms that are best suited for the conversion, as discussed in the previous chapter. Chapter 4 focuses on rate-coded conversion, which is currently the state-of-the-art for conversion. It presents an optimization method in Section 4.1, which minimizes the inference time of converted networks and compares it with other recently published optimization methods. In Section 4.3, the chapter demonstrates the usage of the ReLU1 activation function for rapid prototyping of converted SNNs, presenting a very deep CNN with residual blocks, namely Residual Neural Network (ResNet), for image classification. It then uses the combination of optimization methods with a Feature Pyramid Network (FPN) for object detection, namely Retina Neural Network (RetinaNet), to achieve a more than $10\times$ speed increase compared to the basic conversion approach, while simultaneously achieving the best-in-class accuracy on a very large dataset. Lastly, in Section 4.4, the chapter shows a novel method to convert transformer networks to spiking networks and its use for Natural Language Processing (NLP) and classification tasks.

Chapter 5 presents the conversion of Recurrent Neural Networks (RNNs), utilizing Long Short-Term Memory (LSTM) as memory cells to solve an NLP task, achieving better performance than previous conversion approaches using vanilla RNNs. This is possible due to one of the drawbacks of subthreshold analog neuromorphic hardware, which can represent S-shaped activation functions, such as sigmoid and tanh, with populations of spiking neurons.

Chapter 6 demonstrates the usage of sparse temporal codes instead of encoding information into the firing rates of neurons. Here, information is encoded into the time before a spike arises, resulting in only positive activations in the original ANN that produce a single spike in the converted SNN. The chapter showcases this approach for a classification task with a converted CNN.

Chapter 7 summarizes the outcomes and limitations of the preceding methodological and application-focused chapters. Lastly, Chapter 8 gives an outlook on emerging topics and potential directions of the ANN-to-SNN conversion and its application on neuromorphic hardware.

1.3 Contributions

Several parts of this dissertation have been previously published and presented in peer-reviewed journals, international peer-reviewed conferences or international peer-reviewed workshops.

In order to answer the first RQ, a comprehensive review article was composed and published in a journal, which provides an in-depth analysis of encoding techniques in biological and artificial SNNs. This article's content is reflected in Section 3.1, forming the basis for the classification of ANN-to-SNN conversion.

1. Daniel Auge, Julian Hille, Etienne Mueller, and Alois Knoll. **A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks.** *Neural Processing Letters*, 2021. [41]

Based on the findings of the review article, the conversion approaches have been classified by their spike coding into three groups: rate, population, and temporal coding. The group of rate-coded networks marks the state-of-the-art for conversion and is reflected in Chapter 4. An approach for the improvement of the inference time of rate-coded networks was presented as a poster:

2. Etienne Mueller, Julius Hansjakob, Daniel Auge, **Faster Conversion of Analog to Spiking Neural Networks by Error Centering.** *Bernstein Conference*, 2020. [42]

This approach was used to extend the existing rate-coded approach and to scale CNNs to deep networks with large datasets while speeding up the development and further optimizing the inference time. The details of these evaluations were published as a conference paper:

3. Etienne Mueller, Julius Hansjakob, Daniel Auge, Alois Knoll, **Minimizing Inference Time: Optimization Methods for Converted Deep Spiking Neural Networks.** *International Joint Conference on Neural Networks (IJCNN)*, 2021. [43]

Moreover, a novel approach for the conversion of transformer networks and their attention mechanism was presented based on the above findings. The details are included in Section 4.4, which was previously published as a conference paper:

4. Etienne Mueller, Viktor Studenyak, Daniel Auge, Alois Knoll. **Spiking Transformer Networks: A Rate Coded Approach for Processing Sequential Data.** *7th International Conference on Systems and Informatics (ICSAI)*, 2021. [44]

Additionally, a search algorithm for the approximation of the normalization hyperparameter was proposed. Its finding is included in Section 3.3 and was previously presented as a poster:

5. Etienne Mueller, Daniel Auge, Alois Knoll, **Normalization Hyperparameter Search for Converted Spiking Neural Networks.** *Bernstein Conference*, 2021. [39]

Our experiments showed that a rate-coded approach is not suitable for the conversion of RNNs based on memory cells. We proposed a novel approach that exploits the properties of analog neuromorphic hardware that uses populations of neurons instead of single, rate-coded neurons and builds the base of Chapter 5. The approach was previously published as a conference paper:

6. Etienne Mueller, Daniel Auge, Alois Knoll. **Exploiting Inhomogeneities of Subthreshold Transistors as Populations of Spiking Neurons.** *International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2022. [40]

Moreover, the temporal coding approach has been identified as the most energy-efficient conversion approach. However, previous work did not scale to deep networks, therefore we introduced an approach for solving the problem resulting in a conversion with less than a spike per neuron. It is covered in Chapter 6 and was published at a workshop:

7. Etienne Mueller, Simon Klimaschka, Daniel Auge, Alois Knoll. **Neural Oscillations for Energy-Efficient Hardware Implementation of Sparsely Activated Deep Spiking Neural Networks.** *Association for the Advancement of Artificial Intelligence (AAAI), Practical DL*, 2022. [45]

The findings from the previously mentioned publications have led to the creation of the publicly available toolbox:

8. Etienne Mueller. **Convert2SNN: Toolbox for the ANN-to-SNN Conversion**. Available: `https://github.com/EtienneMueller/Convert2SNN`, *MIT License*, 2021 [Online]. [46]

Additionally, several works contribute to the field of spiking neuron models but are not directly part of this thesis:

9. Daniel Auge, Etienne Mueller. **Resonate-and-Fire Neurons as Frequency Selective Input Encoders for Spiking Neural Networks.** *Technical Report TUM*, 2020. [47]

10. Daniel Auge, Julian Hille, Felix Kreutz, Etienne Mueller, Alois Knoll. **End-to-end Spiking Neural Network for Speech Recognition Using Resonating Input Neurons.** *30th International Conference on Artificial Neural Networks (ICANN)*, 2021. [48]

Furthermore, there are works related to gesture recognition with SNNs but use different training algorithms than those presented in this thesis:

11. Daniel Auge, Julian Hille, Etienne Mueller, Alois Knoll. **Hand Gesture Recognition in Range-Doppler Images Using Binary Activated Spiking Neural Networks.** *IEEE International Conference on Automatic Face and Gesture Recognition*, 2021. [49]

12. Daniel Auge, Philipp Wenner, Etienne Mueller. **Hand Gesture Recognition using Hierarchical Temporal Memory on Radar Sequence Data.** *Bernstein Conference*, 2020. [50]

2 Theoretical Background

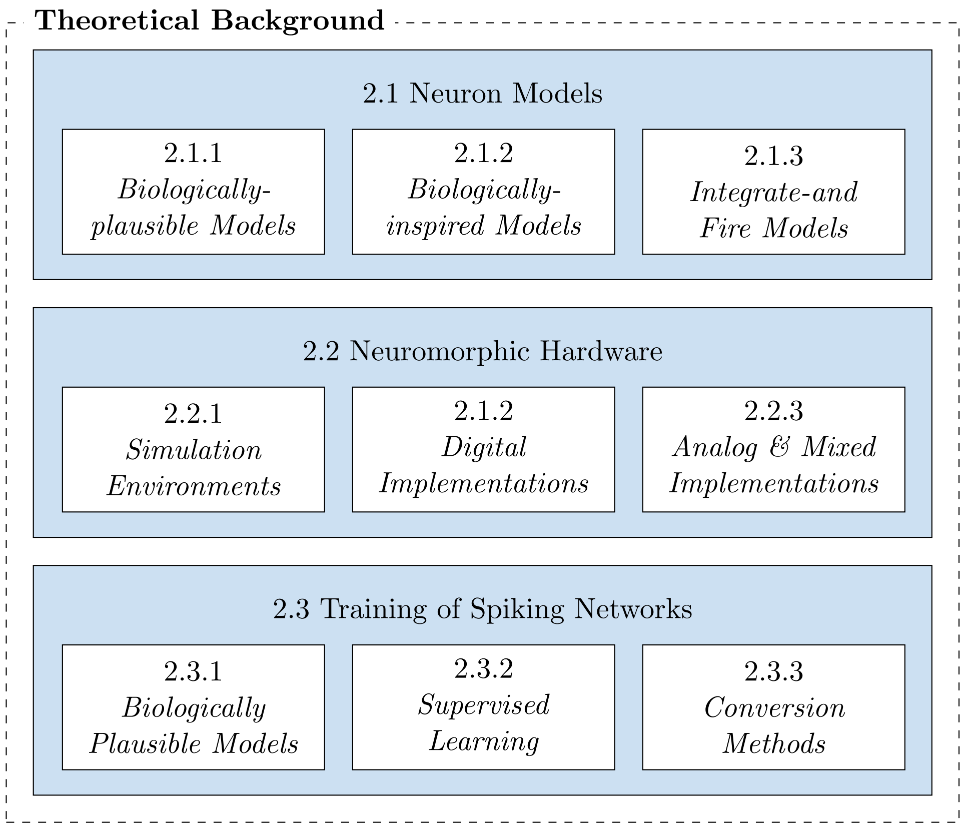

Theoretical Background
2.1 Neuron Models
2.1.1 Biologically-plausible Models
2.1.2 Biologically-inspired Models
2.1.3 Integrate-and Fire Models
2.2 Neuromorphic Hardware
2.2.1 Simulation Environments
2.1.2 Digital Implementations
2.2.3 Analog & Mixed Implementations
2.3 Training of Spiking Networks
2.3.1 Biologically Plausible Models
2.3.2 Supervised Learning
2.3.3 Conversion Methods

Figure 2.1: Chapter structure.

The present chapter provides an overview of the background and related research that underpins our proposed approach. The chapter is structured as follows (see Figure 2.1): In the first section, we provide a succinct introduction to the fundamental neuron models that serve as the basis for this work, including biologically-plausible models like Hodgkin-Huxley models and computationally efficient integrate-and-fire models. Next, we discuss the current state-of-the-art hardware solutions for the energy-efficient execution of SNNs. We review software simulation environments, digital hardware platforms such as TrueNorth, SpiNNaker, and Loihi, as well as analog and mixed analog/digital implementations like BrainScaleS and Neurogrid. The third section is dedicated to various learning algorithms designed specifically for SNNs, consisting of three subsections that examine biologically plausible algorithms, supervised learning algorithms, and conversion methods, which are the primary topic of this thesis.

In subsequent chapters (Chapters 4 to 6), we present relevant prior works that are specific to the application domains explored in later chapters.

2.1 Neuron Models

Biological neurons are typically composed of three main components: a cell body, an axon, and dendrites. Dendrites typically receive information and transmit it to the cell body, axons transmit the neuron's output. Neurons are receiving inputs via electrical or chemical transmissions from different neurons. The point of contact amidst the end of one neuron's axon and the dendrite of a postsynaptic neuron, where information or signals are being transmit, is referred to as a synapse.

Generally, neurons accumulate charge by changes in the voltage potential across their cell membrane of the neuron, as a result of received inputs from previous neurons over synaptic connections. When the cumulative charge within a neuron reaches a specific threshold, it triggers the neuron to set off an action potential, which travels down the neuron's axon, affecting the charge on subsequent neurons via synaptic transmission.

Neuron models are the mathematical representations of biological neurons and are essential in the simulation of neural systems. The first attempt to model biological neurons was made by McCulloch and Pitts in 1943 [30]. In general, in NNs the output value a_j of a neuron j can be calculated with

$$a_j = \sigma\left(\sum_{i=0}^{N} w_{i,j} x_i\right) \tag{2.1}$$

which is the sum of the output value x_i of neuron i weighted by the weight $w_{i,j}$ of the synaptic connection between neuron i and neuron j over the number of inputs N into the neuron j, with a non-linear activation function σ. In the approach of McCulloch and Pitts, the activation function $\sigma_{\text{MP}}(x)$ could only take binary values:

$$\sigma_{\text{MP}}(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

with θ being a threshold, that if exceeded by the accumulated weighted inputs x, the activation function would return 1, otherwise, it returns 0. These NNs are also known as the *first generation* of neural networks.

Today's common ANNs represents the *second generation* of NNs. ANNs use continuous-valued activation functions, for example ReLU [51]:

$$\sigma_{\mathrm{ReLU}}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Their activation can be seen as the firing rate of biological neurons. As a result, ANNs can transmit more information and show far better performance than the previous generation. Due to their continuous-valued activation function, ANNs are sometimes also referred to as *analog neural networks*. This improvement can represent more information and has thus led to very deep networks and many breakthroughs in the last few years.

In the *third generation* of NNs, the SNNs, the activation function is exchanged with biologically-inspired neuron models [32]. These neuron models typically consist of coupled Ordinary Differential Equations (ODEs) to describe the dynamics of the membrane potential, which is mainly influenced by various factors such as ionic currents, synaptic inputs, and intrinsic currents. The goal of the neuron models is to capture the essential features and behaviors of real neurons to make predictions about their response to input stimuli. The choice of neuron model depends on the goal of the research, with some models being more complex and biologically realistic, while others are simplified, computationally light representations [31].

In this section, we introduce the most common neuron models. They can be broadly grouped into three main categories:

- *Biologically-plausible models* which explicitly models the behavior seen in biological neural systems. The most common examples are the Hodgkin-Huxley [52] and the Morris-Lecar model [53].
- *Biologically-inspired models* are attempting to emulate the behavior of biological systems, albeit often without strict adherence to biological plausibility. Notable models are Fitzhugh-Nagumo [54], [55], Hindmarsh-Rose [56] and Izhikevich [57].
- *IF models* are a relatively simple class of biologically-inspired spiking neuron models. There are many models in the family of Integrate-and-Fire (IF) models, besides simple IF neurons, for example, the Leaky Integrate-and-Fire (LIF) [58], Quadratic Integrate-and-Fire (QIF) [59], and Adaptive Exponential Integrate-and-Fire (AEIF) [60] models.

2.1.1 Biologically-plausible Models

The Hodgkin-Huxley model, first proposed in 1952, is considered the most widely used biologically-plausible neuron model [52]. By conducting experiments on the giant axons of squid they developed the first biologically feasible neuron model. They identified that the transport of ions through channels between the exterior and interior of the neuron adapts the cell's membrane potential (see Figure 2.2 left). As main carriers, they identified the presence of three distinct ionic currents: sodium, potassium, and a leakage current that primarily consists of chloride ions. Ion flux through the cell membrane is regulated by voltage-gated ion channels, with one channel specifically for potassium and another for sodium. The leakage current consists of additional ion channels that are not explicitly described.

The Hodgkin-Huxley model is relatively complex, incorporating four-dimensional nonlinear ODEs that capture the flows and concentrations of ions in and out of the neuron. It can be approximated as a capacitance in parallel with multiple resistors (see Figure 2.2 right). The membrane potential $u(t)$ is given by the equation for the membrane capacitance C [58]

$$C\frac{\mathrm{d}u}{\mathrm{d}t} = -\sum_k I_k(t) + I(t) \tag{2.4}$$

where u represents the voltage across the membrane and $\sum_k I_k$ the sum of the ionic currents which pass through the cell membrane, $i(t)$ the input current. The sum of internal ion currents $I_k(t)$ is represented as

$$\sum_k I_k(t) = g_{\mathrm{Na}} m^3 h(u - E_{\mathrm{Na}}) + g_{\mathrm{K}} n^4 (u - E_{\mathrm{K}}) + g_{\mathrm{L}}(u - E_{\mathrm{L}}) \tag{2.5}$$

with E_{Na}, E_{K}, and E_{L} being the reversal potentials for sodium (Na), potassium (K) and the leakage. Hodgkin and Huxley empirically deducted the parameters for them have [52]. The different variables for n, m, and h are described by ODEs in the form

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -\frac{1}{\tau_x(v)}\left[x - x_0(v)\right]. \tag{2.6}$$

The necessary time constants and reverse potentials were empirically determined on the giant axon of the squid. After crossing a certain threshold, the neuron's membrane potential explosively rises and falls back to a resting state. This is commonly referred to as action potential, spike or pulse.

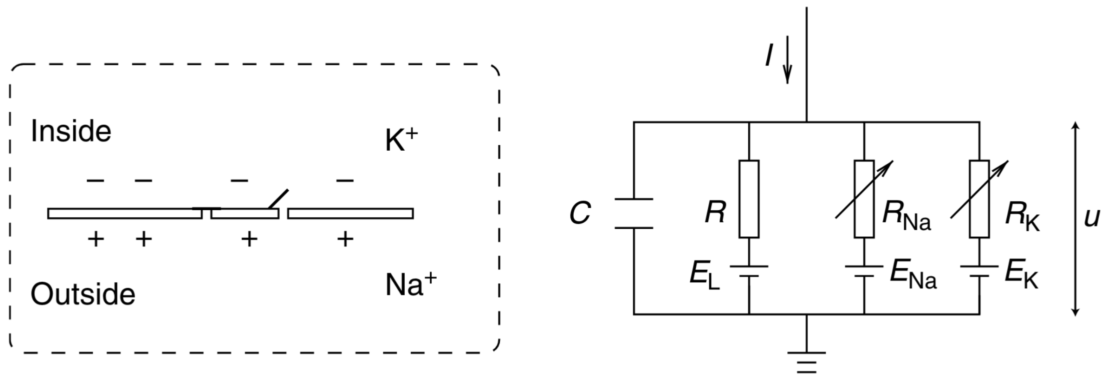


Figure 2.2: Schematic diagram for the Hodgkin-Huxley model. Left: ion channels in the membrane of a neuron. Right: approximation of the Hodgkin-Huxley model as a capacitance with multiple resistors in parallel (Figure from [58])

Approximating the temporal characteristics reduces the four-dimensional Hodgkin-Huxley model to a simpler nonlinear two-dimensional model. The reduction process involves two simplifying approximations [61]. The first approximation involves the recognition that the evolution over time of the gating variable m is much more rapid than that of the other two gating variables n and h. This leads to the treatment of m as an instantaneous variable, represented as $m(t) \mapsto m_0\,[v(t)]$. The second approximation is based on the observation that the time constants τ_n and τ_m exhibit similar temporal dynamics, which suggests the approximation of the two variables n and h by a single variable v_2. This simplification process results in the simplification of the original equations Equations (2.4) to (2.6) to the simplified equation

$$\begin{aligned} \frac{\mathrm{d}v}{\mathrm{d}t} &= \frac{1}{\tau}\,[f(v, v_2) + R\,i(t)] \quad \text{and} \\ \frac{\mathrm{d}v_2}{\mathrm{d}t} &= \frac{1}{\tau_2} g(v, v_2). \end{aligned} \tag{2.7}$$

By reducing the dimensions, this approach enables for a computationally more efficient description of the behavior of neurons while retaining the key features and dynamics of the Hodgkin-Huxley model.

The Morris-Lecar [53] model is another widely used biologically-plausible model that simplifies the Hodgkin-Huxley model into a two-dimensional nonlinear equation. It introduces a recovery variable W that quantifies the likelihood of the K^+-ion channel being in the open (conducting) state. The derivative W' describes the relaxation dynamics wherein proteins channels undergo structural changes between ion-conducting and non-conducting states.

2.1.2 Biologically-inspired Models

There exist several simplified models of the Hodgkin-Huxley neuron for practical implementation in hardware, such as the Fitzhugh-Nagumo model [54], [55] and the Hindmarsh-Rose model [56]. These models are computationally simpler and require fewer parameters than the Hodgkin-Huxley model. Although they are not as biologically plausible, they still aim to model the behavior of neurons.

The Izhikevich spiking neuron model [57] is a popular simplified model that can elicit bursting and spiking behaviors similar to the Hodgkin-Huxley model, albeit with strongly reduced computational needs. It is well-known due to its simplicity and its ability to accurately model biological dynamics with just the 2D system of ODEs

$$\begin{aligned} v' &= 0.04v^2 + 5v + 140 - u + I \\ u' &= a(bv - u) \end{aligned} \tag{2.8}$$

with v and u being dimensionless variables, a, b, c, and d dimensionless parameter, and t the time.

Other popular neuron models are the Mihalaş-Niebur model [62], which replicates bursting and spiking behaviors with a set of linear ODEs, and the quartic model [63], which has two non-linear ODEs that describe its behavior.

2.1.3 Integrate-and-Fire Models

The IF family of spiking neuron models comprises a simpler model group that ranges from relatively straightforward (e.g. the basic IF) to variants that approach the complexity of the previously mentioned Izhikevich model [58]. While these models are less biologically plausible than others, they are useful in spiking neural systems as they produce sufficiently complex behaviors.

The popular Leaky Integrate-and-Fire (LIF) model simplifies the Hodgkin-Huxley model to a basic resistor-capacitor circuit (see the schematic representation depicted in Figure 2.3). It accumulates the incoming charge and includes a decay term to account for the potential decay over time. The LIF model represents the behavior of a single

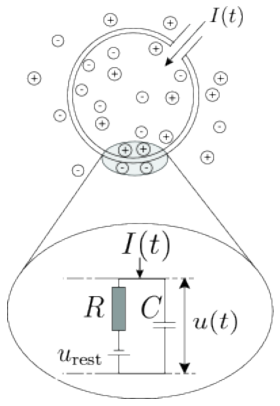


Figure 2.3: Electrical equivalent circuit of the LIF neuron. The membrane potential is applied to the capacitance, the leakage current is simulated by a parallel resistor, causing the voltage to drop to the resting potential. (Figure from [61])

neuron by describing the dynamics of its membrane potential $u(t)$ with

$$C\frac{\mathrm{d}v}{\mathrm{d}t} = -\frac{1}{R}v(t) + i(t) \tag{2.9}$$

with the membrane capacitance C, the input resistance R, and the input current $I(t)$.

Typically, the input resistance and membrane capacitance are merged into a single parameter.

The time constant τ governs the rate of voltage leakage, describing an exponential decay of the membrane potential. Upon reaching a threshold voltage v_{th}, the LIF neuron initiates an action potential, followed by a reset of the membrane potential to v_{reset}. A refractory period can be set for the neuron after resetting the membrane potential, during which it is unable to generate another spike.

The LIF neuron model's dynamics are driven by an input current comprising two components: the continuous stimulation i_{cont} and the contributions from other neurons connected via simple synapses, each with their own unique weighting factor $w_{i,j}$. The equation for this input current is given by

$$i_i(t) = i_{\text{cont}}(t) + \sum_{f,j} w_{i,j}\delta(t - t_j^f). \tag{2.10}$$

where the incoming spikes are represented as delta functions $\delta(t)$. These delta functions are zero at all time except at $t = 0$, and $\int_{-\infty}^{\infty} \delta(x)dx = 1$. Incoming spikes from each connected neuron are weighted and summed, leading to an instantaneous change in the membrane voltage whenever an incoming spike is received. The weights can be positive

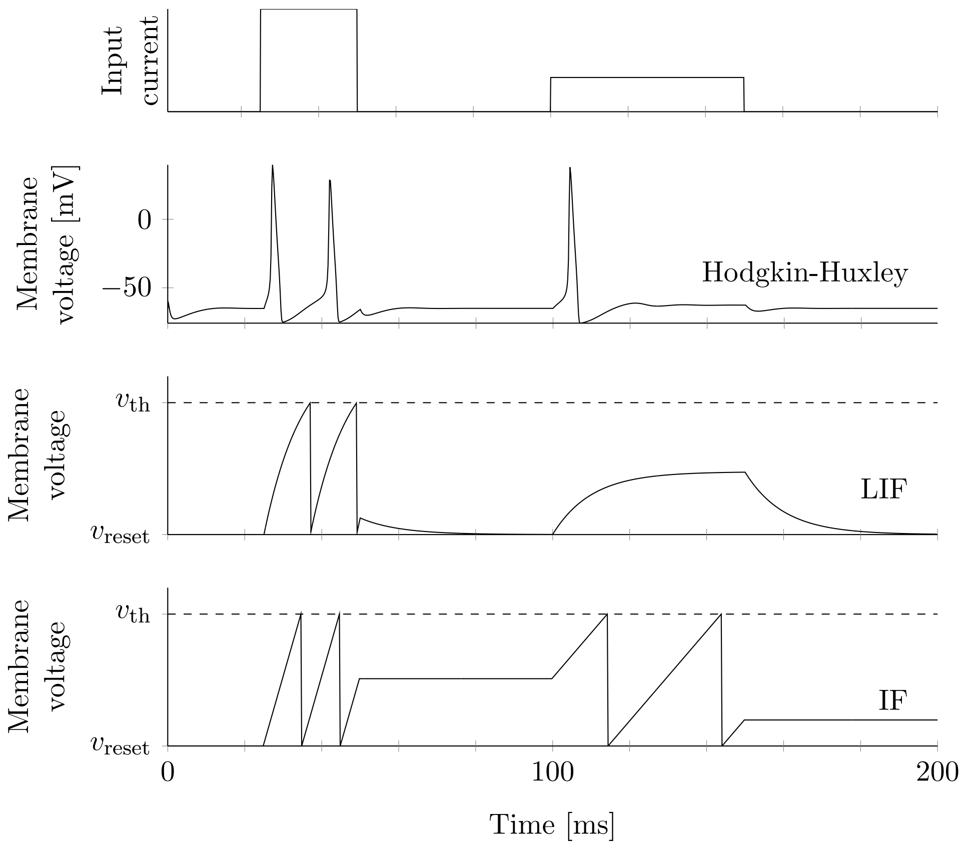


Figure 2.4: Voltage courses of Hodgkin-Huxley, LIF and IF neurons. The exemplary current in the top plot is applied to all three models. The initial current causes all neurons to generate two spikes. The dynamics of the Hodgkin-Huxley model are more complex due to the temporal evolution of the variables n, m, and h (not shown). The second current leads to an action potential of the Hodgkin-Huxley model. However, it is not able to charge the LIF neuron enough to reach the threshold voltage. Due to the missing leakage in the IF neuron, it is even able to generate two action potentials during the second current.

or negative, representing the excitatory or inhibitory nature of the synaptic connection, respectively.

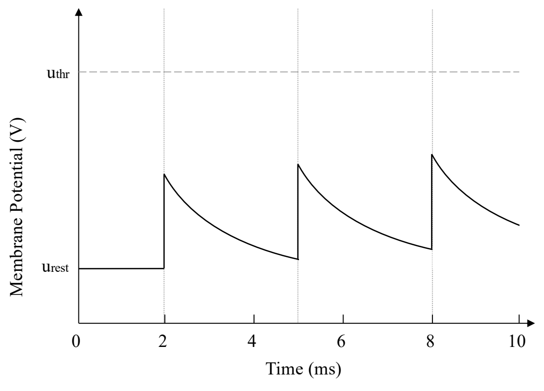
The basic non-leaking IF model simply integrates its input without any decay over time. Its equivalent electric circuit can be seen as just a perfect capacitance without the resistance in parallel. It was already formally described in 1907 by Louis Lapicque [64], even before the neuronal dynamics were discovered. With no leakage present the formula for the membrane potential further simplifies to the time derivative of the law of capacitance $Q = CV$:

$$C\frac{dV(t)}{dt} = I(t) \tag{2.11}$$

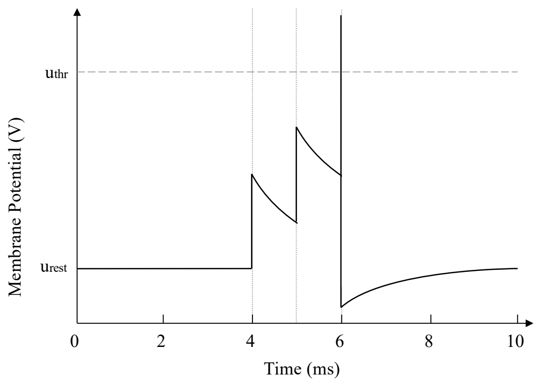
This allows for the modeling of the complex interactions between neurons in a network. An example of the membrane potential dynamics of a neuron modeled as Hodgkin-Huxley, LIF and IF neuron can be seen in Figure 2.4.

The leakage in spiking neuron models can lead to different responses to input patterns, which would result in the same outcome in a ANNs. For instance, the linear activation of an ANN, e.g. ReLU, would equal $a = 1.5$ for both input patterns $[0.8, 0.5, 0.2]$ and $[0.6, 0.5, 0.4]$, when all weights are set to 1. However, when the input is represented as spikes in a 10 ms temporal window, the behavior of LIF and IF neurons can differ. The LIF neuron will remain silent in response to the first input pattern (as depicted in Figure 2.5a), while an action potential is generated in response to the latter input pattern (as depicted in Figure 2.5b).

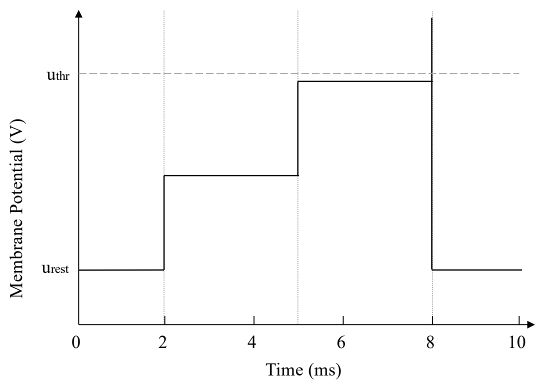
Further neuron models in this family are amongst others the general nonlinear IF method [58], such as the QIF model [59], which introduces further complexity with models, or the AEIF model [60], which is similar in complexity to the Izhikevich model.



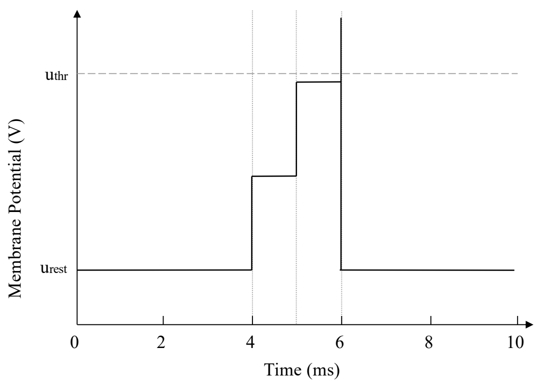
(a)



(b)



(c)



(d)

Figure 2.5: Membrane potential of leaky and non-leaky IF neurons. Change in membrane potential of LIF neurons (a and b) and non-leaky IF neurons (c and d) as a response to input spikes at [2, 5, 8] ms (a and c) and [4, 5, 6] ms (b and d). Whereas the IF neuron generates spikes in both cases, the LIF only creates an action potential in the latter case (Figure previously published in [45]).

2.2 Neuromorphic Hardware

Neuromorphic hardware has emerged as a promising architecture for addressing the limitations of von Neumann systems. Carver Mead first coined the term *neuromorphic computing* in 1990 to describe Very-Large-Scale Integration (VLSI) containing analog components inspired by biological neural systems [25]. However, the term has been expanded to include a wide range of implementations that are based on either biologically inspired or ANNs and employ non-von Neumann architectures.

Neuromorphic systems are characterized by their strongly interconnected and parallel nature, as well as their low-power requirements. These features are of great interest, as they circumvent the limitation of bandwidth constraints between the processing unit and memory imposed by the von Neumann bottleneck in classical digital network designs [65]. Consequently, neuromorphic computers have been gathering increased attention as a promising alternative to traditional von Neumann architectures, offering advantages in terms of speed, energy efficiency, and compactness. As such, there is significant motivation to develop hardware that leverages the advantages of neuromorphic architectures.

Numerous taxonomies have been proposed for neuromorphic hardware systems [66], but they are commonly divided at a high level into three major categories [67]:

- *Digital systems* rely on Boolean logic-based gates for computation and are typically synchronous or clock-based. They can be further grouped into two subcategories: programmable Field-Programmable Gate Arrays (FPGAs) and fully custom Application-Specific Integrated Circuits (ASICs).
- *Analog systems* work with continuous values, exhibiting asynchrony by nature. The computational processes in neuromorphic hardware often align with the kinds of operations that analog systems perform naturally. While analog systems can be more prone to noise than their digital counterparts, the robustness of neural networks to moise and faults makes them an attractive candidate for neuromorphic hardware. [68]. Analog Systems either use circuitry that is operating in superthreshold or subthreshold mode.
- *Mixed analog/digital systems* commonly use analog circuitry as the core processing elements of neurons and synapses. To overcome multiple issues of entirely analog systems, such as unreliability, digital components have been used in some neuromorphic systems to store synaptic weight values or other critical memory component of the system. Digital memory components have been found to show less noise and higher reliability than analog-based solutions.

There are currently a variety of neuromorphic processing chips available, many of them providing scalable multipurpose systems primarily for research purposes [27], [28],

[69]. These hardware accelerators allow for the realization of arbitrarily large networks without being constrained by hardware limitations. The Human Brain Project (HBP) [70] and the Brain Initiative [71] are two prominent projects aiming to scale up these accelerators to architectures capable of emulating the activity of a human brain which comprises approximately 86 billion neurons.

2.2.1 Software Simulation Environments

The modeling of the time-dependent behavior of SNNsis a crucial step in their implementation of hardware accelerators. Several tools have been developed for this purpose, including NEURON [72], GENESIS [73], Brian 2 [74], and NEST [75]. In a comparative evaluation of these tools in [76], NEST has been found to perform well at large network simulations and scaling to large compute systems, making it the preferred simulator in the Neurorobotics platform [77] of the HBP.

On the other hand, Brian 2 is recognized for its ease of use, compact code length, and extensive documentation. Further, NEURON and GENESIS are the most widely used simulators due to their precision in modeling biological behaviors. It is worth noting that many of these software frameworks are utilizing PyNN as a universal interface and offer built-in interfaces to seamlessly integrate with neuromorphic hardware.

Recently, the use of automatic differentiation frameworks such as TensorFlow [78] and PyTorch [79] has been gaining momentum within the SNN community. These frameworks were initially designed for training second-generation networks. Still, the development of pseudo-gradient-based learning algorithms in SNNs has made them available for simulating spiking networks as well.

2.2.2 Digital Implementations

Digital neuromorphic systems can be further grouped into two subcategories: flexible FPGAs and custom ASICs. Whereas the first one is often, but not exclusively, a temporary solution before a custom chip implementation, the latter one is a less flexible, but more energy-efficient solution.

FPGA-based Accelerators

FPGAs have been recognized as a promising solution for the acceleration of neuromorphic processes using commercially available hardware. In recent works, such as [80]–[84], various architectures have been proposed, which take advantage of the unique characteristics of FPGAs. One such architecture is DeepSouth [84], which is highly scalable and demonstrated the simulation of billions of LIF neurons on a chosen FPGAs board.

Table 2.1: Digital neuromorphic hardware solutions.

Name		**Year**	**Neurons**	**Syn.**	**Models**	**Process** [nm]	**Die Size** [mm^2]
SpiNNaker	[85]	2014	16k	16M	Arbitrary	130	100
SpiNNaker 2	[69]	2019	tba	tba	Arbitrary	22	tba
TrueNorth	[28]	2014	1M	256M	LIF	28	430
Loihi	[27]	2018	131k	130M	LIF	14	60
Darwin	[86]	2016	2048	4M	LIF	180	25
ODIN	[87]	2019	256	64k	LIF/Izh.	28	0.086*
Chen2018	[88]	2018	4096	1M	LIF	10	1.72
Cho2019	[89]	2019	2048	149k	LIF	40	2.56
Park2019	[90]	2019	410	200k	Sigmoid	65	10
Tianjic	[91]	2019	40k	10M	hybrid	28	14.5
μBrain	[92]	2021	336	20k	LIF	40	2.82
Kuang2021	[93]	2021	1024	1M	LIF	28	3.66
Kuang2021	[94]	2021	64k	64M	LIF	65	107
Zhong2021	[95]	2021	1024	256k	LIF	28	1.41

*Without peripherals and pads.

It is worth noting that the size of the networks that can be simulated depends on factors such as the temporal simulation speed and available memory. However, one significant disadvantage of FPGAs is their relatively poor energy efficiency when compared to ASICs. This is mainly due to the inherent flexibility introduced by the programmable hardware, which inevitably leads to higher energy consumption. Nevertheless, FPGAs are relatively ubiquitous and accessible to most researchers, who can implement circuits in FPGAs using hardware description languages such as VHDL or Verilog. These benefits make FPGAs an attractive option for many neuromorphic computing applications.

ASIC-based Accelerators

Fully custom or ASICs chips have also been widely used in neuromorphic implementations. The most widely used are SpiNNaker, IBM TrueNorth and Intel Loihi. An overview of some popular digital implementations can be found in Table 2.2.

SpiNNaker SpiNNaker is a powerful multicore platform designed to simulate large SNNs at sizes comparable to that of the human brain [85]. The architecture of SpiNNaker is composed of an arbitrary number of processing nodes, enabling scalability to any desired size. Each processing node is equipped with 18 cores per package that includes local and shared memory, and an integrated packet router for managing communication within and between nodes. In the node, the processors are programmable, which allows them the simulation a couple hundreds of neurons, depending on the synaptic and neural complexity, and the learning rule. By running multiple nodes in parallel, SpiNNaker can perform large-scale neural simulations.

SpiNNaker 2 [69], the successor to SpiNNaker, aims to significantly increase the simulation capacity by over 50 times through numerous improvements compared to the older architecture, including a smaller production technology of 22 nm compared to the original's 130 nm. Additionally, the new architecture features hardware-acceleration and adaptive scaling of voltage and frequency, which adjusts according to the current workload of the nodes, further enhancing the efficiency of the platform. Furthermore, SpiNNaker 2 is designed to serve a broad spectrum of applications extending beyond neural simulations, including ML, robotics, and cybersecurity, to name a few. Overall, SpiNNaker 2 promises to revolutionize large-scale computing and usher in a new era of advanced applications.

TrueNorth TrueNorth [28] is a fully digital neuromorphic platform designed for large-scale neural simulations. With its high capacity to simulate up to one million neurons, TrueNorth is built from 4,096 neurosynaptic cores, each of which comprises a self-contained neural network consisting of 256 LIF neurons. Each of the LIF neurons is equipped with 256 input and 256 output lines and a 256×256 matrix to specify synaptic connections. The cores enable a single TrueNorth chip to accommodate up to 256 million synapses in total, with the potential for larger networks by interconnecting multiple chips into larger clusters. TrueNorth's digital architecture is well-suited for efficient and reliable simulations of large-scale neural networks. Its ability to handle such simulations has potential applications in various fields such as cognitive computing, ML, and robotics. Additionally, TrueNorth's high scalability makes it a valuable tool for investigating the computational mechanisms underlying neural information processing.

Loihi The Loihi chip [27] is a state-of-the-art digital neuromorphic platform that incorporates 128 cores, each of which has the capacity to simulate 1,024 neurons. These cores share identical fan-in and fan-out connections, as well as configuration settings, while ten onboard memory blocks store essential information, including spike traces used for learning, synapse memories and mapping information, and neuronal state variables. The on-chip learning mechanism is also fully programmable through 4-bit microcode. The Loihi chip uses the LIF model and current-based synapses to simulate the neurons.

To ensure efficient event processing, the Loihi chip uses synchronizing messages to propagate spikes generated by neuromorphic cores through the grid structure of the chip. Additionally, the cores operate asynchronously apart from this synchronization. Like SpiNNaker, the Loihi architecture is designed for scalability, making it possible to

Table 2.2: Analog and mixed analog/digital neuromorphic hardware solutions.

Name		**Year**	**Neurons**	**Syn.**	**Models**	**Process** [nm]	**Die Size** [mm^2]
HiCANN	[97]	2010	512	100k	AEIF	180	55
HiCANN-DLS	[98]	2016	512	131k	LIF	65	32
NeuroGrid	[99]	2014	64k			180	168
Braindrop	[100]	2018	4096	65k		28	0.65
DYNAP-SEL	[101]	2017	1088	78k	LIF	180/28	7.28
Rolls	[102]	2015	256		AEIF	180	51.4
Buhler2017	[103]	2017	512		LIF	40	

combine multiple chips to clusters. For instance, the most extensive Loihi-based system consists of 768 chips with 98,304 cores and 100M neurons, making it ideal for simulating large-scale neural networks.

Recently, Intel Corporation announced a successor to Loihi, Loihi 2 [96]. However, there is limited information available about the underlying architecture. Nonetheless, Loihi 2 is expected to improve on the current design of the Loihi chip and advance the field of neuromorphic computing.

2.2.3 Analog and Mixed Analog/Digital Implementations

Analog neuromorphic hardware can be divided into two subcategories: namely superthreshold-operating implementations for processing at higher rates and subthreshold-operating implementations for higher energy efficiency. Mixed analog/digital neuromorphic systems commonly use analog circuitry for processing components, like neurons and synapses, and implement digital components, to store synaptic weights or some components of the memory, to increase reliability.

Superthreshold Analog Neuromorphic Hardware

Superthreshold analog neuromorphic hardware can be further classified into flexible Field-Programmable Analog Arrays (FPAAs) and custom implementations. FPAAs like the Field-Programmable Neural Array (FPNA) [104] and NeuroFPAA [105] have been developed to provide a high degree of flexibility in analog computation. Custom implementations such as BrainScaleS [97]. offer specialized hardware that can efficiently perform specific neural computations.

HICANN/BrainScaleS High Input Count Analog Neural Networks (HICANNs) serve as chiplets for the hierarchic neuromorphic system, BrainScaleS [97]. The architecture and communication protocols of BrainScaleS have been tailored for large NN configurations. The HICANNs in BrainScaleS consists of AEIF neurons [60], which provide the option to be reduced to regular IF model if necessary. One significant advantage of HICANNs is their fast timescale, achieving a 10^3 to 10^5 acceleration over equivalent biological circuitry. Adjustments to the time constants of the neurons are made by varying the membrane capacitance and the leak conductance. Additionally, the fast time constants are due to the small electrical capacities, which in turn have compact physical dimensions. Interestingly, as the membranes are implemented as Metal-Insulator-Metal (MIM)-capacitors, allowing them to be integrated without increasing the overall surface area, making the design compact and efficient.

The Analog Network Cores in BrainScaleS comprise 512 membranes that can be recombined to form neurons, two blocks of synapse arrays, and synaptic drivers. The latter converts digital address events into analog currents using 4-bit Digital-to-Analog Converters (DACs). Furthermore, each neuron can receive over 14k synapses, allowing for the simulation of large-scale neural networks. BrainScaleS can simulate up to 180k neurons using 352 HICANN chiplets housed on a single wafer, and the reference system of BrainScaleS comprises 20 wafers in total.

The next generation of HICANN chiplets, High Input Count Analog Neural Network with Digital Learning System (HICANN-DLS) [98], have incorporated more modern manufacturing processes, higher bit-precision for neurons and synapses, and onboard learning capabilities. BrainScaleS' design makes it a promising candidate for simulating large-scale neural networks.

Subthreshold Analog Neuromorphic Hardware

Neuromorphic hardware operating in subthreshold mode is employed to increase power efficiency. Carver Mead'S original definition of neuromorphic hardware refers to analog circuitry that operates in subthreshold [25]. Examples of subthreshold neuromorphic hardware are Braindrop [100], Neurogrid [99], and DYNAP-SEL [101]. These implementations strictly speaking fall under the mixed analog/digital group as they use digital communication frameworks.

Of particular significance is the property of subthreshold analog neuromorphic hardware, which is characterized by inhomogeneities in the silicon transistors, which can result in inaccurate spiking thresholds of neurons due to device mismatch, shot noise, and thermal noise [106]. These sources of mismatch can either be minimized at the device level [107] or exploited for computational purposes [108].

Braindrop and Neurogrid Braindrop [100], a mixed-signal processor, was specifically designed for the use with Neural Engineering Framework (NEF) [109] networks. NEF facilitates the transcription of ODEs to hardware. This is achieved by exploiting the inherent imperfections of analog circuitry to produce the desired neuronal variability. Braindrop is equipped with 4096 neurons with a weight memory of 64 KB that has been designed to store 16 8-bit synapses for each neuron.

In large neuromorphic systems, digital communication consumes a significant amount of power. Therefore, the authors of Braindrop have focused on spatially and temporally sparse communication using sparse encoding, as well as accumulative thinning. While the communication is digitally implemented, the neurons are implemented analog designed for subthreshold operations. Because of the mixed-signal design and communication scheme, Braindrop exhibits stronger neuronal density and significantly smaller energy consumption for each synaptic operation than its full digital counterparts [100].

As a successor of the large-scale system Neurogrid [99], Braindrop is set to serve as fundamental blocks for the Brainstorm chip, that has been designed for the implementation of multicore systems containing millions of neurons [100]. By incorporating Braindrop in this way, the authors are looking to develop even larger and more sophisticated neuromorphic systems.

DYNAP-SEL The DYNAP-SEL processor represents a highly advanced mixed-signal chip, designed for scalable routing infrastructure with the ability to combine multiple chips [101]. This chip is composed of four cores for neural processing and one additional core featuring plastic synapses. The non-plastic cores have the capacity to house 256 neurons each, and each included neuron is equipped with 64 4-bit synapses. The supplementary core has 64 neurons combined with 128 plastic synapses, equipped with on-chip learning and 64 programmable synapses per neuron. This novel architecture empowers large network simulations with efficient on-chip learning, structural plasticity, and strong biological plausibility. With its cutting-edge features and advanced technology, the DYNAP-SEL chip is poised to transform the field of neuromorphic engineering.

2.2.4 Summary

Various approaches have been developed to efficiently execute SNNs on specialized hardware. These approaches include software-based solutions, digital FPGAs and ASICs and analog compute elements, operating in superthreshold or subthreshold mode, each with their unique advantages and disadvantages.

For instance, many-core systems such as Loihi [27] and TrueNorth [28] use digital processors that can efficiently simulate large networks, but with restrictions in neuron and synapse models. On the other hand, SpiNNaker [85] is extremely flexible since neurons and synapses are computed entirely in silico, albeit with larger energy usage. Analog implementations, however, can replicate biological behaviors much faster than their biological counterparts. Implementations like Neurogrid [99] operate in subthreshold mode at very low energy, whereas superthreshold approaches like BrainScaleS [97] operate at a much higher rate. However, these implementations are sensitive to ambient temperature, process variations, and noise.

As there is no universal solution that fits all applications, this work aims to give a brief overview of the approaches and consider their properties in the simulation environment if necessary. The inhomogeneities of the silicon transistors mark the concept idea of Chapter 5 for the processing of sequential data.

2.3 Training of Spiking Networks

The most popular methods for training conventional ANNs commonly rely on stochastic gradient descent and error backpropagation. However, these methods require differentiable activation functions and therefore require modifications for the use with binary values such as the action potentials used in SNNs. Over the past years, this has led to the development of several strategies for training deep SNNs. These can be broadly categorized into three main categories:

1. *Biologically plausible learning algorithms.* This approach involves the use of local learning rules at the synapses, such as STDP [33], [34], [110]. STDP is a computationally expensive approach that is used for more biologically realistic training for example in computational neuroscience simulation.
2. *Supervised learning with spikes.* This approach involves the direct training of SNNs by using variants of error backpropagation through the implementation of surrogate gradients [111]–[113].
3. *Conversion Methods.* Conventional deep ANNs are trained with gradient descent-based learning algorithms, and then the continuous-valued activation functions of the ANN are converted into spiking neurons. This approach allows the use of existing deep ANNs architectures and training techniques, but it requires additional hardware resources to implement the spiking neurons. Special subcategories of conversion approaches are *contrain-then-train*, where the original ANN is re-trained under constraints designed to emulate the characteristics of the spiking neuron models, and *binarization*, where conventional ANNs are trained with binary activations.

2.3.1 Biologically Plausible Learning Algorithms

Understanding how the training of hierarchically organized NNs can be carried out with local learning rules such as Hebbian learning [114] or STDP [33], [34], [110], is of great interest for neuroscience. The usage of local learning rules is beneficial as it allows the detection of spatio-temporal patterns and would enable hardware-efficient ways of training. However, purely local learning rules pose a challenge for deep networks as backpropagation of supervised error signals is difficult. To overcome this, recurrent feedback connections are introduced to modulate learning in lower layers in most studies investigating local learning in hierarchies.

Hierarchically organized NNs, such as the brain, frequently utilize feedback connections from higher to lower layers [115]. Another approach, random backprojections of error signals [116], has been shown to be effective for training lower layers in deep SNNs [117] and networks with spiking multi-compartment neurons [118].

STDP's function depends on the applied network architecture. In competitive networks, STDP can solve unsupervised tasks like clustering [119], [120], as shown by recent work training competitive convolutional networks with unsupervised filters [121], [122]. Spiking neurons can be connected as restricted Boltzmann machines to approximate contrastive divergence in an event-based way [36], [123], and this can extend to multi-layer Deep Belief Networks (DBNs) for layer-by-layer training. Spiking CNNs [124], [125] and autoencoders [126] can also be trained with unsupervised STDP layer-by-layer, but a supervised learning approach is required for the output layer [127]. Reward-modulated STDP with multiple layers has been introduced to obtain fully spiking supervised training [128].

2.3.2 Supervised Learning with Spikes

Various supervised learning methods for SNNs have been presented. These methods implement the learning on the spike level and typically use variants of backpropagation to train deep spiking networks. Unlike methods using local learning with STDP, they do not necessarily aim for biological plausibility. Spike-based learning rules offer the advantage of not being constrained to mean-rate codes, making them a good choice for usage with spatio-temporal patterns from inputs like event-based sensors.

Together with several single-layer learning methods, such as ReSuMe [129] or Tempotron [130], many spike-based learning rules for multilayer SNNs rely on a differentiable proxy, enabling backpropagation. Early attempts at this included SpikeProp [131] and its variants [132], [133], but they are computationally expensive and not used for modern deep learning applications. Lee et al. [134] presented a backpropagation approach based on spikes that train deep SNNs used for classification tasks directly from spike signals. This is achieved by performing stochastic gradient descent on real-valued membrane potentials and low-pass filtering discontinuities at spike times.

O'Connor and Welling [135] suggest using signed spikes in an SNN approximating a ReLU-activated deep multilayer perceptron. Stromatias et al. [127] fine-tuned the output layer of deep SNNs by performing backpropagation on histogram bins. Mostafa et al. [136] make usage of the timing of the first action potential for every neuron as its activation feature during training, resulting in a sparsely firing network.Jin et al. [137] propose a hybrid model that combines a long-term gradient descent-based rule for a rate-encoded error signal with a short-term update. Spike-based learning approaches for supervised training have been increasing in number but the benefits of these approaches for ML on neuromorphic sensor data remain largely unexplored, but exploiting temporal codes may lead to greater performance gains.

2.3.3 Conversion Methods

ANNs trained with conventional approaches, can be converted into SNNs by adjusting the parameter and weights of the pulsed neuron to achieve the same input-output mapping as the original networks (see Figure 2.6). The first conversion approaches were developed to process event-based data using CNNs. First attempts used manually programmed convolution kernels [138], while others introduced systematic approaches to map conventionally trained CNNs to SNNs [139]. This approach translates the continuous-valued activations of the ANN into firing rates of the SNN's neurons. The network's weights necessitate a rescaling according to the parameters of the spiking neuron, what introduces hyperparameters that have to be set prior to the conversion. Diehl et al. (2016) presented a novel approach for the conversion of RNNs under the constraints of neuromorphic systems (see Section 5.1) [140].

The conversion approach allows for the usage of deep learning techniques in SNNs, which has resulted in state-of-the-art performance on classification tasks [43], [141], [142]. Conversion from ANNs to SNNs is a simple process that adds minimal training overhead, with negligible deviations in accuracy. However, ANNs cannot always be easily converted to SNNs, particularly when dealing with negative activations, which are incompatible with the positive firing rates of SNNs and generally require an additional inhibitory spiking neuron. This problem can be addressed by using ReLU activation functions [51], which are mainly linear and have only positive activations. Sigmoid activation functions, on the other hand, require additional approximations and introduce errors [143]. Another problem arises when dealing with softmax layers at the output, which can have negative activations. This problem can be solved with practical solutions, for example by accumulating all outgoing spikes in a separate layer and using a conventional softmax classification [144].

Most approaches for converting CNNs to SNNs struggle with realizing max-pooling operations [145], a common feature of analog deep networks. This is because the maximum operation is non-linear and therefore is difficult to be computed with spikes. To get around this issue, many approaches replace max-pooling with average pooling [143], [146], [147], for ease of implementation in SNNs but leads to accuracy loss. However, a max-pooling mechanism was presented by Rueckauer et al. [144] that uses gating functions to only pass action potentials from the neuron with the highest firing rate, leading to better accuracy.

While scaling all weights in a layer in ReLU-activated networks does not change the final output, SNNs are sensitive to weight scaling as spiking neurons comprise a maximum spike frequency. To address this, Diehl et al. [146] propagated a subset of training examples through the network to rescale input weights to each layer. Sengupta et al. [147] and Rueckauer et al. [144] further extended this method to improve the results for deep NNs by increasing robustness against outliers and incorporating the actual firing rates during weight normalization into account.

Conversion with weight normalization can lead to increased spike production and decreased energy efficiency, in non-ideal conditions, the converted SNN needs more spiking operations than the original ANN. Alternative spike codes based on timing information are being researched to address the inefficiencies of rate-coded conversion. These approaches, such as HFirst [149], time surface features [150], and HATS [151], capture spatio-temporal dynamics and utilize event-based sensors. Asynchronous pulsed Sigma-Delta coding is also introduced [152] to maintain accuracy while utilizing fewer spikes.

In contrast to this conversion method that converts fully trained ANNs into SNNs, Esser et al. [153] introduced the term *constrain-then-train* to describe an approach that incorporates the training of the ANN under the constraints of spiking neurons or the

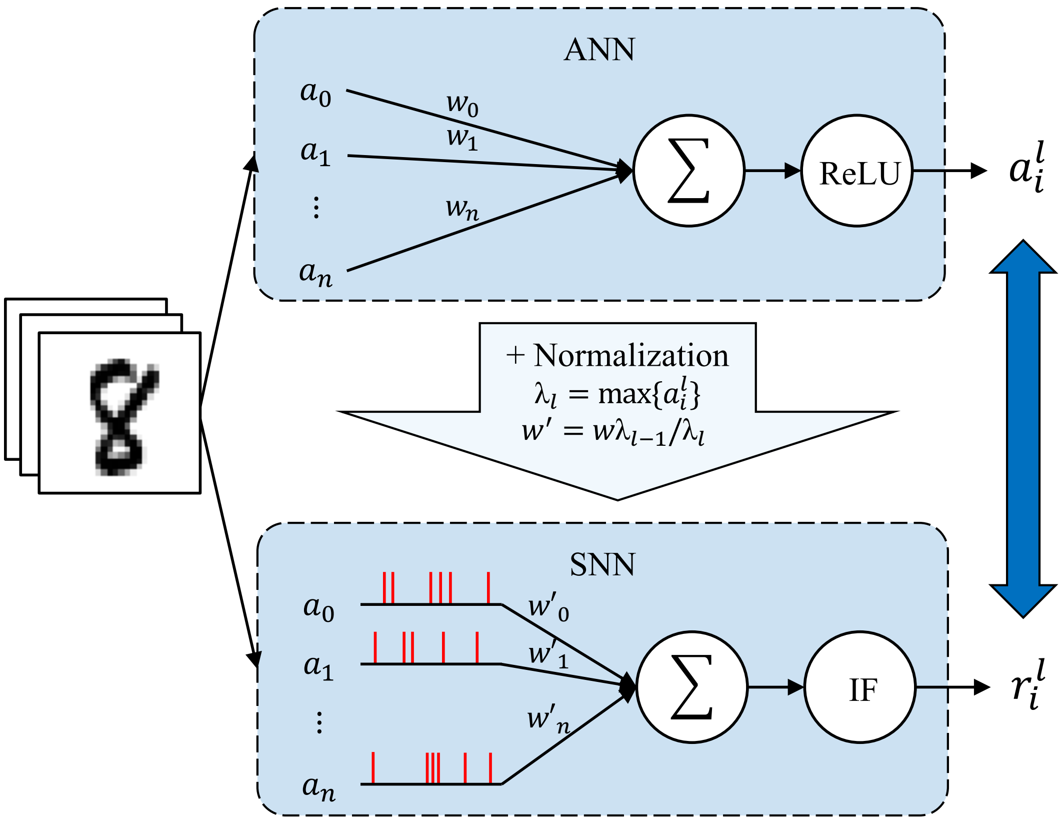


Figure 2.6: Conversion approach. Visualisation of the basic conversion of a ReLU-activated ANN to SNN with a spiking IF neuron (Figure adapted from [148])

target hardware. Constrain-then-train methods apply conventional gradient descent-based learning algorithms to learn weights under spiking constraints, and then convert the ANN into an SNN without the need for further weight scaling. While conversion algorithms also impose some constraints on the ANN, constrain-then-train methods train the ANN for a specific set of spiking neuron model parameters and require complete retraining if these parameters change, unlike conversion methods which map weights for arbitrary parameters.

Constrain-then-train models can potentially adapt better to the target platform, and show no conversion loss, but require more complex retraining of the whole ANN. Esser et al. constrained the continuous-valued weights and activations of the training network between $[0, 1]$, matching the TrueNorth platform constraints [28]. This approach yields highly accurate classifiers on MNIST at low energy costs and was further improved and extended to multi-chip setups [154]. For more realistic models, non-differentiable transfer functions relating the input current to the neuron parameters can be approximated by modeling variability in input spike trains.

One step further is the method of *binarization* of ANNs by reducing the activations to binary values. In binarized networks, the information is propagated in a synchronized, layer-by-layer manner like in conventional ANNs, which lacks asynchronous information processing. One advantage of binarization is the energy-efficient benefits for execution on neuromorphic systems due to sparse activations and computation on demand. Additionally, binarization reduces computational costs on conventional hardware, as the memory bandwidth and multiply-add operation complexity are reduced [155]. Networks with binary activations typically use lower-bit weight representations.

Given that, today's large ANNs require long training times, it makes it impractical to retrain the entire network for the constrain-then-train and binarization methods on an industrial scale. This thesis, therefore, emphasizes the general approach utilizing weight normalization.

3 Conversion Challenges and Opportunities

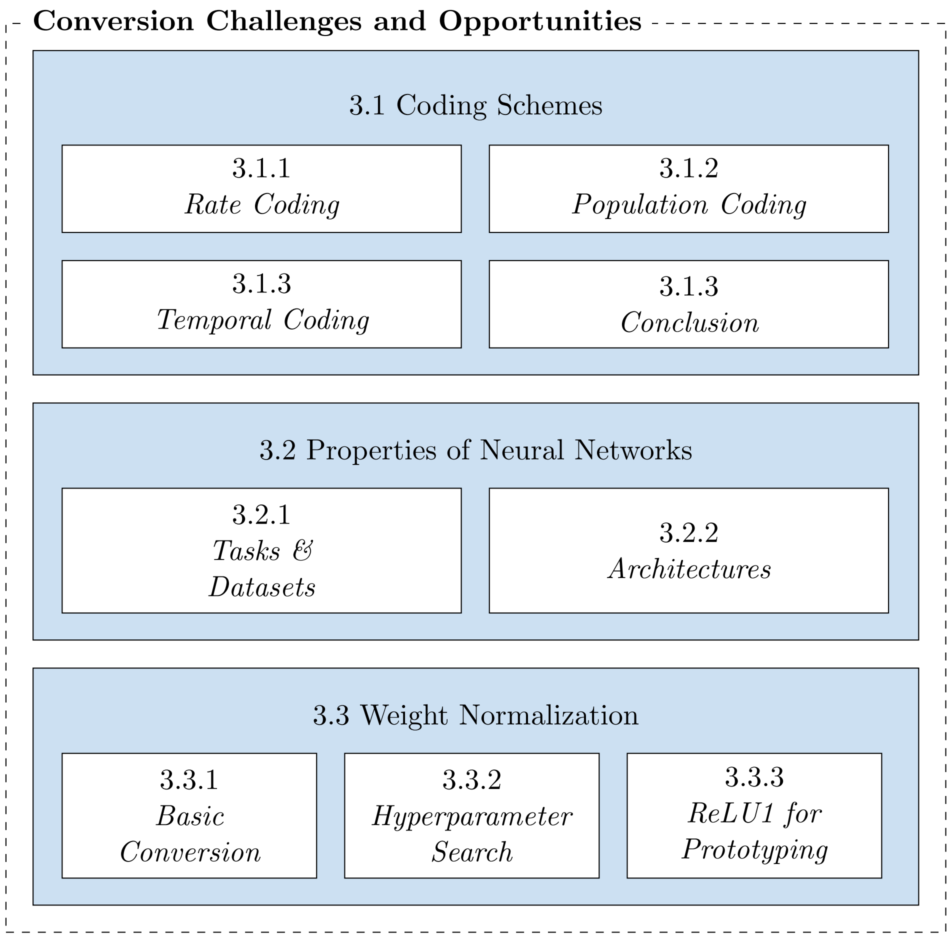


Figure 3.1: Chapter structure.

The majority of conversion approaches capitalize on the linear relation of both the ReLU activation function and the firing rate of spiking neurons in response to their input. This rate-coded conversion technique exhibits great performance but has to generate many action potentials for converging to its highest accuracy. In this chapter, we examine various encoding schemes and assess their viability for achieving enhanced conversion outcomes. Subsequently, we scrutinize existing ANN architectures and identify the ones that lack a conversion mechanism. Lastly, we explore the normalization procedure that is imperative for the conversion process and explore techniques for approximating it to expedite development.[1]

3.1 Coding Schemes

The conventional conversion technique employed in SNNs involves substituting the activation function of the ANNs with a spiking neuron. The input to the neuron is directly proportional to the firing rate, thereby encoding the information as a rate-coded signal. Unfortunately, this method can be substantially energy-intensive, and recent literature indicates that SNNs must use less than 1.72 spikes per neuron to achieve improved energy efficiency than state-of-the-art hardware accelerators running conventional ANNs [156]. Here, we summarize the available signal encoding schemes that have been presented in the literature and discuss which can be used for the ANN-to-SNN conversion.[2]

Given that the shape of an action potential is the same every time, information has to be carried in the presence or absence of a spike [61]. Drawing upon the principles of biology, it is evident that diverse coding schemes exist that are specialized for processing specific types of data. For instance, in our eyes, photoreceptor cells convert variations in light intensity into patterns of spikes, while microscopic hair cells within the inner ear translate fluctuations of air pressure into frequency-selective trains of action potentials. Similarly, chemical receptors within our olfactory system emit spikes in response to specific molecules within the air.

The encoding of information in neural systems has been a topic of interest in neurobiological research for several decades. Coding schemes can be broadly categorized into three groups based on whether the explicit timing of spikes and their order is necessary for information transmission and if the encoding is done by a single or a population of neurons:

- *Rate codes* encode information in the instant or averaged spike rate of an individual neuron. It can be further subcategorized to *count rate codes* and *density rate codes.*
- *Temporal codes* depend on the exact timing of individual spikes, which can be understood in relation to a fixed reference time, the interspike intervals, or spike arrival order. In this context, even slight variations in spike timing can drastically alter temporal coded information. Temporal codes can be further categorized (see Figure 3.5).

[1]Parts of this chapter have been previously published in [39], [41], [43].
[2]Parts of the following section have been published in [41].

- *Population codes* require multiple neurons for representing the desired code. However, since that can happen both in temporal and rate codes, this group is sometimes combined with either one of them.

Rate and temporal codes can furthermore be subcategorized to distinct coding schemes. The classification of these individual schemes often involves ambiguity. A simple definition based on the importance of the precise timing or order of the spikes for the transferred information is used here. Accordingly, if precise spike timing or order is crucial, it is a temporal code; otherwise, it is a rate code.

For a long time, there was a consensus that biological systems primarily used rate codes. However, subsequent research did suggest that exact spike times are also utilized to encode sensory perceptions. Thorpe's experiment demonstrated that the human visual system can process new stimuli in less than 150 ms, indicating that a rate code describing the retinal image is very unlikely. These findings were later supported by publications in visual, audio, tactile, and olfactory systems. In the latter system, experiments with mice showed that they could discriminate between simple odors within 200 ms, and when the smells were similar, distinction took 100 ms longer, suggesting the temporal integration of information.

Building on these biological principles, a range of coding schemes can be adapted for AI use cases. However, the suitability of coding schemes depends on the use case and the type of input. Networks dealing with rapid-changing input that require fast feedback will probably not use rate-based encoding schemes. In contrast, rate codes might be useful for networks handling low-frequency, high-dimensional data. Currently, there is no universal answer on which coding scheme performs best.

Figure 3.2 illustrates data encoding utilizing assorted schemes. For instance, an image sequence can be encoded to spikes. In the case of the rate-based count code, the intensity of every pixel in the individual frames is represented by the number of spikes emitted per frame. On the other hand, Time-to-First-Spike (TTFS) represents a temporal coding where only a single action potential is emitted for each pixel and frame, and the exact spike time encodes the brightness. Temporal Contrast (TC) encodes the continuous change of the light intensity over time, where positively or negatively valued action potentials are transmitted once a relative intensity change exceeds a threshold. Specialized cameras have been presented that make use of this event-based coding type.

In conclusion, the encoding of information in neural systems can be grouped in two broad categories, rate coding and temporal coding, with further subcategorization into distinct coding schemes. The importance of temporal codes in biological systems has been demonstrated by various experiments. These biological coding schemes can be adapted for artificial applications, depending on the application and input

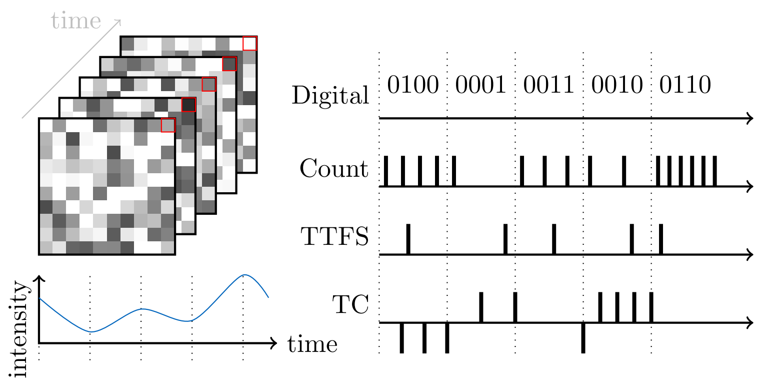


Figure 3.2: Exemplary coding schemes for a sequence of images over time. The intensity-time plot indicates the changes of the pixel value in the red square as a continuous function. The dashed lines indicate the time instances at which the images have reached the colour value. Digital, count, and TTFS spikes in correlation to the local minima and maxima in the intensity curve. The TC emits spikes if the continuous intensity change exceeds a certain threshold. (Figure previously published in [41])

3.1.1 Rate Coding

Rate codes are a widely used type of encoding scheme that can furthermore be grouped into two subcategories: count rate and density rate coding. A schematic visualization of the encoding resulting from a random input can be found in Figure 3.3. The definitions are founded on the publication on neuronal dynamics by Gerstner et al. [61].

Count Rate – Average Over Time

Count rate codes, also referred to as frequency coding, are the most commonly used coding scheme and are defined by the mean firing rate

$$r(t) = \frac{N_{\text{spike}}}{\Delta t} \tag{3.1}$$

where N_{spike} represents the number of spikes and Δt represents the time interval. This type of rate coding can be used to encode any analog value with slow variations, such as pixel intensities or gas concentrations.

Count rate codes can either have exact or random spike times. The random spike times are typically modeled using a Poisson distribution. The reconstruction error as a result to the discretized number of action potentials in a given time interval decreases by ${}^{1}/_{N_{\text{spikes}}}$ as the number of spikes increases. However, the variations in Poisson-distributed spike trains result in a decrease in error only proportional to the square root of the number of spikes ${}^{1}/\sqrt{N_{\text{spikes}}}$ [157].

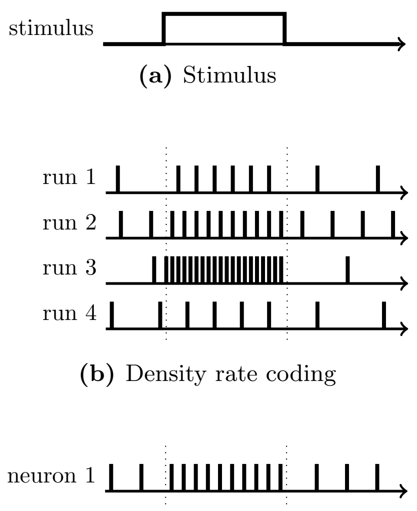


Figure 3.3: Visualization of rate coding techniques. The exemplary stimulus is a wide pulse (a). The dashed line in the encoding visualizations (b-c) indicates the rising and falling edge of the stimulus. (Figure adapted and previously published in [41])

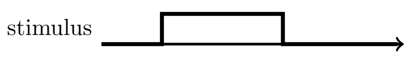
As both the firing frequency of rate-coded neurons, as well as the activation of the glsrelu function, increase linearly with their input, rate codes are well suited for the ANN-to-SNN conversion. Because of this property, they represent the state-of-the-art for conversion.

Density Rate – Average Over Multiple Runs

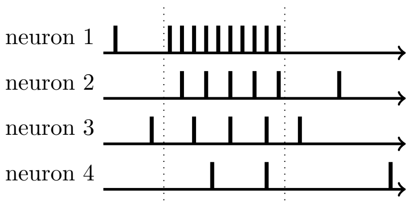
Density rate codes, on the other hand, are not a biologically plausible encoding method. In this type of rate coding, the neuronal activity is measured over multiple simulation runs, and the result of the neural response is presented in a Peri-Stimulus-Time Histogram (PSTH). The spike density $p(t)$ is defined as the number of spikes $N_{\text{spikes,K}}$ in a time interval $[t; t + \Delta t]$ divided by the duration of the interval Δt and the total number of iterations K:

$$p(t) = \frac{1}{\Delta t} \frac{N_{\text{spike}}(t; t + \Delta t)}{K} \tag{3.2}$$

This type of rate coding can be beneficial in stochastic SNNs, where multiple simulations with the same inputs result in different outputs. This approach can be useful for



(a) Stimulus



(b) Population rate coding

Figure 3.4: Visualization of population coding techniques. The exemplary stimulus is a wide pulse (a). The dashed line in the encoding visualization (b) indicates the rising and falling edge of the stimulus. (Figure adapted and previously published in [41])

evaluating conversion methods that include noise but are not directly applicable to real-world applications.

3.1.2 Population Coding

Population codes are based on comparable properties of neurons in a population. A schematic visualization of the codes resulting from an arbitrary input can be found in Figure 3.4.

The firing rate $A(t)$ is defined as the number of spikes of N neurons averaged in a given time interval $[t; t + \Delta t]$:

$$A(t) = \frac{1}{\Delta t} \frac{N_{\text{spike}}(t; t + \Delta t)}{N} \tag{3.3}$$

A population of neurons is not required to exhibit a uniform spike response to a given stimulus. If every neuron has a known tuning curve, the superposition of the responses in a large population can encode single numbers, vectors, or even function fields. Population codes play an important role in the n the NEF [109].

Given that this encoding scheme averages a firing rate over time, similar to the count rate coding but with a population of neurons, it is suitable for the ANN-to-SNN conversion.

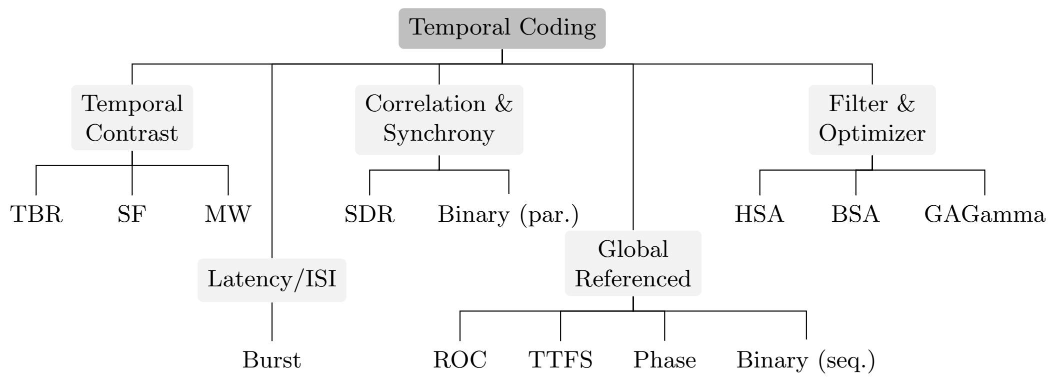


Figure 3.5: Taxonomy of temporal coding techniques. Temporal codes use the precise timing of spikes to encode information. (Figure previously published in [41])

3.1.3 Temporal Coding

As shown in Figure 3.5, temporal codes are further divided into several subcategories, each with its unique processing mechanisms of the input signal. TC codes concentrate on the derivative of the input signal, while globally referenced codes are processing the stimulus in fixed-size packages relative to a periodic signal or oscillation. Interspike Interval (ISI) codes analyze the relative timing between groups of spikes, while correlation codes rely on the simultaneous activity of multiple neurons. Filter and optimizer-based approaches generate spike patterns based on the comparison of the input and a predefined kernel function. Figure 3.6 provides a visual representation of the various temporal encoding schemes in relation to a stimulus. It's worth mentioning that binary codes, Ben's Spiker Algorithm (BSA), and TC utilize a different input stimulus in the illustration.

Global Referenced Codes

Codes that are globally referenced encode the incoming information in the interval between action potentials relative to a static or periodic reference. The encoded information is thus processed in packages between two successive points of reference.

The initial action potential of all globally referenced encoding mechanism is often the element with the highest importance, similar to to binary representations. This dynamic relationship between network parameters and spike timing leads to an intriguing consideration for optimizing output neuron thresholds, balancing speed and accuracy according to the specific requirements of each application. As a result, the network can

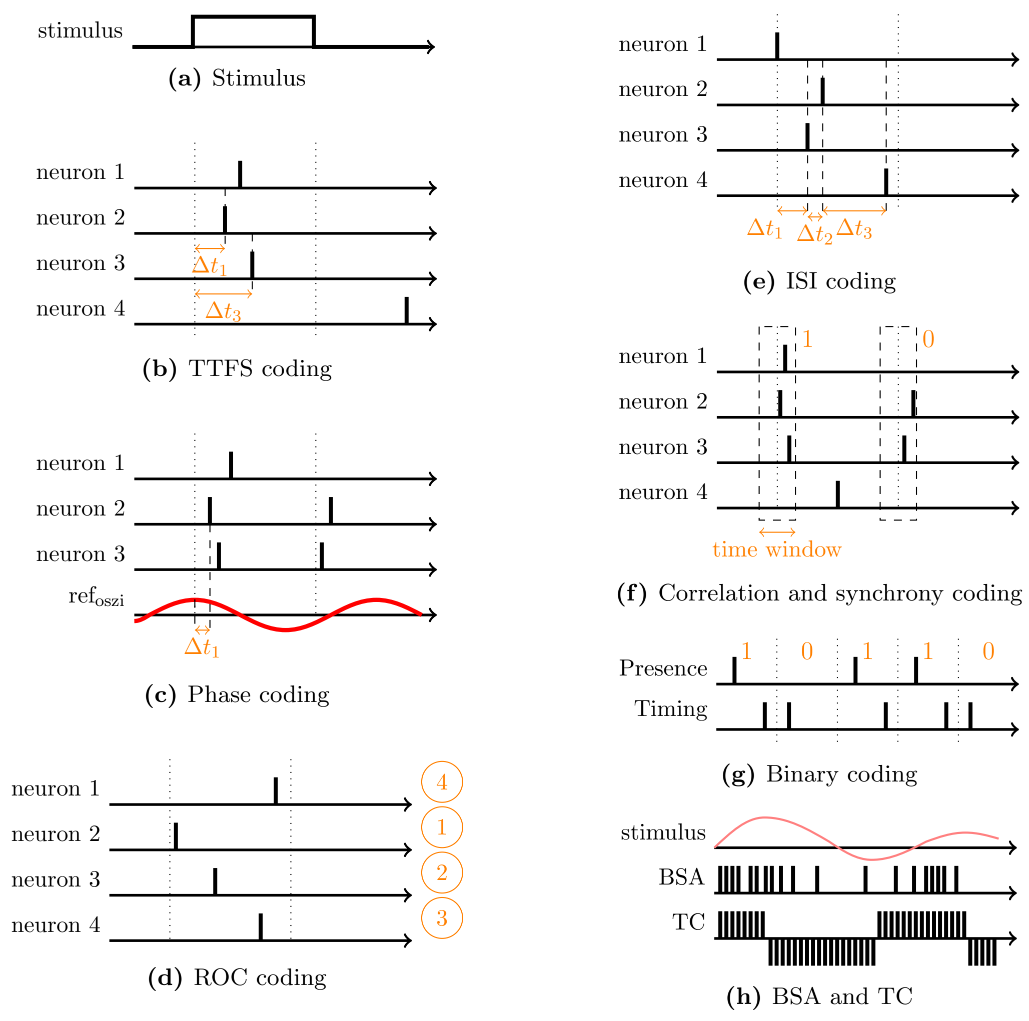


Figure 3.6: Visualization of temporal coding techniques. The wide pulse stimulus in (a) is used for the visualizations in (b-g). The dashed line indicates the rising and falling edge of the stimulus. Δt describes the latency between the reference point and the spike. In (d), the order of spikes is numbered on the right. The stimulus for the coding visualization in (h) is the sinusoidal wave, which is directly given in the same sub figure. (Figure previously published in [41])

anticipate and generate the output pattern prior to completing the processing of the entire input stimulus [158].

One of the simplest temporal coding schemes is **TTFS** coding, which conveys information through the time interval Δt between the stimulus onset and the generation of the first action potential in a neuron. Firing times can be related to the stimulus amplitudes in various ways. For example, the firing time can be inversely proportional to the amplitude of the stimulus, such that $\Delta t = 1/a$ or it can be linearly related to the amplitude $\Delta t = 1 - a$, where a is the normalized signal amplitude. For each of these cases, high amplitudes result in early firing times, while low amplitudes lead to longer intervals or no spikes at all.

Johansson and Birznieks have demonstrated the importance of TTFS in encoding information, as they identified that the relative timing of the initial action potential in response to a discrete mechanical stimuli carries information about the direction and force [159]. Similarly, Gollisch and Meister, in their study of the retinal pathway, observed that TTFS is robust to noise variations and invariant with respect to stimulus contrast [160]. However, it is important to note that they used the term "latency coding" to describe TTFS, which can lead to confusion with ISI coding, as the definition of latency between spikes and a global reference or between multiple spikes is not well-defined.

TTFS codes have been used in conversion approaches before [38]. Although this approach performed well on shallow networks, it failed to do so in our experiments for deeper networks [45] (see Section 6.1).

In contrast, **phase coding** conveys data through the relative timing of action potentials with respect to a periodic reference signal [161], [162]. In this scheme, each neuron fires in relation to a reference signal and encodes information similarly to TTFS. This behavior was observed by Gray, König, Engel, and Singer in their study of the firing probabilities of neurons in the cat visual cortex [163].

This coding scheme is very well suited for the ANN-to-SNN conversion and marks the basis for our method in chapter 6.

Another type of globally referenced coding scheme is **ROC**, which relies on the relative ordering of spikes within a population of neurons with respect to a common temporal reference [158], [164]. Unlike TTFS, Rank-Order Coding (ROC) encodes information without the consideration of the precise timing of action potentials, functioning as a discrete normalization filter that loses absolute amplitude information. Therefore, it is not possible to reconstruct the absolute signal amplitude or an exactly constant signal and therefore it is not a suitable coding mechanism for the conversion approach.

Another globally referenced scheme is **sequential binary** coding, where every spike corresponds to a "1" or "0" within a bit stream. Relative to a fixed reference clock, there are two schemes for encoding bits: the presence or absence of an action potential within a given interval [165] or the spike time within the interval [166]. In the first scheme, a logical "1" stands for a spike present during one clock cycle, while in the latter, the clock cycle is divided into two sub-intervals: if an action potential is present in the first half, a "0" is encoded, and if it is present in the second half, a "1" is encoded. This guarantees the continuous generation of action potentials across all possible binary patterns being represented.

This encoding scheme fails to be directly translated from continuous-valued activation functions into spiking neurons. Therefore it is omitted from further investigation in the context of ANN-to-SNN conversion.

ISI Coding

ISI or **latency coding** is a method that embeds information in the temporal gap (latency) between action potentials of within a specific population of neurons [167]. This coding scheme has been observed in various studies to be highly dependent on the stimulus intensity. Pyramidal cells, in particular, have been noted to exhibit this phenomenon [168]. There are several nuances to how ISI coding functions. For example, Li and Tsien [169] have theorized that rare events such as prolonged silence periods contain more information compared to periods of higher spike activity.

A further subcategory of ISI coding is known as **burst coding**, which is characterized by the conversion of the input to discrete inter-spike latencies. A burst, as defined in this context, is a group of action potentials with a very minor ISI [170]. The distinction between a normal spike and a spike that is part of a burst is dependent on the ISI threshold as well as the expected number of spikes within the group [171], [172].

In contrast to the activation functions found in ANNs, encoding information with ISI relates the spike times to those of neighboring neurons. This encoding would necessitate the creation of additional connections between the neurons of a layer in the converted SNNs, which we were not able to find general valid rules.

Correlation and Synchrony

Correlation and synchrony coding in neural activity involves the utilization of temporal references to other spiking neurons. This results in the representation of the input patterns by groups of spikes with relative short ISIs [61]. The information is coded in the correlation that emerges from the simultaneous firing of specific neuron groups, thus forming a correlation between their spiking patterns.

SDR are a subcategory of correlation and synchrony coding [173], [174], where only a neuron subset within a population shos activity at any given time, enabling representation of an effectively infinite number of patterns with minimal errors [175]. In the extreme scenario, information is encoded by the activation of a single neuron at a time, referred to as **amplitude coding**. Here, the strength of the signal is directly encoded into the activity of a single neuron, with a spike generated as soon as a specific value is crossed.

Studies have observed the general synchronous coding scheme inside the somatosensory cortex in monkeys [176] or in the visual cortex in cats [163], [177], hypothesizing that synchrony provides evidence of the significance of incoming stimuli. Grid and place cells [178], [179] provide a biological example of synchrony coding, where spatial information is encoded through the synchronized activity of a particular population of neurons.

Synchronous firing of neurons can also be interpreted as **parallel binary** codes, where each neuron is encoding a specific bit of a larger word. In contrast to sequential binary codes, where a single neuron encodes information through precise timing within a stream, parallel binary codes encode the whole word at once.

Although the correlation and synchrony codes offer the potential to an even sparser activation of the spiking neurons, it necessitates the usage of connections between neurons of a layer. Therefore, this coding scheme can not be directly used for the ANN-to-SNN conversion.

Filter and Optimizer

In the field of neuroscience and control theory, one common approach to describe a system is through the examination of its input-output behavior. In this regard, the input is a known analog stimulus, the system is represented by one or multiple neurons, and the output is a spike train. However, instead of measuring the output in response to a known input signal, **HSA** [180] and its successor **BSA** [181] adopt a reverse approach by using a pre-specified filter to generate the spike train corresponding to a known input signal. The method operates by generating a spike whenever the convolution of the signal and the filter surpasses a defined threshold.

It is worth mentioning that this approach is limited by the range of inputs that it can process, thus the input signal needs to be normalized before conversion.

In an attempt to interpret the encoding process as a data compression problem with prior knowledge, Sengupta, Scott, and Kasabov [182] introduced the **GaGamma** scheme. This approach aims to maximize the amount of information encoded while minimizing the spike density. The optimization problem is framed as a mixed-integer optimization problem and can be solved more efficiently by leveraging prior knowledge about the signal being encoded.

The utilization of filter and optimizer encodings for the conversion of ANNs to SNNs is impracticable since the output of the spiking neuron has to be known, which is not readily available.

TC Coding

The final subcategory of temporal coding techniques is referred to as TC coding. This approach is converting an analog stimulus into a train of action potentials by examining changes in signal amplitude [183]. This coding method can be further divided to three distinct algorithms: Threshold-Based Representation (TBR), Step-Forward (SF), and Moving-Window (MW).

TBR involves comparing the absolute change of the input with a threshold and emitting positive or negative spikes in response. The threshold value is determined by combining the mean derivative of the signal with a factor multiplied by the derivative standard deviation.

On the other hand, **SF** only employs the next available signal value and assesses whether the previous value and a threshold have been exceeded. Based on the polarity of the signal difference, appropriate spikes are then transmitted.

Meanwhile, **MW** uses a base that is defined by the mean of the previous signal over a certain time window. Positive or negative spikes are emitted when the current signal value surpasses the base and threshold. For further information and practical implementations, readers can refer to [184].

TC codings are also not useful for the ANN-to-SNN conversion, as it is based on the changes in the input signal. As ANNs are not working on this principle it is not directly feasible to implement in a conversion approach with common datasets.

3.1.4 Conclusion

In this section, we present an analysis of various spiking encoding schemes in the context of converting ANNs to SNNs. Our investigation reveals that only certain encoding schemes are suitable for this purpose. We find that among the rate coding schemes, the count rate coding method represents the state-of-the-art approach for conversion. Thus, it serves as the foundation for our methods presented in chapter 4.

Density rate coding is useful for evaluating stochastic conversion methods but is not a viable approach for conversion. Population rate coding is a feasible alternative for conversion, but its selection should depend on additional benefits as the number of spikes generated by this method can be significant. Our proposed approaches for chapter 5 are based on population codes. As we are using a stochastic approach, we evaluate it over several runs, and, therefore, we implement density rate coding for the evaluation.

Among the temporal codes, only certain codes from the global referenced group can be easily utilized for conversion. TTFS is a suitable candidate, as demonstrated by a previous conversion approach (see Section 6.1). However, this method fails to scale to deep networks. Hence, we present the necessity for a reference oscillation and propose phase coding as a viable alternative, as discussed in chapter 6.

3.2 Properties of Neural Networks

NN architectures represent a class of ML models inspired by structural and functional principles of the human brain. They comprise interconnected nodes, or neurons, arranged in layers that are responsible for processing and transmitting information. These neurons can be organized in diverse ways to produce varying types of neural network architectures, each with distinct advantages and disadvantages.

- *Feedforward (FF) networks* constitute the most common and straightforward neural network architecture. These networks consist of interconnected layers of neurons, with each neuron from a particular layer connected to every neuron in the following layer. Information flows through the network in a forward direction, passing through one or more hidden layers to the output layer, with each layer performing specific transformations on the input.

- *Convolutional neural networks (CNNs)* are a class of architecture created to handle multidimensional data like images, videos, or other visual data [185]. They incorporate convolutional layers, which identify local patterns in the input data. CNNs usually contains several convolutional layers, followed by one or more fully connected layers.

- *Recurrent neural networks (RNNs)* represent an architecture designed to deal with sequential data such as time series data, natural language text, and speech. Unlike feedforward networks, RNNs contain feedback connections that enable data to flow backward through the network. This property allows the network to retain a memory of prior inputs, which is beneficial in predicting future outputs.

- *Transformer networks* were introduced in 2017 and have gained prominence in NLP tasks such as language translation and text summarization [186]. The attention mechanism represents the critical breakthrough in transformer networks, that allows the NN to concentrate on relevant aspects of the input data during prediction. Transformer networks include a sequence of encoder and decoder layers, where the encoder layers process the input data, and the decoder layers produce the output data.

The selection of NN architecture for any specific task is contingent upon the particular data being used and the problem to be solved. FF networks are optimal for simple classification tasks, while CNNs are suitable for visual data processing. RNNs are practical for sequential data analysis, and transformer networks are well-suited for natural language processing tasks.

3.2.1 Tasks and Datasets

NNs are designed to recognize patterns and relationships in data. Neural networks are widely used in a diverse field of applications, such as computer vision, natural language processing, and speech recognition.

Image classification.

Image classification represents a cornerstone problem in both computer vision and machine learning, involving the assignment of a label or category to an input image based on its content. The goal of image classification is to develop algorithms that can automatically identify and distinguish between different objects, scenes, or patterns within an image. This task is typically performed using supervised learning techniques, where a dataset of labeled images is used to train a ML model.

The **MNIST** dataset [187] is a widely used benchmark dataset for image classification. It consists of a collection of 70,000 grayscale images of handwritten digits (0-9) of size 28x28 pixels. The dataset is split into a training set of 60,000 images and a test set of 10,000 images, with labels indicating the corresponding digit for each image.

Another popular dataset is **CIFAR-10** [188]. It consists of a collection of 60,000 color images of size 32x32 pixels, with 10 classes of objects, including animals, vehicles, and everyday objects. The dataset is divided into 50,000 training images and 10,000 test images, each with a corresponding label indicating the class of the object.

A large-scale visual recognition dataset that is commonly used is **ImageNet** [189]. It contains over 14 million images, each with over 1,000 object categories, making it one of the largest and most complex image datasets currently available.

Object Detection.

Another important task is object detection. It involves the identification and localization of objects of interest inside an image or video sequence. The goal of object detection is to develop algorithms that can automatically detect and classify multiple objects of different classes within an image, and provide accurate bounding boxes that localize each object.

Another important neural network task is object detection. Object detection involves identifying the location and type of objects in an image. This task is typically performed using a combination of CNNs and other ML techniques.

The most commonly used dataset for object detection is the **COCO** dataset [190]. It contains more than 330,000 images with more than 2.5 million object instances labeled into 80 different classes, such as people, animals, vehicles, and household items, and includes images with complex scenes and multiple objects. The Common Objects in

Context (COCO) dataset has been used in several challenges and competitions, including the COCO Object Detection Challenge and COCO Captioning Challenge, and has emerged as benchmark for assessing the efficacy of ML algorithms.

NLP.

NLP entails designing and training NNs to comprehend, analyze, and synthesize human language, encompassing written text, spoken dialogue, and other modalities. NLP has many applications, including machine translation, sentiment analysis, chatbots, speech recognition, and text summarization.

A common dataset is the **Internet Movie Database (IMDb) movie review sentiment classification** dataset. It consists of 50,000 movie reviews from the IMDb website, with an equal number of positive and negative reviews. The dataset is commonly used for training and evaluating models for sentiment classification, with the goal of accurately predicting whether a given movie review is positive or negative. The reviews are preprocessed and labeled with binary sentiment values, and the dataset is divided into separate training and testing sets.

3.2.2 Architectures

Convolutional Neural Networks (CNNs) CNN have demonstrated exceptional efficacy in solving a diverse range of computer vision tasks, such as image classification, object detection, and semantic segmentation. CNNs comprise multiple layers that extract increasingly intricate features from input images through convolution and pooling operations.

Currently, the AlexNet, Inception-V3 and ResNets represent some of the most advanced CNN models, with ResNets showing the best results on common datasets [191].

Residual Networks (ResNets).

ResNets are a novel architecture introduced in 2015 to address the challenge of vanishing gradients that often arise in deep neural networks. The vanishing gradient issue arises when the gradient of the loss function concerning the weights of a neural network diminishes significantly as it propagates backwards through the network. This phenomenon makes it difficult to train very deep networks [192].

ResNets employ residual connections to resolve this problem (see Figure 3.7). Specifically, residual connections incorporate the input of a layer with the output of the same layer, thereby allowing information to bypass some of the layers in the network. This results in the formation of a shortcut path for the gradient to propagate through, which can effectively alleviate the vanishing gradient problem.

ResNets are constructed from a sequence of residual blocks, each containing multiple convolutional layers followed by batch normalization and non-linear activation functions.

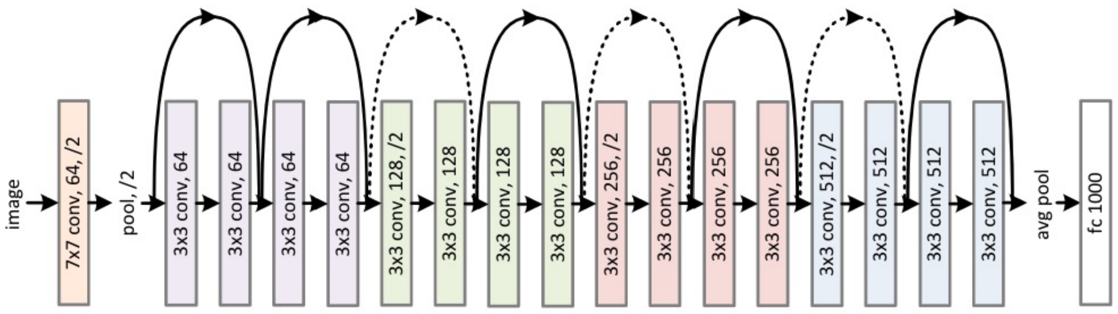


Figure 3.7: Architecture of Resnet18. A residual network with 18 layers. The dotted shortcuts increase dimension (Figure adapted from [192])

The input to each residual block is processed by a series of convolutional layers, producing an output that is subsequently added to the original input by a residual connection. The output of the residual block is fed forward to the subsequent block in the network.

ResNets provide a notable advantage over traditional neural networks in that they can be much deeper while still delivering superior performance. This is primarily due to the residual connections, which facilitate the smooth flow of gradients throughout the network, enabling the training of very deep models. ResNets have found successful applications in various domains, such as image classification, object detection, and speech recognition.

Due to their exceptional performance, these networks have become the dominant architecture for CNNs and have inspired a multitude of similar designs that leverage the same fundamental principles. Since their introduction, many architectures have been developed, each one drawing from the ResNets design and offering further optimizations. Some of the most notable of these designs include ResNeXt [193], DenseNet [194], Inception-v4 [195], MobileNetV3 [196], and EfficientNet [197].

The original ResNet proposal by He et al. included several specific network configurations, namely ResNet18, ResNet34, ResNet50, ResNet101, and ResNet152, with the number indicating the number of layers in the network.

Feature Pyramid Networks (FPNs) FPNs are a type of convolutional neural network architecture designed for object detection in images. The key idea behind FPNs is to address the problem of scale variance in object detection. Objects can appear at different scales in an image, and traditional CNNs can struggle to detect objects at small scales. FPNs address this problem by constructing a pyramid of feature maps at different scales, where each level of the pyramid corresponds to a different level of image resolution.

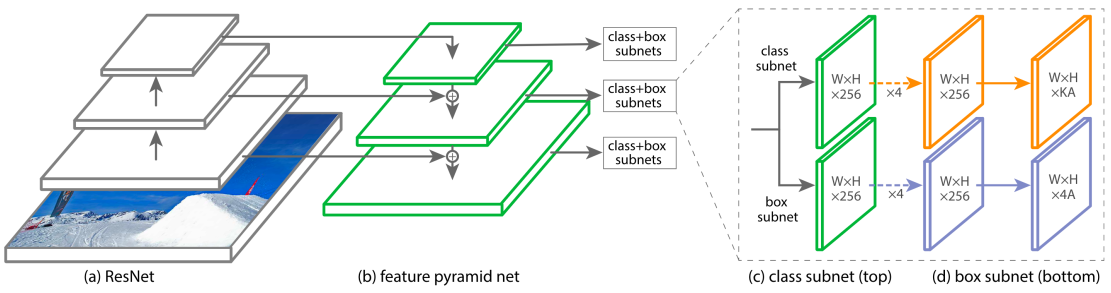


Figure 3.8: Architecture of the RetinaNet Feature Pyramid Network. (a) ResNet architecture, (b) convolutional feature pyramid, (c) anchor boxes, (d) ground truth boxes (Figure from [198])

FPNs consist of two components: a bottom-up pathway and a top-down pathway (see Figure 3.8). The bottom-up pathway is a traditional CNN that extracts features from the input image. The top-down pathway starts with the highest-resolution feature map from the bottom-up pathway and applies a series of upsampling and merging operations to produce a set of feature maps at different scales. The top-down pathway is designed to integrate high-level semantic information from the bottom-up pathway with low-level spatial information from the earlier layers.

The resulting feature maps are used for object detection by applying a set of Region Proposal Networkss (RPNs) at each level of the pyramid. The RPNs generate a set of candidate object bounding boxes, which are then refined and filtered using a set of classification and regression networks.

Some of the best performing FPN architectures consist of RetinaNet [198], Faster R-CNN [199], SSD [200], YOLO v2 [201], YOLO v3 [202], and EfficientDet [203]. RetinaNet showed the best performance for a long time and just recently was overtaken by more modern architectures like YOLOv5l [204].

The key advancement of RetinaNet is the use of a unique loss function called ”focal loss” that handles the issue of class imbalance in object detection. In most object detection datasets, the number of negative examples exceeds the number of positive examples. Focal loss assigns higher weights to misclassified difficult examples and down-weights easy examples, thus ameliorating the effects of class imbalance.

The network predicts a series of anchor boxes at each feature map position, which are used to localize objects. The architecture of RetinaNets is comprised of four sub-networks, each designed to perform a specific task in the overall object detection pipeline. Firstly, a ResNet is utilized to extract meaningful features from the input image and generate feature maps. These maps are then combined by the FPN, resulting in semantically rich

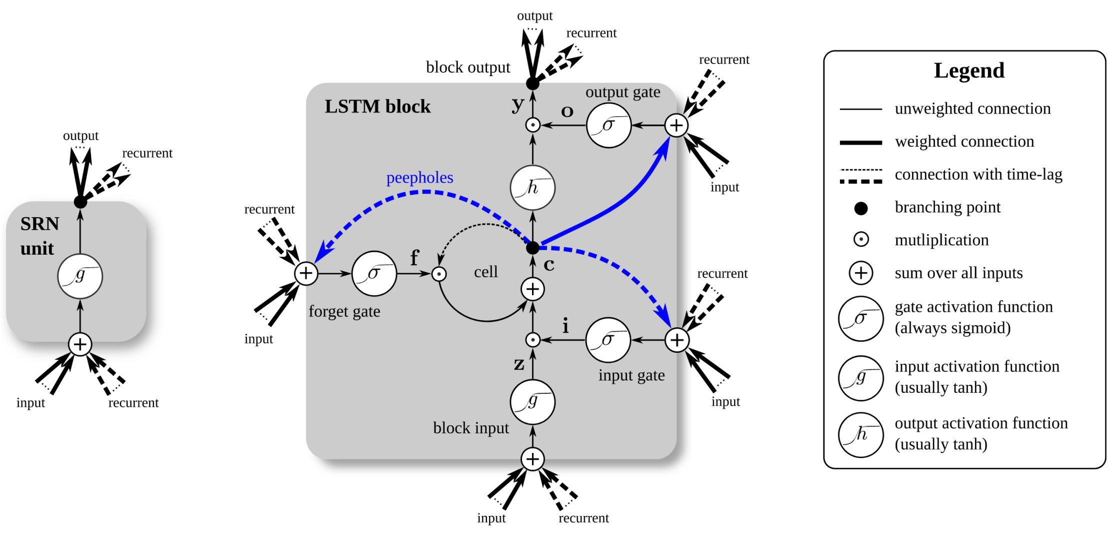


Figure 3.9: Simple recurrent unit and LSTM block. Simple recurrent unit (left) and an LSTM block (right) as used in the hidden layers of an RNN (Figure from [205])

feature maps of low and high resolution. Finally, two sub-networks handle the tasks of classification and bounding box regression.

Recurrent Neural Networks (RNNs) with Memory Cells RNNs are a class of neural networks suited for handling sequential data by employing feedback connections, enabling them to demonstrate dynamic temporal behavior. RNNs possess the unique capability of retaining a memory of previous inputs, which allows them to process sequences of any length.

The network functions on a sequence of input vectors, each vector representing the characteristics of the input at a specific time step (see Figure 3.9 left). The network then sequentially processes these inputs, calculating an output vector and a hidden state vector at each time step. The hidden state vector is determined by combining the current input vector with the previous hidden state, serving as a form of memory. The output vector at each time step is then computed using the hidden state vector and a set of learned parameters.

The use of feedback connections allows RNNs to be trained on sequences of varying lengths and to capture long-term dependencies in the input sequence. Nonetheless, the vanishing gradient problem, where the gradients used to update the network's parameters become infinitesimal, making it difficult to learn long-term dependencies, presents challenges when training RNNs.

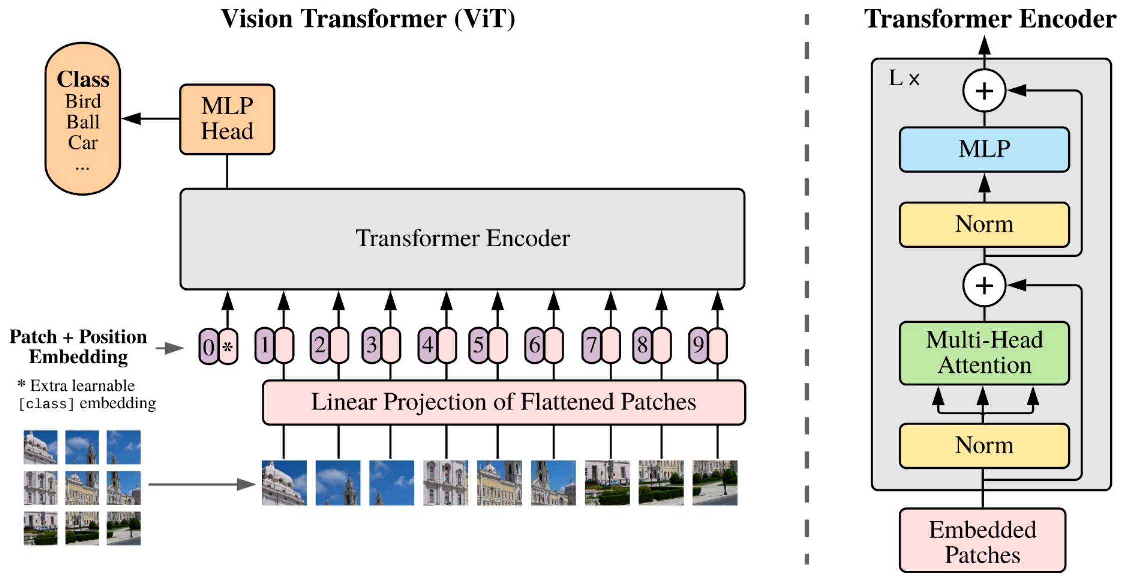


Figure 3.10: Vision transformer architecture. Left: architecture with embedding and encoder. Right: detailed transformer encoder (Figure from [206])

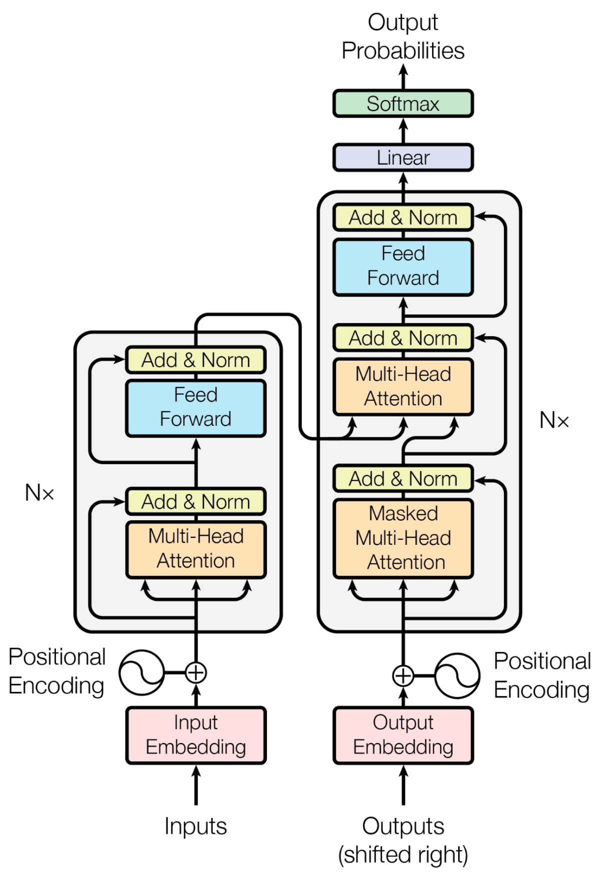
To address the vanishing gradient problem, LSTMs and Gated Recurrent Units (GRUs) have been proposed as memory cells for RNNs. These architectures utilize additional learnable gates to regulate the information flow in the network, enabling selective updates to the hidden state and mitigating the vanishing gradient issue (see Figure 3.9 right).

Transformer Networks Transformer networks are a class of neural network architecture that is designed to handle sequential data, such as text, speech, or music. They were first introduced by Vaswani et al. in 2017 [186] and have subsequently gained considerable popularity in many natural language processing tasks.

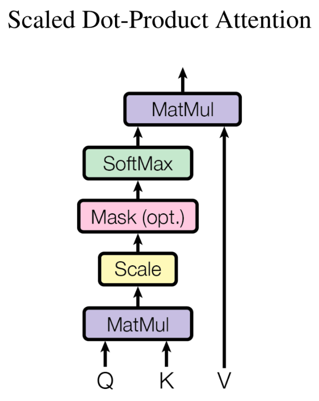
The primary concept behind transformer networks is the self-attention mechanism (see Figure 3.11a), which enables the network to assign importance scores to different parts of the input sequence when making predictions. Unlike conventional RNNs, which process the input sequence element-wise, transformers can process the entire sequence concurrently, resulting in superior speed and efficiency.

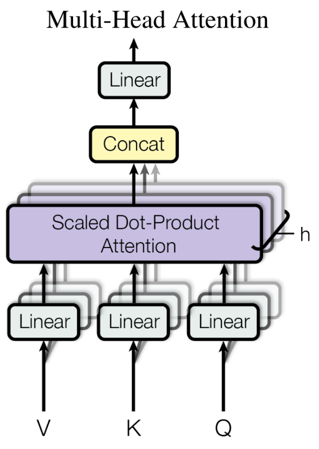
The self-attention mechanism functions by calculating a set of attention weights for each element in the input sequence (Figure 3.11b). These weights are then used to calculate a weighted average of the sequence elements, which serves as input to the subsequent layer of the network. This procedure is repeated multiple times, with each layer refining the representation of the input sequence.

Furthermore, the transformer architecture contains several other important features, including multi-head attention (Figure 3.11c), which enables the network to attend to



(a)





(b) **(c)**

Figure 3.11: Transformer architecture and attention mechanism. (a) Transformer model architecture, (b) Scaled Dot-Product Attention and (c) Multi-Head Attention consisting of several attention layers running in parallel (Figure from [186])

multiple aspects of the input sequence at the same time, and residual connections, which mitigate the problem of vanishing gradients that can arise in deep neural networks.

3.3 Weight Normalization

3.3.1 Basic Conversion

The field of converting ANNs to SNNs has been the subject of extensive research over the years. The study by Rueckauer et al. [144] is of particular significance as it provided a theoretical basis for the underlying mechanisms of different spiking implementations of ANN operators, such as max-pooling [187], batch normalization [207], and an improved version of the softmax mechanism by Nessler et al. [208].

When converting from ANN to SNN, the spike rate $r(t)$ can be derived from the ReLU activation a of the original network [144]:

$$r(t) = ar_{\text{max}} - \frac{V(t) - V(0)}{tV_{\text{thr}}} \tag{3.4}$$

where $V(t)$ represents the membrane potential and V_{thr} represents the firing threshold.

As highlighted by Diehl et al. [146], spiking neurons differ from ReLU units in having an upper bound set by the maximum spike rate r_{max}. Thus, the weights of the ANN must be normalized to fall within the spike frequency range. The normalized weights w' can be calculated by

$$w'_{ij} = \frac{w_{ij}}{\lambda_l} \tag{3.5}$$

with λ being the layer-wise maximum firing rate in the original ANN. However, this approach guarantees that the spike frequency remains within the upper bound and therefore may lead to a large decrease in weights because of single outliers, which would in turn increase the inference time of the network.

To address this issue, Rueckauer et al. proposed *robust normalization* [144], which only utilizes the p^{th} percentile of the absolute activity distribution for each layer. The highest accuracies were obtained with values of p in the range $[99.0, 99.999]$. We refer to the conversion with the usage of robust normalization as *basic conversion.*

3.3.2 Hyperparameter Search

The normalization parameter p regulates the trade-off between inference speed and accuracy, with higher values promoting lower conversion loss at the cost of converging slower, whereas smaller values expedite convergence while introducing accuracy degradation. To

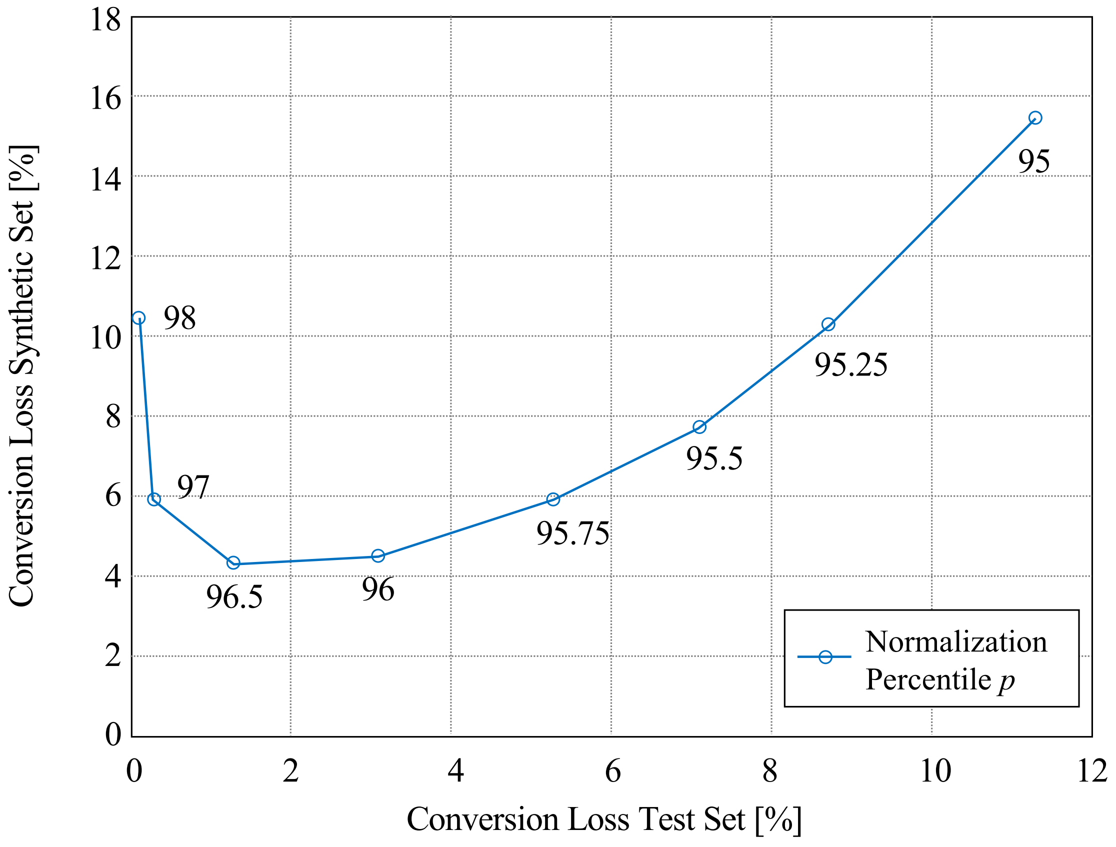


Figure 3.12: Relation between the conversion loss of test set and synthetic set. For an exemplary 3-layer fully connected converted spiking network for different normalization percentiles. The determined value of 96.5 converges twice as fast as the higher values while showing a reasonable loss of only 1.2% (Figure previously published in [39])

address this challenge, we propose a methodology to computationally detect the value of p that balances the speed/accuracy trade-off.

Feeding uniformly distributed noise into a trained ANN, results in random predictions. As the result is very sensitive to small adjustments in the ANN, this can be utilized to evaluate how well the converted SNN represents the original ANN. Converting with inferior normalization parameter choices will result in poor performance on the synthetic data created this way (see Figure 3.12). Small values of p show even larger losses on the synthetic dataset than the test set, while large values result in too slow convergence, such that single action potentials are not averaged out during the individual time steps, which also leads to larger losses. The minimizationg of the conversion loss on the synthetic set determines the value of p for the fastest converging SNN with the highest accuracy at that speed. This approach on networks with different sizes, performances, and random seeds has shown that it can be reliably implemented by using only a small synthetic dataset of roughly 1% of the size of the original test set [39].

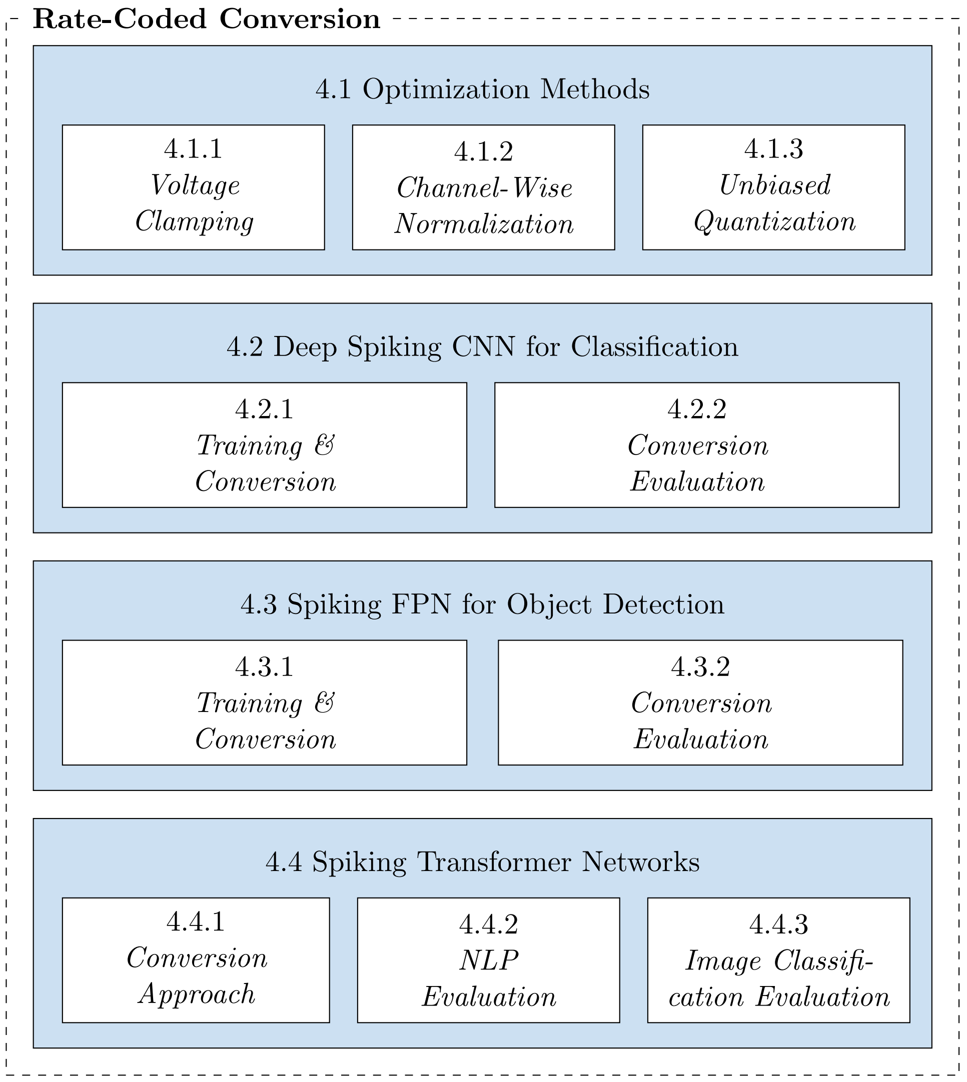
3.3.3 ReLU1 for Prototyping

In order to accurately evaluate the efficacy of the normalization parameter, it is necessary to convert the normalized ANN to a SNN and subsequently conduct a simulation for evaluation. However, this process is often characterized by high computational demands and energy consumption, which may prove impractical. To overcome this challenge, it may be advantageous to determine the resulting accuracy of the converted SNN prior to engaging in these simulations.

We proposed the implementation of the ReLU1 activation function to evaluate accuracy while the ANN is still in its conventional form [43]. After the ANN undergoes conventional training and normalization, the ReLU activation function is replaced with the ReLU1 activation function, which is limited to a maximum value of 1, therefore including an upper limit similar to the maximum firing range of spiking neurons. Consequently, the resulting SNN accuracy can be evaluated within the same environment in which the ANN was trained, thereby reducing computational complexity.

This approach provides a means of evaluating the accuracy of the conversion process without incurring the high costs associated with traditional simulation methods. It is evaluated in Section 4.2.2 in combination with different optimization methods for increasing the inference speed of converted SNNs.

4 Rate-Coded Conversion



Rate-Coded Conversion
4.1 Optimization Methods
4.1.1 *Voltage Clamping*
4.1.2 *Channel-Wise Normalization*
4.1.3 *Unbiased Quantization*
4.2 Deep Spiking CNN for Classification
4.2.1 *Training & Conversion*
4.2.2 *Conversion Evaluation*
4.3 Spiking FPN for Object Detection
4.3.1 *Training & Conversion*
4.3.2 *Conversion Evaluation*
4.4 Spiking Transformer Networks
4.4.1 *Conversion Approach*
4.4.2 *NLP Evaluation*
4.4.3 *Image Classification Evaluation*

Figure 4.1: Chapter structure.

In this chapter, we evaluate the current state-of-the-art rate-coded conversion techniques. Prior research in this domain has primarily concentrated on smaller networks, as simulations involving deep SNNs tend to be associated with significant time and resource expenditures. In order to minimize the inference time, various optimization methods have been proposed and evaluated. These methods aim to increase the accuracy of predictions within a shorter time frame. Two of the optimization methods include voltage clamping by Rueckauer et al. [144] and channel-wise normalization by Kim et al. [142]. In addition to these methods, we propose a new method, called unbiased quantization, which reduces the error in converted networks. This optimization is aimed at reducing the time required for accurate predictions and thus improving the overall efficiency of the spiking network.

Additionally, we undertake an evaluation of the suitability of the ReLU1 activation function to approximate the conversion loss resulting from the normalization hyperparameter, thereby accelerating the development of converted SNNs. We also show an approach for converting transformer networks to their spiking neural network counterparts.[1]

4.1 Optimization Methods

In the inference process of a SNN, the input necessitates multiple time steps as it propagates through the NN. The initial response is primarily determined by a subset of highly active neurons and subsequently influenced by the output of an increasing number of neurons as the simulation progresses. During this transient phase, the accuracy of the output prediction starts with a significant reduction, ultimately converging towards its maximum accuracy over time. The number of needed time steps for the accurate predictions from the output is dependent on the number of layers in the network, possibly needing a very long time for converging.

4.1.1 Related Work: Voltage Clamping

The spiking neuron model involves the integration of inputs $z_i^l(t)$ until the membrane potential $V_i^l(t)$ reaches a threshold V_{thr}, triggering a spike. Following the spike, the membrane potential is reset, with two main reset modes identified in the literature: *reset to zero* [146], which sets the membrane potential to a baseline, commonly zero, and *reset by subtraction* [209], [210], subtracting the threshold value V_{thr} from the membrane potential at the time the threshold is exceeded.

The spike rates r_i^l in relation to the ANN activation a_i^l have to average the membrane potential over the simulation time and reads [144]:

[1]Parts of this chapter have been previously published in [39], [42]–[44].

$$r_i^l(t) = \begin{cases} a_i^l \cdot r_{\max} \cdot \dfrac{V_{\text{thr}}}{V_{\text{thr}} + \epsilon_i^l} - \dfrac{V_i^l(t)}{t \cdot (V_{\text{thr}} + \epsilon_i^l)} & \text{reset to zero} \quad (4.1) \\ a_i^l \cdot r_{\max} \qquad\qquad - \dfrac{V_i^l(t)}{t \cdot V_{\text{thr}}} & \text{reset by subtraction} \quad (4.2) \end{cases}$$

The *reset to zero* case includes an additive approximation error and a further multiplicative error term which will lead to a continuous excess of the threshold by the same constant amount [144]:

$$\epsilon_i^1 = V_i^1(n_i^1) - V_{\text{thr}} = n_i^1 \cdot z_i^1 - V_{\text{thr}} \geq 0 \tag{4.3}$$

The residual charge ϵ_i^1, which is lost during reset, reduces the firing rate and leads to information loss, particularly in deeper layers.

In contrast, *reset by subtraction* benefits the approximation and makes conversion suitable for deeper NNs, where the excess charge ϵ can be used for the subsequent spike generation. Therefore, the error term for ϵ is absent in the spike rate and reduces the error to just the additive error Equation (4.2).

One effective strategy proposed by Rueckauer et al. [144] to mitigate the impact of transients in neuron dynamics is to clamp the membrane potential to zero for an initial N time steps that scales with the depth of the neural layer l:

$$N(l) = d \cdot l \tag{4.4}$$

The parameter d, which represents the temporal gap between releasing the clamp on successive layers, allows for increased convergence of the preceding layer towards the steady-state before the subsequent layer begins integrating its signal.

The authors determined that a clamping delay of $d = 10$ was sufficient for an Inception-V3 network [211], although it did not have a significant impact on the accuracy of VGG16 [212]. This method introduces another hyperparameter d, which has to be further evaluated to determine the optimal value.

4.1.2 Related Work: Channel-Wise Normalization

Kim et al. [142] further improved the layer-wise normalization method (see Section 3.3.1) by introducing *channel-wise normalization.* In contrast to conventional methods, which apply per-layer normalization by scaling all channels within a feature map together, the authors observed that this approach can be suboptimal due to significant variability in channel activations within individual feature maps of a CNN. To address this issue, they proposed to normalize each channel individually, which led to at least a 2.3× increase in inference speed on object detection datasets. The normalization is performed by normalizing the weights and biases of the network using the formula in

$$W_{i,j}^{l} \leftarrow W_{i,j}^{l} \frac{\lambda_i^{l-1}}{\lambda_j^{l}} \quad \text{and} \quad b_j^{l} \leftarrow \frac{b_j^{l}}{\lambda_j^{l}} \tag{4.5}$$

where λ_c^k represents the p^{th} percentile of the c^{th} channel in the k^{th} layer. In their experiments, the authors utilized the 99.9^{th} percentile for normalizing, but this approach also introduces an additional hyperparameter that has to be carefully adjusted for optimal results.

4.1.3 Unbiased Quantization

During the inference phase of the converted network, the spike activity needs to be integrated over time, which can lead to prolonged simulation time until the accuracy of the converted network reaches the baseline ANN. To mitigate this issue, we have demonstrated an approach that significantly reduces the inference time while preserving accuracy. This is achieved by adding half of the threshold voltage to the bias after resetting the membrane voltage of the spiking neuron, resulting in a better representation of the ground truth (see Figure 4.2a) and leading to a faster inference of converted networks (see Figure 4.2b). The resulting rates can be calculated in accordance with [144] using the equation

$$r_i^l(t) = a_i^l r_{\max} - \frac{V_i^l(t)}{t \cdot V_{\text{thr}}} + \frac{V_{\text{thr}}}{2 \cdot n_{\text{steps}}} \tag{4.6}$$

where a_i^l is the ReLU activation, V_i^l is the membrane potential, V_{thr} is the firing threshold, and n_{steps} is the number of time steps in the simulation.

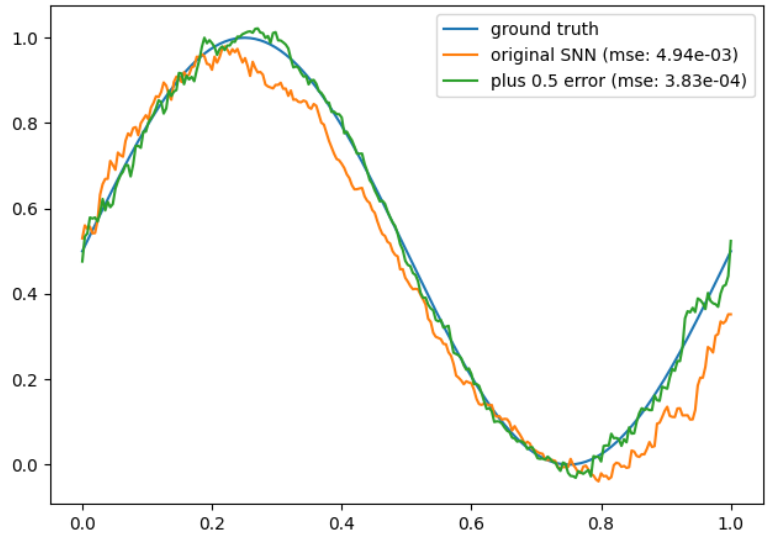
Rate Codes transform the continuous activation of analog neurons into a quantized signal. The quantization in the basic conversion rounds down the activation to the closest quantization step, which results in a biased quantization error of $\frac{1}{2t}$ if the activations are uniformly distributed. This is illustrated in Figure 4.3a.

Our proposed method initializes the membrane potential of the spiking neurons with $\frac{1}{2}V_{thr}$ that unbiases the quantized error. Therefore, the error is averaged to zero as the spike rate consistently over- and underestimates the corresponding activation of the network (see figure Figure 4.3b). A similar approach was presented by Yousefzadeh et al. [213], who proposed a hysteresis quantization of the ReLU activation function prior to conversion. This helps in evening out the error but can lead to an increase in the fluctuation of the output signal.

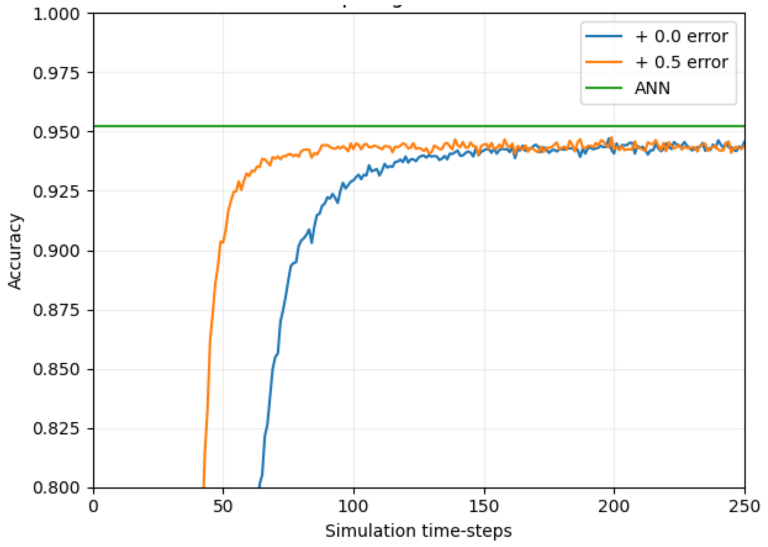
The quantization error in the first hidden layer, $e_i^1(t)$, is calculated using the spike rate $r_i^1(t)$ (Equation (3.4)) and the activation a_i^1 of the corresponding ANN neuron with an initial potential $V_i^1(0) = 0$:

$$\begin{aligned} e_i^1(t) &= |a_i^1 - r_i^1(t)| \\ &= \left| a_i^1 - \left(a_i^1 - \frac{V_i^1(t)}{tV_{\text{thr}}} \right) \right| \\ &= \left| \frac{V_i^1(t)}{tV_{\text{thr}}} \right| \end{aligned} \tag{4.7}$$

With the activation a_i^1 being greater than 0, it is reasonable to assume that $0 < V_i^1(t) < V_{\text{thr}}$. With this setup, the quantization error is limited by $e_i^1(t) < \frac{1}{t}$. However, initializing



(a) Inference of a regression task after 25 time steps. While the SNN converted with centered error matches the ground truth closely, the baseline model has not fully converged.



(b) The full convergence of a converted ResNet34 with centered error is reached one-third faster than the baseline model.

Figure 4.2: Error-centering of spiking neurons. Figures previously published in [42]

the membrane potential of the neurons with $\frac{1}{2}V_{\text{thr}}$ halves this bound:

$$\begin{aligned} e_i^1(t) &= \left| a_i^1 - \left(a_i^1 - \frac{V_i^1(t) - \frac{1}{2}V_{\text{thr}}}{tV_{\text{thr}}} \right) \right| \\ &= \left| \frac{V_i^1 - \frac{1}{2}V_{\text{thr}}}{tV_{\text{thr}}} \right| < \frac{1}{2t} \end{aligned} \tag{4.8}$$

The spike rate $r_i^l(t)$ of a deeper layer $l > 1$ can be calculated using:

$$r_i^l(t) = \sum_{j=1}^{M_{l-1}} W_{i,j}^l r_j^{l-1}(t) + r_{\max} b_i^l - \frac{V_i^l(t) - V_i^l(0)}{tV_{\text{thr}}} \tag{4.9}$$

With $r_{max} = 1$, the quantization error is determined by:

$$\begin{aligned} e_i^l(t) &= \left| a_i^l - r_i^l(t) \right| \\ &= \left| \left(\sum_{j=1}^{M_{l-1}} W_{i,j}^l a_j^{l-1} + b_i^l \right) - \left(\sum_{j=1}^{M_{l-1}} W_{i,j}^l r_j^{l-1}(t) + b_i^l - \frac{V_i^l(t) - V_i^l(0)}{tV_{\text{thr}}} \right) \right| \\ &= \left| \sum_{j=1}^{M_{l-1}} W_{i,j}^l e_j^{l-1}(t) + \frac{V_i^l(t) - V_i^l(0)}{tV_{\text{thr}}} \right| \end{aligned} \tag{4.10}$$

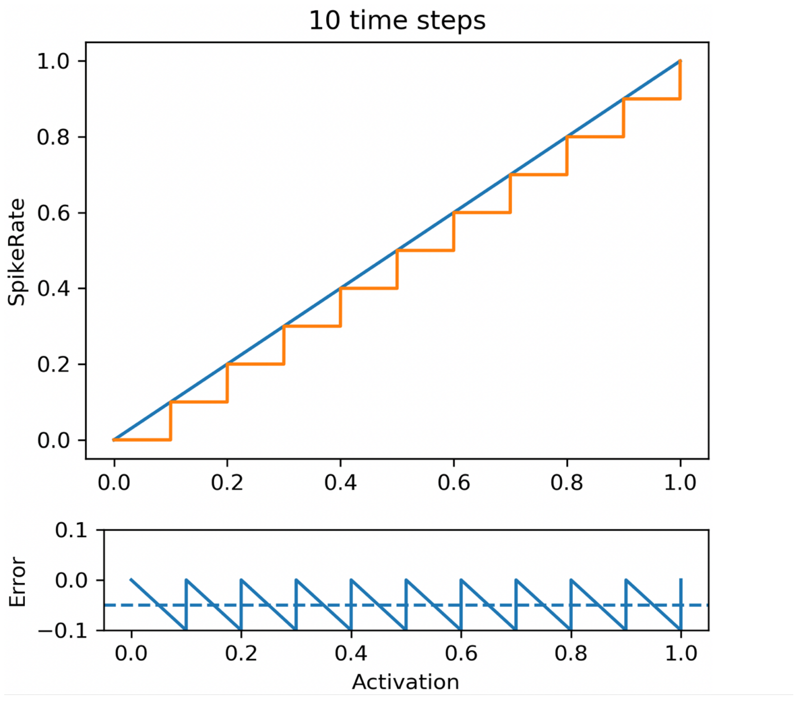
And the quantization error for $V_i^l(0) = 0$ is limited by:

$$e_i^l(t) = \left| \sum_{j=1}^{M_{l-1}} W_{i,j}^l e_j^{l-1}(t) + \frac{V_i^l(t)}{tV_{\text{thr}}} \right| < \left| \sum_{j=1}^{M_{l-1}} W_{i,j}^l e_j^{l-1}(t) + \frac{1}{t} \right| \tag{4.11}$$

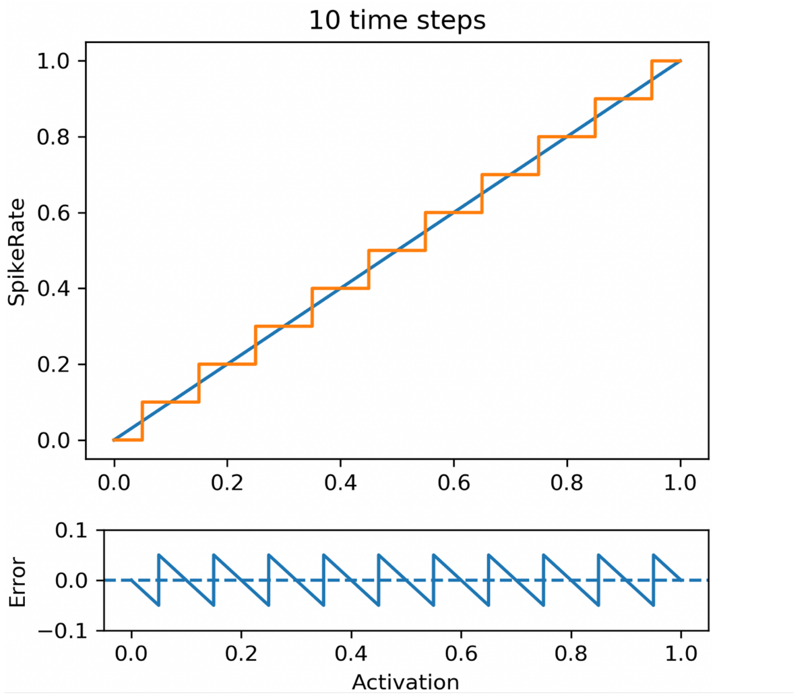
Initializing the membrane potential of the neurons with $\frac{1}{2}V_{\text{thr}}$ halves the maximal error due to quantization:

$$e_i^l(t) = \left| \sum_{j=1}^{M_{l-1}} W_{i,j}^l e_j^{l-1}(t) + \frac{V_i^l(t) - \frac{1}{2}V_{\text{thr}}}{tV_{\text{thr}}} \right| < \left| \sum_{j=1}^{M_{l-1}} W_{i,j}^l e_j^{l-1}(t) + \frac{1}{2t} \right| \tag{4.12}$$

In conclusion, initializing SNNs using our proposed method leads in a prediction with higher accuracy with shorter simulations by centering the error. It does not introduce any additional hyperparameters, making it an efficient and effective solution to the problem of biased quantization error in ANNs and SNNs.



(a)



(b)

Figure 4.3: Biased and unbiased quantization. Overview of the biased (a) and the unbiased (b) quantization method in spiking neural networks (Figure previously published in [43])

4.2 Deep Spiking CNN for Classification

For a comprehensive evaluation of the three optimization methods, as well as their combined effect, we conduct experiments on four different ResNets architectures, ResNet18 to ResNet101, using the CIFAR-10 dataset [188]. The CIFAR-10 dataset is a popular image classification benchmark, comprising 60,000 images of size $32 \times 32 \times 3$ pixels, classified into 10 different categories. The dataset is divided into 50,000 training images and 10,000 test images. As there is no validation set provided in CIFAR-10, we utilize the last 5,000 images from the training set for validation purposes, reducing the training set to only its first 45,000 images.

In order to obtain an individual overview of the optimization methods, we perform a separate evaluation of unbiased normalization, voltage clamping, and channel-wise normalization on the validation set, to determine the best-performing hyperparameters. These hyperparameters are then jointly evaluated on the test set to determine their overall performance.

For the purpose of simplicity, we utilize potential encoding for the output layer [144], where the softmax function is calculated based on the membrane potential of the last layer. All SNNs are simulated for 1,000 time steps, providing ample opportunity to evaluate their performance.

We evaluate the performance of the optimization methods of unbiased quantization, channel-wise normalization and voltage clamping using the hyperparameter value $d = [1, 2, 3]$. This initial step adheres to the normalization method proposed by Kim et al. [142] which suggests using the 99.9^{th} percentile. To determine the optimal normalization parameter in this application, we examine various values beyond just the 99.9^{th} percentile. This is motivated by the findings of Rueckauer et al. [144] which showed that percentiles in the range $[99, 99.999]$ performed best for layer-wise normalization. In our evaluation, we consider percentiles $[99, 99.9, 99.99, 99.999, 100]$ for channel-wise normalization and assess their performance in combination with the other two methods.

After conducting experiments and evaluating the results on the validation set of CIFAR-10, we then assess the best-determined configuration on the test dataset to obtain an objective result.

4.2.1 Training and Conversion

The training of the ResNets required slight adaptations to suit the small size of only 32×32 pixels. This was accomplished by modifying the first convolution layer to a 3×3 convolution with a stride of 1 and removing the max pooling layer. Additionally, the number of channels in all convolutions was halved. The networks were trained for a total of $50 + 100n$ epochs, where $n = [1, 2, 3, 4]$ for the ResNet18, ResNet34, ResNet50, and ResNet101, respectively. The training process involved the use of Stochastic Gradient

Descent (SGD) as optimizer with a batch size of 128, an initial learning rate of 0.1, momentum of 0.9 and a weight decay of 0.0005. The initial learning rate was decayed by a factor of 10 at $50n$ and $25 + 75n$ epochs, respectively. The images were standardized by normalizing each channel to have a zero mean and unit variance. Data augmentation was performed following the techniques described by He et al. [192] and Lee et al. [214]. This involved zero-padding the images with 4 pixels on each side and then randomly cropping 32x32 pixels, as well as horizontally flipping each image with a probability of 50%.

The conversion of the ResNets involves a specific normalization process of the residual blocks, as the converted activations of the ReLU after the element-wise addition of the residual block should not exceed the maximum value, r_{max}. In order to achieve this, the residual blocks of the ResNets are normalized by the method outlined by Hu et al. [141]. This normalization process takes into consideration both the main paths and the shortcuts of the residual blocks and normalizes them using their activations.

4.2.2 Conversion Evaluation

Figure 4.4 illustrates the comparison of our proposed unbiased normalization method, the channel-wise normalization (using the 99.9^{th} percentile), and voltage clamping (with $d = [1, 2, 3]$) to a baseline SNN. It has been observed that these three optimization techniques exhibit a relatively similar level of accuracy and converge faster in comparison to the baseline spiking ResNet. This observation can be attributed to the simplicity of the CIFAR-10 dataset and the large size of the neural networks. As a result, lossless conversion was achieved for ResNet18 and ResNet34. On the other hand, the baseline spiking ResNet50 and ResNet101 failed convergence within the simulated 1,000 time steps, although their classification accuracy was still rising. It can be deduced that they would converge to the same accuracy as the original ANN if the simulation time was prolonged.

Results of ReLU1

To assess the usage of ReLU1 for faster prototyping of converted SNNs, we also compute the ReLU1 performance for the normalized ANNs. This involves replacing all ReLU activation functions with the ReLU1 function, which clips all activations between 0 and 1. This gives us valuable insights into the extent to which the accuracy drop can be attributed to the clipping versus the quantization errors of the SNN. Furthermore, the ReLU1 performance can be considered a theoretical, upper-bound estimate of the potential performance of the SNN.

The accuracy results for different percentiles of converted SNNs and the corresponding ReLU1 activation, are depicted in Table 4.1. The ReLU1 accuracy was found to be very close to the SNN's performance, making it a useful predictor for evaluating different

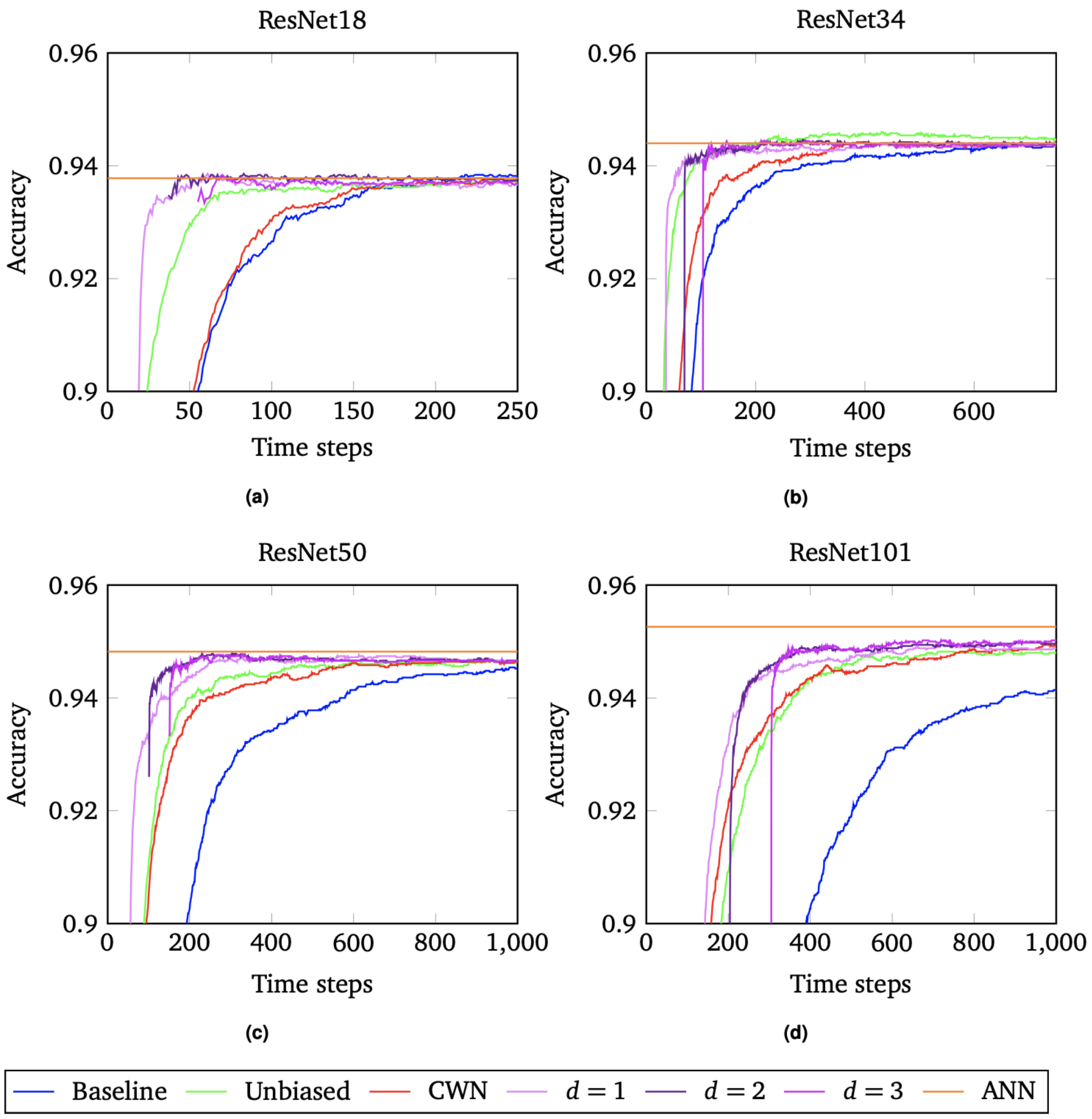


Figure 4.4: Inference of the different optimization methods for different-sized ResNets. Inference of the different optimization methods for ResNets of different sizes on the CIFAR-10 dataset (Figure previously published in [43])

Table 4.1: Accuracy for different optimization methods. Accuracy for the channel-wise normalization combined with unbiased quantization and voltage clamping for different percentiles as well as their corresponding ReLU1 activation. For the spiking networks, the average accuracy of the last 100 time steps is given (Table previously published in [43])

%ile	**ResNet18**		**ResNet34**		**ResNet50**		**ResNet101**	
	SNN	ReLU1	SNN	ReLU1	SNN	ReLU1	SNN	ReLU1
99^{th}	92.24%	92.24%	92.42%	92.42%	92.84%	92.86%	92.66%	92.66%
99.9^{th}	93.71%	93.70%	94.26%	94.24%	94.71%	94.70%	95.01%	95.04%
99.99^{th}	93.85%	93.82%	94.40%	94.38%	94.78%	94.80%	95.27%	95.24%
99.999^{th}	93.80%	93.76%	94.40%	94.44%	94.74%	94.80%	95.28%	95.24%
100^{th}	93.87%	93.78%	94.35%	94.44%	94.78%	94.82%	95.25%	95.26%
ANN	93.78%		94.40%		94.82%		95.26%	

percentiles prior to the conversion. This additionally suggests that the main source of variability in accuracy lies in the activation clipping rather than quantization errors in the SNN.

Results of the individual optimization methods.

The results of the individual optimization methods were analyzed in order to determine the most effective approach for improving inference time in SNNs. The voltage clamping method was found to be the most effective, demonstrating the largest improvement in inference time for all of the ResNet networks tested. Although the number of time steps before the first output is produced corresponds to the number of layers in the network multiplied by the hyperparameter d, the different values of d did not result in a large difference in performance. With $d = 1$ the network needs slightly more time steps for convergences, whereas $d = 3$ needs three times as long to generate the first spiking output, but subsequently converges at an accelerated pace. We decided to use $d = 1$ for the following experiments, as the difference in accuracy between different values of d could be neglected and higher values would result in excessively long delays in output generation.

The second most effective optimization method was unbiased quantization, which alone was able to more than half the necessary inference time compared with the baseline SNN. While this method converges slower than the voltage clamping approach, it does not introduce an additional hyperparameter, making it a cost-effective solution.

The channel-wise normalization method was found to be the least effective of the tested approaches, particularly for smaller networks. Despite converging to similar accuracy levels, this approach achieved only a slight speedup compared to the baseline ResNet18. For deeper networks, this more fine-grained normalization method produced a larger difference and was more comparable in performance to the other methods. For the ResNet101, the channel-wise normalization method resulted in a slightly faster inference time than the unbiased quantization.

Given the benefits demonstrated by all optimization methods, further investigation was conducted into the hyperparameter p for channel-wise normalization in conjunction with the other two approaches. The final accuracy results for the spiking neural networks, calculated as the average accuracy of their last 100 time steps can be found in Table 4.1.

The 99^{th} percentile performed the worst for both the SNN and the ANN, resulting in lower accuracy compared to the other methods by 1.5 to 2.5 percentage points. The 99.9^{th} percentile converged to a lower accuracy, which was most noticeable for the ResNet101. The 99.99^{th}, 99.999^{th}, and 100^{th} percentiles all converged for all ResNets to the same accuracy as the ANN, with only minor variations in difference. However, the 100^{th} percentile required roughly double the time for full convergence, highlighting the importance of considering the hyperparameter tuning if inference speed is a relevant concern.

Results of the combined optimization methods.

To evaluate the combined effect of the three optimization methods, we have conducted an extensive experiment on the CIFAR-10 test set. The optimization methods used in this experiment are unbiased quantization, channel-wise normalization (using the 99.99^{th} percentile), and voltage clamping with $d = 1$. This configuration is referred to as the *extended conversion* and will be compared to the basic conversion.

Our results indicate that the extended conversion method substantially increases performance in comparison to the basic conversion method. Specifically, we observe a roughly ten-fold gain in the speed of ResNet18 when compared to the basic conversion (see Figure 4.5). The extended conversion crosses the 94% accuracy bar with only 56 time steps, whereas the basic conversion requires approximately 600 steps to reach the same bar. This difference is even more pronounced when considering larger networks such as ResNet101. In the 1,000 time steps of the simulation, the basic spiking ResNet101 does not converge, whereas the extended conversion reaches the same accuracy with less than 150 time steps.

For both ResNet34 and ResNet50, we observe a significant drop in accuracy from the basic conversion to the original ANN. However, the extended conversion method results in networks that reach the same accuracy as the original ANN. This suggests that the conversion process for simple datasets such as CIFAR-10 is lossless. The final accuracy results, listed in Table 4.2, demonstrate the performance after the simulation was completed. A comparison to other conversion work for image classification can be found in Table 4.3

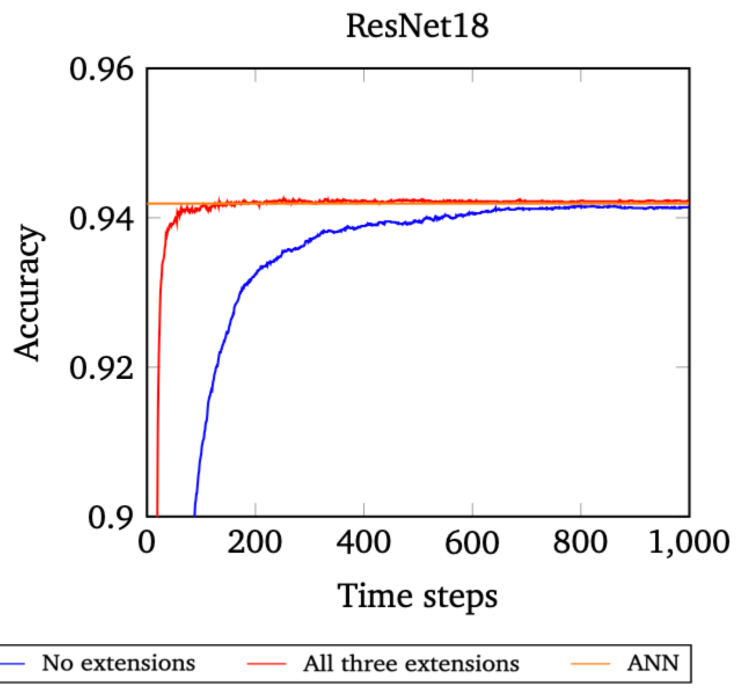


Figure 4.5: Accuracy of the three optimization methods on the CIFAR-10 dataset. Combined voltage clamping, channel-wise normalization and unbiased quantization for a ResNet18 (Figure previously published in [43])

Table 4.2: Resulting accuracies for the original ANN. the basic and the extended conversion. For the SNNs the average accuracy of the last 100 time steps is given (Table previously published in [43])

	ResNet18	**ResNet34**	**ResNet50**	**ResNet101**
ANN	94.19%	94.67%	95.11%	94.92%
Basic Conversion	94.13%	94.41%	94.59%	93.45%
Extended Conversion	94.22%	94.61%	95.12%	94.95%

4.3 Spiking FPN for Object Detection

To evaluate the scalability of our approach, we extend it to a large-scale application by converting a RetinaNet with a ResNet18 backbone to a spiking network and benchmark its performance on the challenging Microsoft COCO dataset [190]. COCO is a large-scale benchmark for object detection, segmentation, and captioning, containing around 118,000 training, 5,000 validation, and 41,000 test images, in total roughly 164,000 images. The objects in the images are classified into 80 different categories, with varying resolutions that range from 72×51 to 640×640 pixels, and aspect ratios of up to $6 : 1$. Due to the high computational complexity of the network, the performance of the SNN is only assessed on the 5,000 validation images. In this experiment, all three optimization methods are applied combined.

To pick a reasonable percentile for the channel-wise normalization, a process outlined in Section 4.2.2, the Mean Average Precision (mAP) of the ReLU1 must be computed with different values. This will allow for the selection of the most suitable percentile before the conversion takes place.

4.3.1 Training of RetinaNets

The training process for the conventional RetinaNet differs from the original work by Lin et al. [198]. The Adam optimizer, as described by Kingma and Ba [215], is used instead of SGD with a batch size of 14, a learning rate of initially 0.0001, and the weight decay set to 0.00001. The hyperparameters are set to $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$, respectively.

The training of the RetinaNet is done for a total of 90,000 iterations, with the learning rate being divided by 10 after 60,000 and 80,000 iterations. Unlike the original work, we treat the weight introduced by the focal loss as a constant in the derivation step.

Training involves random resizing of the image's shorter side to a value between 640 and 896 pixels, and all images within a batch of 14 must have the same resolution. Therefore, zero-padding is applied to the images to ensure that they all have the same size. The images are also augmented by randomly flipping them horizontally, with a 50% chance.

Converting the RetinaNet requires additional considerations for the FPN. The FPN weights require adjustment to compensate for the normalization applied to the backbone network. Specifically, the four convolutional layers that take feature maps from C3, C4, and C5 as input need to undo this normalization. After normalizing the activation of the 3×3 convolution following C5, it is passed through ReLU, then immediately denormalized by the subsequent 3×3 convolution. With P_6 normalized, all pyramid feature maps are now consistent and can proceed to the regression and classification subnetworks.

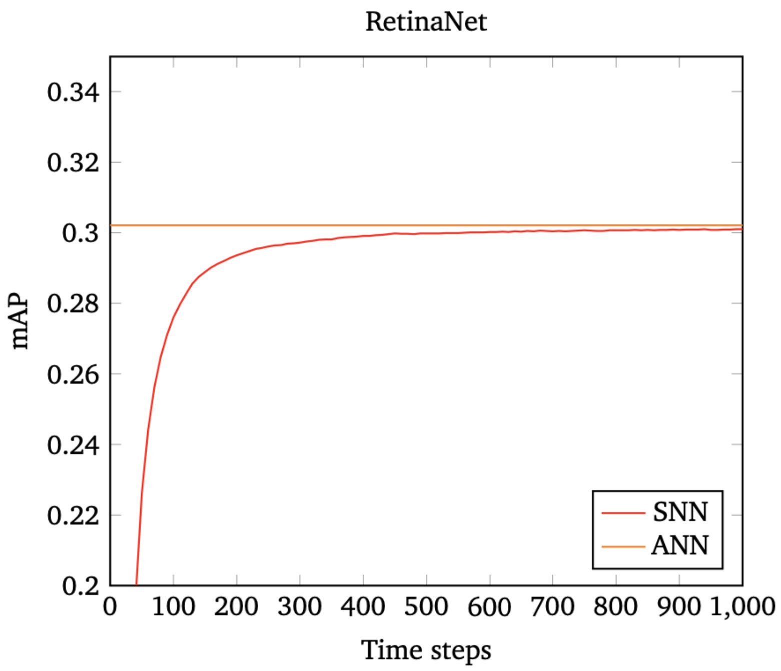


Figure 4.6: mAP of the three optimization methods. mAP of the three optimization methods on the COCO dataset for a converted RetinaNet with a ResNet18 as the backbone, as well as the corresponding ANN's mAP (Figure previously published in [43])

Given that both convolution and nearest neighbor upscaling are linear operations, it follows that any combination of these two operations will yield another linear transformation. As a result, all computations within the FPN, except for the ReLU, can be condensed into their subsequent weights, which greatly simplifies the overall process.

4.3.2 Evaluation of the Results

In addition to evaluating the extended conversion method on simple datasets, we have also conducted a large-scale evaluation using a RetinaNet. To find the best-suited hyperparameter for the channel-wise normalization, we computed the ReLU1 mAP for different percentiles. Our results indicate that the 99.999^{th} percentile performs the best with an mAP of 30.16, followed closely by the 99.99^{th} percentile with an mAP of 29.68. The 99.9^{th} percentile performed the worst with an mAP of 26.75. Our findings indicate that there is no one-size-fits-all solution, and the optimal percentile value varies across applications. However, we show that using ReLU1 as a performance metric can serve as a useful predictor to guide the selection of an appropriate value tailored to each specific use case.

Within our experiments using 1,000 simulation time steps, we observed that the original ANN achieved an mAP of 30.21, while the converted SNN reached 30.09 (see Figure 4.6). The difference between these two values amounts to a mere 0.12 or 0.40%, highlighting the effectiveness and efficiency of the extended conversion method.

Other methods for spiking object detection have also been proposed. Hu et al. [141] introduced conversion methods for ResNets, while Xiao et al. [216] and Kerapdiseh et al. [124] presented biologically plausible learning algorithms for object recognition. Wu et al. [217] introduced Progressive Tandem Learning, a layer-wise learning framework with an adaptive training scheduler for rapid pattern recognition. A comparison of our approach to other conversion work for object detection can be found in Table 4.3

Table 4.3: Overview of Conversion Losses. Overview of Conversion Losses of Different Network Architectures and Datasets (Table previously published in [43])

Author		Network	Dataset	ANN	SNN	Loss
Diehl et al. (2015)	[146]	3-layer CNN	MNIST	99.14%	99.10%	<0.1%
Stromatias et al. (2017)	[127]	2-layer CNN	MNIST	98.30%	98.32%	<0.1%
Hu et al. (2018)	[141]	ResNet8	MNIST	99.59%	99.59%	<0.1%
Yousefzadeh et al. (2019)	[213]	4-layer CNN	MNIST	99.21%	99.19%	<0.1%
Cao et al. (2014)	[143]	3-layer CNN	CIFAR10	79.12 %	77.43%	2.14%
Hunsberger, Eliasmith (2015)	[218]	5-layer CNN	CIFAR10	85.97%	83.54%	2.43%
Rueckauer et al. (2017)	[144]	9-layer CNN	CIFAR10	91.91%	90.85%	1.06%
Hu et al. (2018)	[141]	ResNet44	CIFAR10	92.85%	92.37%	0.52%
Sengupta et al. (2019)	[147]	ResNet34	CIFAR10	89.10%	87.46%	1.84%
Han et al. (2019)	[219]	VGG16	CIFAR10	93.63%	93.63%	<0.1%
Our work (2021)	**[43]**	**ResNet18**	**CIFAR10**	**94.19%**	**94.22%**	**<0.1%**
Our work (2021)	**[43]**	**ResNet34**	**CIFAR10**	**94.67%**	**94.61%**	**<0.1%**
Our work (2021)	**[43]**	**ResNet50**	**CIFAR10**	**95.11%**	**95.12%**	**<0.1%**
Our work (2021)	**[43]**	**ResNet101**	**CIFAR10**	**94.92%**	**94.95%**	**<0.1%**
Hu et al. (2018)	[141]	ResNet44	CIFAR100	70.18%	68.56%	1.62%
Han et al. (2019)	[219]	VGG16	CIFAR100	71.22%	70.93%	0.29%
Rueckauer et al. (2017)	[144]	VGG16	ImageNet	63.9% (84.9%)	49.6% (81.6%)	22.4% (3.8%)
Rueckauer et al. (2017)	[144]	Inception-V3	ImageNet	76.1% (93.0%)	74.6% (92.0%)	2.0% (1.0%)
Sengupta et al. (2019)	[147]	ResNet34	ImageNet	80.69% (89.69%)	65.47% (86.33%)	18.18% (3.75%)
Kim et al. (2019)	[142]	YOLO	PascalVOC	53.01%	51.83%	2.23%
Kim et al. (2019)	[142]	YOLO	COCO	26.24 mAP	25.66 mAP	2.21%
Our work (2021)	**[43]**	**RetinaNet**	**COCO**	**30.21 mAP**	**30.09 mAP**	**0.40%**

4.4 Spiking Transformer Networks

For the development of a conversion method, we use networks similar to the transformer networks used in the original work for NLP [186] (see Figure 4.7) and image classification [206] (see Figure 4.9). The architecture comprises a preprocessing stage that depends on the data type of the input, followed by the implementation of a transformer encoder, which forms the backbone of the network, and various subsequent layers with a classification output layer.

The transformer encoder is constructed using multi-head self-attention, comprising of several scaled-dot attention modules to perform the necessary computations. The computation performed by the scaled-dot attention modules involves a series of stacked layers of matrix multiplication, scaling, softmax, and a final matrix multiplication layer. After the multi-head self-attention layer, a subsequent dense layer is implemented, followed by the application of a ReLU activation function and yet another dense layer.

In our approach, we use an average pooling layer instead of the commonly used max pooling operation, as the latter is difficult to realize in SNNs. For classification of the output, different spiking operators have been proposed [144], but for simplicity, we have included an additional layer that accumulates all spikes generated by the last dense layer. This enables the use of a common softmax layer for evaluating the accuracy, without the need for a more complex implementation.

4.4.1 Conversion Approach

As for the conversion of CNN, conversion is done by replacing ReLU activation functions of the ANN with IF neurons. The rate-coded spiking neurons offer a low computational complexity by integrating inputs until a threshold is reached, at which point the neuron triggers an action potential transmission to subsequent neurons before resetting. This spike rate $r(t)$ can be calculated using the ReLU activation a of the original ANN through the equation:

$$r(t) = ar_{max} - \frac{V(t) - V(0)}{tV_{\text{thr}}} \tag{4.13}$$

where $V(t)$ is the membrane potential and V_{thr} is the firing threshold.

However, unlike the ReLU activation function, which have no upper bound, spiking neurons are characterized by a maximum firing rate that defines an upper limit r_{max} [146]. This means that the weights of the original ANN must be normalized to avoid extended inference time or decreased accuracy. To achieve this normalization, a subset of the training set is taken and the activations of the layers to be converted are computed.

Extreme outliers are discarded through the implementation of the *robust normalization* algorithm (Section 3.3 [144]), which only normalizes the p^{th} percentile of the activation.

The authors suggest p values in the range of [99.0, 99.999]. However, it was found that a value of $p = 99.0\%$ performed the best.

In addition, the *reset-by-subtraction* method [220] was used instead of resetting to 0, as it has shown better efficiency in the conversion process.

In order to evaluate the efficacy and accuracy of the proposed conversion method, we have selected two distinct transformer network architectures for training, one is closely related to the original NLP sentiment classification model as presented in [186], while the other is based on the vision transformer design proposed for image classification [206].

The conversion process involves the normalization of weights and the replacement of traditional activation functions with IF neurons. Once the networks have been converted, they are then subjected to an evaluation in a simulation process that involves the propagation of spikes over a period of 50 time steps.

4.4.2 NLP Evaluation

For the first experiment, we undertake the training of a transformer network specifically designed for NLP tasks, utilizing the IMDb movie review sentiment classification dataset [221]. This dataset comprises 25,000 movie reviews, each represented by a sequence of word indices and labeled as positive or negative sentiment. The dataset is split into two equal parts, with one half being utilized for training and the other half reserved for testing purposes.

The data is prepared by limiting the length of each review to 200 words and padding any shorter reviews with zeros. Furthermore, a vocabulary list comprised of the 20,000 most frequently used words is utilized for the encoding of each review. The labels for each review are also transformed into categorical vectors, with a positive or negative state.

The network inputs are encoded movie reviews, presented as sequential lists of word indices. To properly handle this input data, embedding layers are employed for both the word sequences and for encoding the positions of each word in the sequence. The resulting token embeddings and positional embeddings are then summed and fed into the transformer encoder. Subsequently, the data passes through an average pooling layer and then two densely connected layers, each equipped with a ReLU activation function. Finally, a softmax layer is utilized to perform the classification.

The training procedure for both the NLP and vision transformers is carried out identically. The training process is performed using the Adam optimization algorithm with a batch size of 64, over 2 epochs. The robust normalization strategy is applied to the resulting weights. This process scales the weights to fit the maximum spike rate of the integrate and fire neurons and includes setting the percentile value to $p = 99.0$.

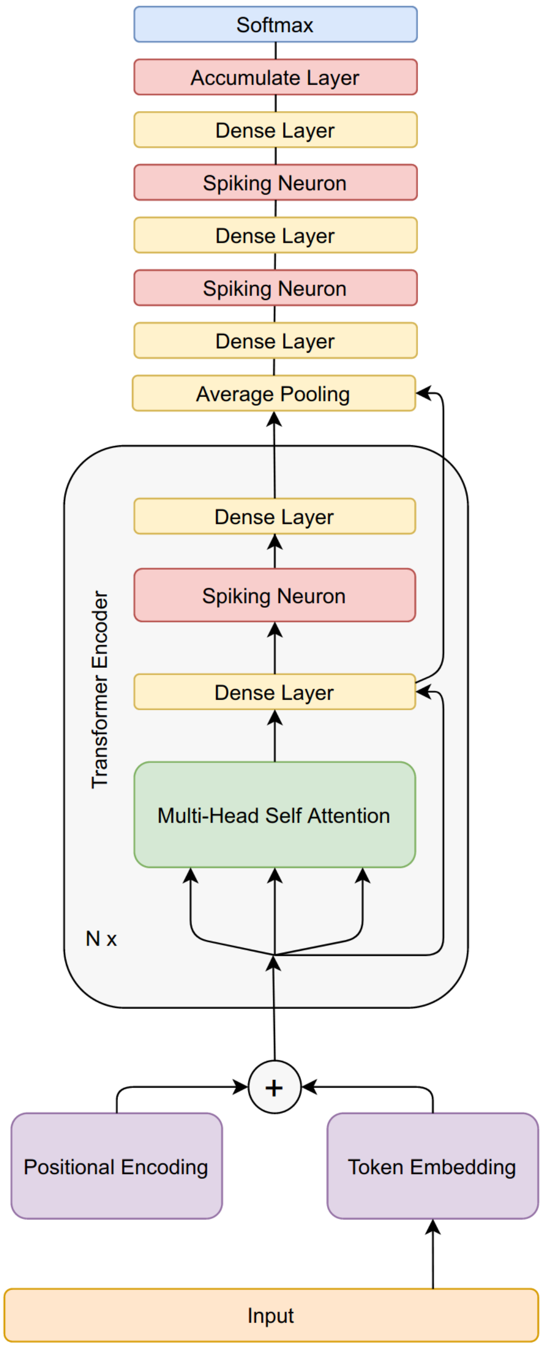


Figure 4.7: Architecture of the converted spiking NLP transformer network. The ReLU activation of the trained ANN is replaced by spiking neuron Models. In contrast to the original work for NLP [186], we use average pooling instead of max pooling after the transformer encoder module (Figure previously published in [44])

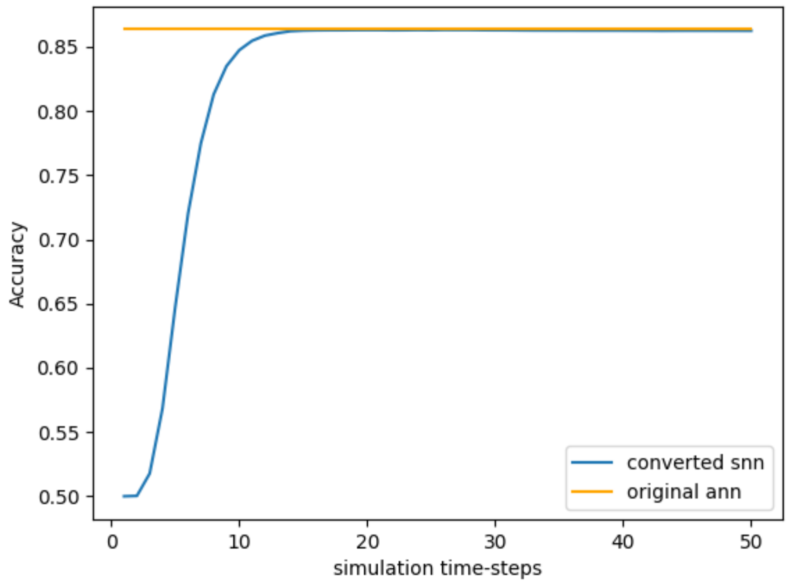


Figure 4.8: Results of NLP transformer conversion. Averaged accuracy of the converted spiking NLP transformer network over 50 time steps on the IMDB sentiment classification dataset (Figure previously published in [44])

The spiking transformer network is created by replacing traditional ReLU activations with a spiking neuron model. Its performance is subsequently assessed through a 50-step simulation, where the test set accuracy is calculated and reported. To account for any variability in the results due to random seed selection, each experiment was repeated 50 times, with the results averaged to produce a more accurate representation of the network's performance.

The original NLP transformer showed an average accuracy of 86.36% on the IMDb movie review sentiment classification dataset test set. The conversion of this ANN to a spiking network resulted in a slightly lower accuracy of 86.25% after the simulation of 50 time steps. The accuracy evened out after only 13 time steps, which is illustrated in Figure 4.8. The comparison between the original ANN and the converted spiking network reveals a conversion loss of only 0.11%.

4.4.3 Image Classification Evaluation

For the second experiment, we aim to evaluate the efficacy of the proposed vision transformer network for image classification using the Modified National Institute of Standards and Technology Database (MNIST) dataset [187]. The MNIST dataset consists of 60,000 handwritten digit images ranging from 0 to 9, each with a corresponding label.

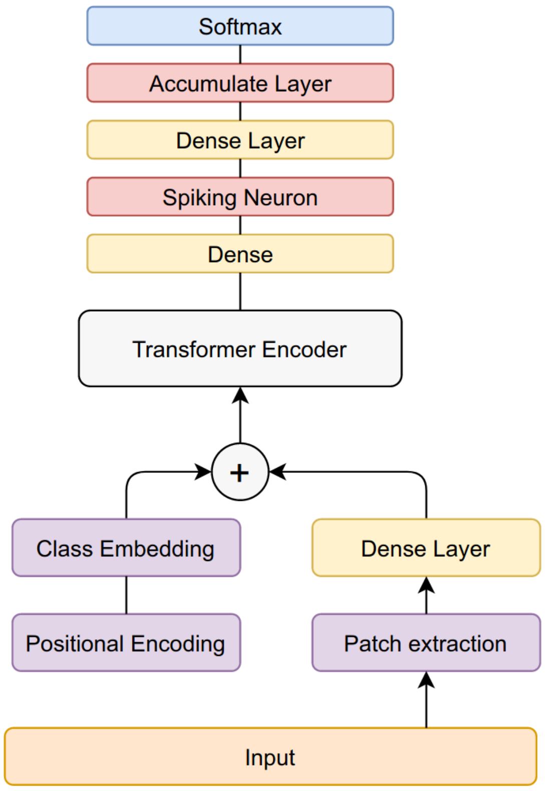


Figure 4.9: Architecture of the converted spiking vision transformer network. The ReLU activation of the trained ANN is replaced by spiking neuron Models. In contrast to the original work for NLP [186], we use average pooling instead of max pooling after the transformer encoder module (Figure previously published in [44])

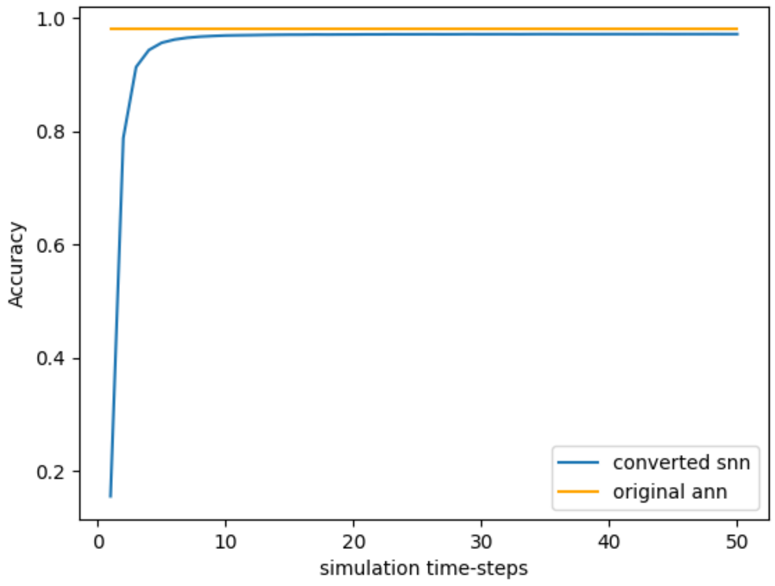


Figure 4.10: Results of vision transformer conversion. Averaged accuracy of the converted spiking vision transformer network over 50 time steps on the MNIST image classification dataset (Figure previously published in [44])

All images in the dataset have a resolution of 28×28 pixels. The test set contains an additional 10,000 images.

To incorporate the images into the transformer network, we segment each image into 49 4×4-pixel patches, arranged in a 7×7 grid. Each patch is then flattened and transformed into a linear embedding of all patches. Positional encodings and class embeddings are appended to these projections to provide contextual augmentations. The resulting feature representation is then processed by the transformer encoder, which uses the same architecture as described in the previous section. Finally, the output is passed through a dense layer with ReLU activation, followed by another dense layer with softmax activation for classification purposes.

The training procedure for the vision transformers is carried out identically as for the NLP transformer. The same for the conversion, where the ReLU activation functions are replaced with IF neurons. Also, the performance of the spiking Transformer Neural Network (TransformerNet) is then assessed through a simulation over 50 time steps and each experiment is repeated 50 times for averaging the results.

The vision transformer of the latter experiment attained an average accuracy before conversion of 97.99% on the MNIST test set. The converted network resulted in a slightly lower average accuracy of 97.16% after the simulation of 50 time steps. Similarly to the NLP transformer, the accuracy of the vision spiking network evened out after 13 time steps, reaching 97.02%, as illustrated in Figure 4.10. Despite this resulting good performance, the conversion of the vision transformer to a spiking network resulted in a higher conversion loss of 0.83% compared to the NLP transformer.

Notably, despite sharing similar architectures and containing comparable numbers of spiking neurons, the vision transformer exhibited a greater conversion loss than the NLP transformer. This discrepancy may be attributed to the more complex preprocessing of input data in the vision transformer, which could have introduced additional sources of error.

5 Population-Coded Conversion

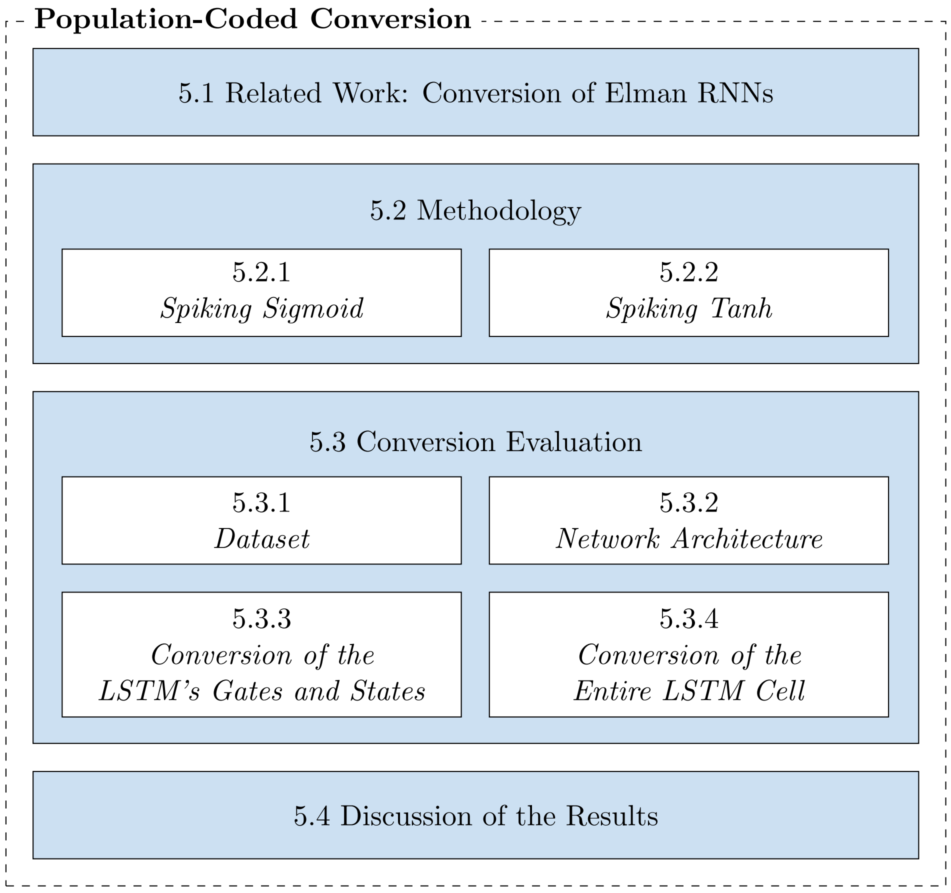

Population-Coded Conversion
5.1 Related Work: Conversion of Elman RNNs
5.2 Methodology
5.2.1 *Spiking Sigmoid*
5.2.2 *Spiking Tanh*
5.3 Conversion Evaluation
5.3.1 *Dataset*
5.3.2 *Network Architecture*
5.3.3 *Conversion of the LSTM's Gates and States*
5.3.4 *Conversion of the Entire LSTM Cell*
5.4 Discussion of the Results

Figure 5.1: Chapter structure.

The population coding strategy shares similarities with rate coding regarding its conversion process. However, population coding distributes spikes across multiple neurons with varying spiking thresholds instead of a single neuron emitting multiple spikes. With a linear distribution of these thresholds, the outcome would be identical to that of the rate-coded network, running for the equivalent number of time steps as the number of neurons in the population. Running the rate-coded network for more time steps can increase accuracy, resulting in a smaller quantization. Therefore, the advantage of a population-coded network lies in the processing speed, as it does not require running for a minimum of time to achieve reasonable accuracy. However, this approach is only beneficial if enough neurons are utilized to process the same node. In simulation environments, the disadvantage of using population coding becomes more pronounced, as multiple neurons need to be computed for a similar outcome as a rate-encoded network.

Population coding is, therefore, better suited for specialized hardware that can accommodate many neurons. Subthreshold analog neuromorphic hardware offers a significant advantage as it utilizes transistors operating in subthreshold mode, which are commonly implemented in silicon. The disadvantage mentioned in Section 2.2.3 can be transformed into an advantage, as the subthreshold analog neuromorphic hardware suffers from inhomogeneities in the spiking threshold, exhibiting a normal distribution of thresholds. These normally distributed thresholds respond with an S-shaped cumulative normal distribution as an output for a linear input.

This characteristic of population-coding is utilized to convert sigmoid and tanh activation functions in artificial neural networks ANNs, which were previously challenging to convert[1]. Figure Figure 5.1 depicts the structure of this chapter.

5.1 Related Work: Conversion of Elman RNNs

Diehl et al. [140] presented a methodology for mapping simple Elman RNNs without memory cells onto SNNs. The proposed approach involves training RNNs using backpropagation through time, discretizing their weights, and then converting these networks into spiking RNNs by aligning the output responses of artificial neurons with those of biological spiking neurons.

They demonstrated their method on an NLP task, where they discovered that brief synaptic delays were sufficient to capture the temporal dynamics required for effective question classification. Due to the lack of memory cells, the original ANN achieved only 85% accuracy on the test set. The converted SNN dropped to 72.2%, resulting in a conversion loss of over 15%.

5.2 Methodology

Different memory cells such as LSTM cells and GRU have been developed to address the vulnerability of Elman RNNs to vanishing gradients over time. However, a viable conversion method for these cells is currently lacking, as they are based on sigmoid and tanh activation functions which translate poorly to rate-coded spiking neurons. To

[1]This approach has been previously published in [40].

address this issue, non-conversion-based implementations have been presented, although they still lag behind the performance of ANNs.

The subthreshold analog neuromorphic hardware is characterized by inhomogeneities in the silicon transistors, including device mismatch, shot noise, and thermal noise, which can lead to inaccurate spiking thresholds of neurons. These sources of mismatch can either be minimized at the device level or exploited for computational purposes. In this research, we show that the inhomogeneities in subthreshold analog neuromorphic hardware can be utilized to resolve the previously mentioned difficulties of converting RNNs with memory cells.

LSTM units are widely utilized in various artificial neural network applications due to their capability to preserve information over prolonged sequences. This memorization is achieved by incorporating a cell state variable, which is updated based on multiple input-dependent non-linear functions. These functions are responsible for influencing the cell state whenever the desired information is present in the input.

The architecture of a standard LSTM cell comprises three key components: sigmoid activation functions for its input gate, output gate, and forget gate, as well as two tanh activation functions for the hidden and carry state, respectively.

Despite their widespread use in ANNs, the direct conversion of these activation functions into a spiking representation has proven to be a challenging task. This is primarily because the activation functions in LSTMs exhibit S-shaped curves while the spiking frequency of pulsing neurons increases linearly with the input [139]. This mismatch between the activation and spiking functions has motivated researchers to explore alternative approaches to convert LSTMs into spiking networks.

One such approach has been to adapt the spike frequency of single neurons to exhibit a similar activation function as the sigmoid or tanh activation functions[152]. Another approach has been to develop new SNN architectures that can effectively model the short-term memory behavior of LSTMs [112].

5.2.1 Spiking Sigmoid

In our approach, we introduce the utilization of inhomogeneous thresholds in subthreshold operating analog neuromorphic hardware to enable the approximation of sigmoid and tanh activation functions. By leveraging the property of a normal distribution's cumulative distribution function, which results in an S-shaped curve, we can effectively approximate the sigmoid and tanh functions, which exhibit similar behavior. The result is achieved by feeding input to a sufficiently large population of spiking neurons with randomly initialized, normally distributed thresholds. The number of spikes produced can be predicted with the cumulative distribution function given the input current I. The cumulative distribution function, $F(I)$, is expressed as

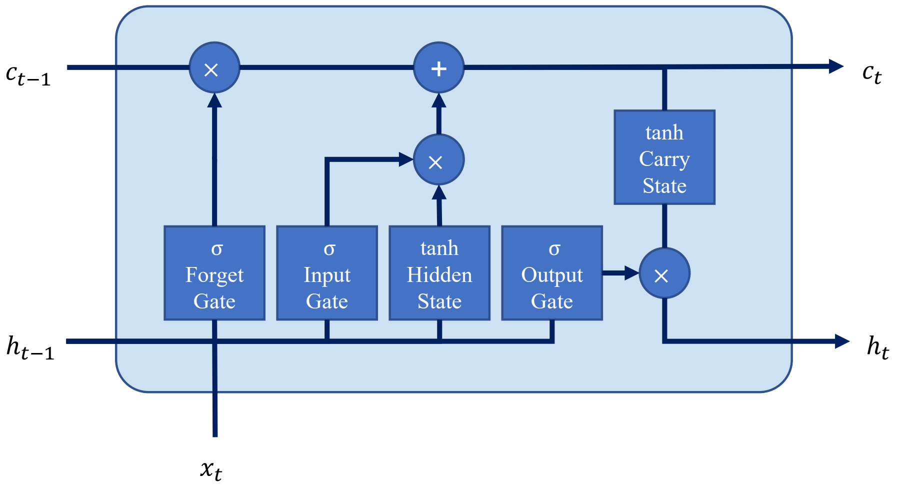


Figure 5.2: Population-coded spiking LSTM. The three gates are replaced by excitatory populations of spiking neurons, and the two gates are replaced by an excitatory and an inhibitory population, each.

$$F(I) = P(V_{\text{thresh}} \leq I) \tag{5.1}$$

where P represents the probability that the spiking threshold V_{thresh} is less than or equal to the input I.

To accurately represent the sigmoid function, the sum of spikes per population must be normalized to a value in the range of $[0, 1]$. To achieve this, the standard deviation must be set such that the cumulative distribution function closely maps the sigmoid function. The optimal standard deviation can be approximated by iteratively minimizing the error between both functions and is determined to be $\sigma_{\text{sigmoid}} = 1.75$. This value, when applied to a population of as few as 25 neurons, already produces a reasonable approximation of the sigmoid function (see Figure 5.3).

5.2.2 Spiking Tanh

The conversion of the tanh function, which ranges between $[-1, 1]$, requires further steps beyond the approximation of the sigmoid function. A second population of negative-valued (inhibitory) spikes is required, in addition to the positive-valued (excitatory) spikes emitted by the first population [139]. Both populations are initialized with the same random threshold, with the second population's sign being reversed. Like for the sigmoid activation, the optimal standard deviation can be approximated by iteratively minimizing the error between the cumulative distribution function and the tanh function and is

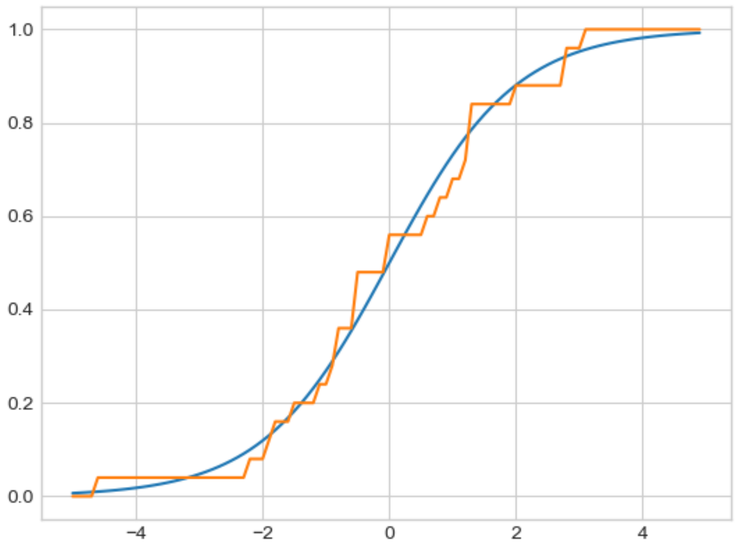


Figure 5.3: Example of an approximated sigmoid function with a population of neurons. Sigmoid function (dark) and the representation through the accumulation of spikes over the input of an excitatory spiking population of 25 neurons with a normally distributed, randomly initialized threshold with mean $\mu = 0$ and standard deviation $\sigma_{\text{sigmoid}} = 1.7$ (light). (Figure previously published in [40])

determined to be $\sigma_{\text{tanh}} = 0.88$. With this value, a small population of 50 neurons, 25 for each excitatory and inhibitory population, can visibly approximate the tanh function (see Figure 5.4).

In the event that the normal distribution is given by design, it may be necessary to make modifications to the input current to ensure the preservation of the accuracy of the approximation of the activation functions. In such instances, if the mean current $\mu_{\text{I,real}} \neq 0$ then $\mu_{\text{I,real}}$ has to be subtracted from the input current so it evens out to 0. For the standard deviations, if their values are not equivalent to the previously noted target values of $\sigma_{sigmoid} = 1.75$ and $\sigma_{tanh} = 0.88$, the input current for each population has to be multiplied by the ratio of the actual and target value. For the sigmoid population, the input current formula then reads:

$$I_{\text{sigmoid}} = I \left(\frac{\sigma_{\text{real}}}{\sigma_{\text{sigmoid}}} \right) + \mu_{\text{real}} \tag{5.2}$$

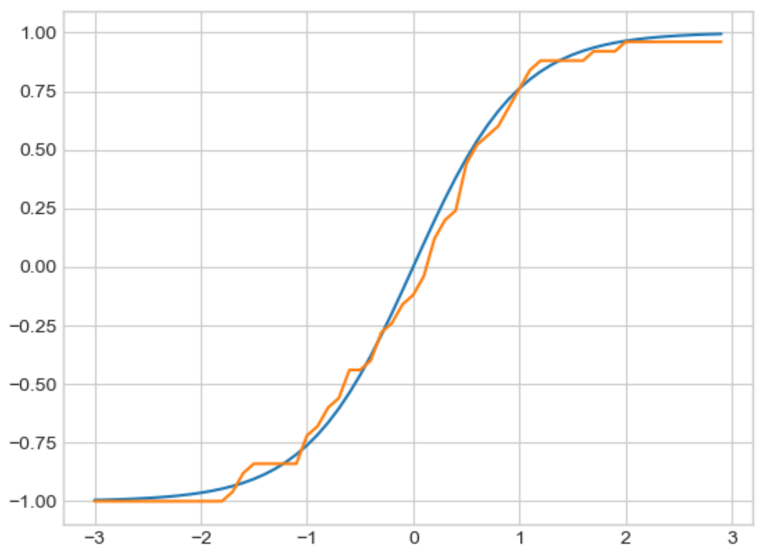


Figure 5.4: Example of an approximated tanh function with two joint populations of neurons. Tanh function (dark) and the representation through the accumulation of spikes over the input of an excitatory and an inhibitory spiking population of 25 neurons each with normally distributed, randomly initialized thresholds with mean $\mu = 0$ and standard deviation $\sigma_{\text{tanh}} = 0.88$ (light). (Figure previously published in [40])

with $\sigma_{\text{sigmoid}} = 1.75$ as explained before. For the tanh population, the input current formula reads:

$$I_{\text{tanh}} = I\left(\frac{\sigma_{\text{real}}}{\sigma_{\text{tanh}}}\right) + \mu_{\text{real}} \tag{5.3}$$

with $\sigma_{\text{tanh}} = 0.88$, accordingly. By using these modified formulas, it is possible to account for deviations from ideal conditions and ensure that the spiking populations produce accurate approximations of the sigmoid and tanh activation functions.

However, as the experiments discussed in this paper were conducted in a simulation environment, there was no need to use these compensation formulas in practice. Nevertheless, it is important to consider these modifications in real-world applications to guarantee reliable results.

5.3 Conversion Evaluation

In order to evaluate our proposed method, a LSTM-based neural network was trained on a widely-used sentiment classification dataset, which was selected for its representativeness and applicability to this particular field of study.

In the first experiment, the primary objective was to assess the performance and accuracy loss that resulted from the conversion of the individual sigmoid-based gates and tanh-based states of the cell to the spiking domain. This was accomplished by implementing the method described in the previous section and observing the resulting spike-based representations.

In order to gain deeper insights into the impact of the proposed method, the second experiment was designed to evaluate the entire network, once the necessary population sizes had been estimated based on the results of the first experiment. This experiment was critical in demonstrating the feasibility and practicality of our proposed method and its ability to handle more complex and sophisticated neural network architectures.

The results of both experiments were carefully analyzed and compared to existing benchmarks in the field, providing a comprehensive evaluation of the efficacy and robustness of the proposed method in approximating the activation functions of the standard LSTM cell.

5.3.1 Dataset

For our experiments, we have selected the IMDB movie review sentiment classification dataset [221] which comprises of 25,000 movie reviews encoded as a list of word indices and annotated as either positive or negative sentiment.

The dataset has been split into two equal parts, with one half being utilized as the training set and the other half serving as the test set. Only the 2,500 most frequent words are utilized and each review is limited to 500 words. Any review that falls short of this word limit is filled with zeros to complete the requirement. To represent the input in a spiking format, the data has been encoded as a series of one-hot vectors, with one word being fed into the network at each time step.

5.3.2 Network Architecture

The baseline network that we aim to convert to a spiking network consists of three primary components: an input layer, a hidden layer composed of 25 LSTM cells, and a single densely connected output neuron designed to perform binary classification. The network was trained with the Adam optimizer [215] over four epochs, ultimately achieving an accuracy of 92.6% on the training set and 89.8% on the test set. However, due to the increased complexity and larger populations after the conversion to a spiking network,

Table 5.1: Results of the individual gate and state conversion. The mean accuracy μ_{acc} and the standard deviation σ_{acc} in percentage points after conversion of the individual activation function in the LSTM cell to populations of different sizes. Averaged over 50 passes with random initialization of the spiking thresholds (Table previously published in [40])

Pop.	**Input Gate**		**Output Gate**		**Forget Gate**		**Hidden S.**		**Carrs S.**	
Size	μ_{acc}	σ_{acc}	μ_{acc}	σ_{acc}	μ_{acc}	σ_{acc}	μ_{acc}	σ_{acc}	μ_{acc}	σ_{acc}
1	75.7%	9.8	70.6%	11.2	52.6%	3.3	54.2%	7.8	53.5%	7.1
10	84.2%	3.8	84.0%	3.9	65.4%	12.9	61.4%	10.9	55.7%	7.9
100	88.9%	1.0	88.9%	1.0	87.0%	2.4	75.8%	9.2	68.5%	12.7
1,000	89.6%	0.2	89.5%	0.3	86.2%	1.2	82.3%	2.4	77.8%	6.9
10,000	89.7%	0.3	89.7%	0.2	85.8%	0.5	83.4%	0.7	83.2%	1.0

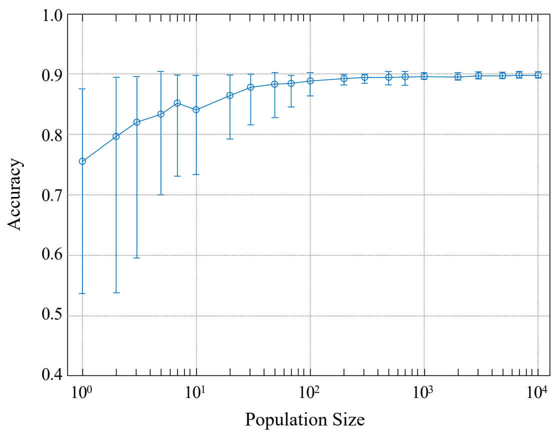
the test set used to evaluate its performance had to be reduced to only 1000 reviews in order to mitigate the computational demands.

5.3.3 Conversion of the LSTM's Gates and States

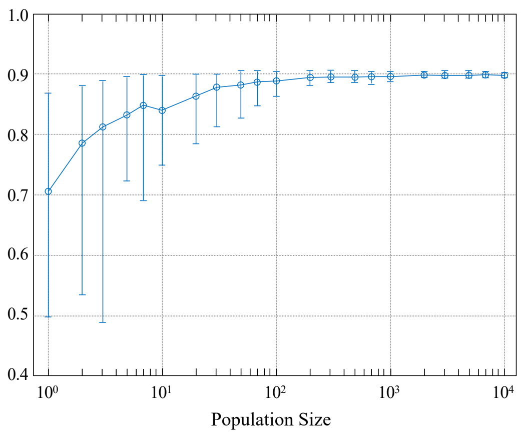
The first experiment in our study aims to replace the activation functions of different gates and states of the LSTM cell with populations of spiking neurons. The size of these populations ranges from one neuron to 10,000 neurons, allowing us to observe the influence of population size on the accuracy of the system. To account for the direct influence of the random initialization of the spiking threshold on the classification accuracy, each simulation is conducted fifty times to obtain an average performance over different random seeds.

Table 5.1 displays the results of this experiment, indicating that as the number of neurons in the population increases, the mean accuracy (μ_{acc}) increases and the standard deviation (σ_{acc}) decreases. The conversion of the sigmoid-activated gates (Figures 5.5a to 5.5c) performs well, with populations of 200 neurons for the input and output gates already performing close to the original ANN classification accuracy of 89.8% with only a small deviation between the different random seeds. Meanwhile, populations that activate the forget gate reach a peak mean accuracy of 87.0% with a size of 100 neurons, but the scattering decreases as the population size increases beyond 1,000 neurons. Although the accuracy slightly decreases, this decrease can be mitigated by averaging over more passes.

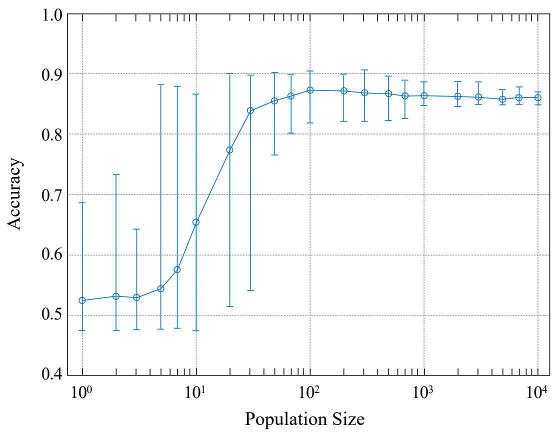
On the other hand, the conversion of the tanh-activated cell states (Figures 5.6a and 5.6b) performs worse than the sigmoid conversion. The hidden state's accuracy converges to roughly 83% with 2,000 neurons for each inhibitory and excitatory population, but a further increase of the population size does not improve the deviation. On the other hand, the activation function of the carry state needs a total of 20,000 neurons to



(a) Input Gate

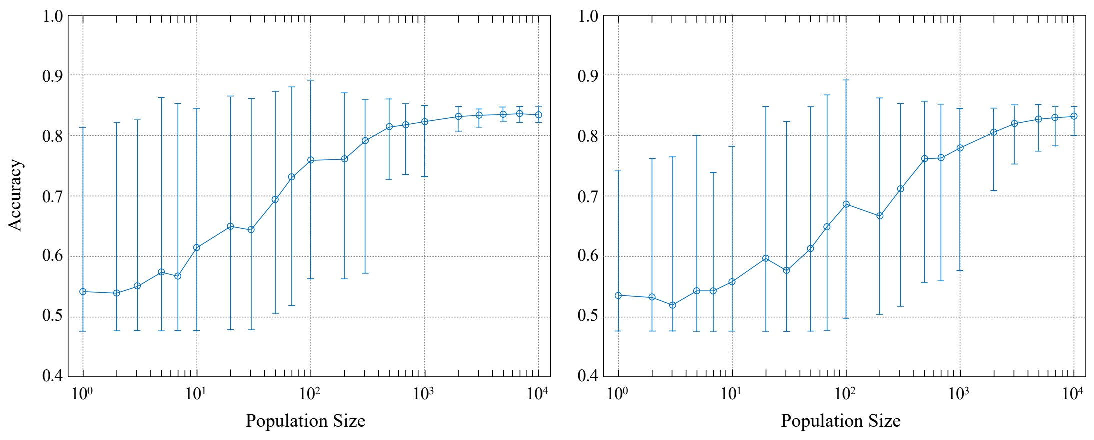


(b) Output Gate



(c) Forget Gate

Figure 5.5: Accuracy of the population-coded gates. Overview of the accuracy range after converting each of the three sigmoid-activated gates in relation to the number of neurons in the excitatory populations. The exemplary population sizes are run 50 times with randomly initialized thresholds. (Figure previously published in [40])



(a) Hidden State **(b)** Carry State

Figure 5.6: Accuracy of the population-coded states. Overview of the accuracy range after converting each of the two tanh-activated states in relation to the number of neurons in the inhibitory and excitatory populations. The exemplary population sizes are run 50 times with randomly initialized thresholds. (Figure previously published in [40])

reach just over 83% accuracy. Although the accuracy appears to have leveled off from this point on, a further increase in population size may still decrease the divergence.

Finally, the results of converting all three sigmoid functions with a population of 1,000 neurons each show that the network performs with a mean accuracy of 85.8% and a standard deviation of 1.4 percentage points, which is only slightly below the single conversion of the forget gate. This demonstrates the considerable performance of the sigmoid conversion. However, replacing the two tanh functions with a population of two times 10,000 neurons (inhibitory and excitatory) each resulting in a mean accuracy of 80.9% with a standard deviation of 0.9 percentage points, exhibiting a larger loss compared to the single tanh conversions and indicating that the tanh function is not represented as well by spiking populations.

5.3.4 Conversion of the Entire LSTM Cell

In the second experiment, an evaluation of the network with all converted activation functions was conducted. The sizes of the populations were determined based on the best results obtained from the first experiment. To be specific, 200 neurons were utilized for the sigmoid-activated input and output gates, while 1,000 neurons were employed for the forget gate. In the case of tanh activation functions, 2×2000 neurons were utilized for the hidden state, and $2 \times 10{,}000$ neurons were employed for the carry state.

Table 5.2: Results of the LSTM conversion. Overview of the mean accuracies, the standard deviation in percentage points, as well as maximum/minimum values for the conversion of an LSTM cell to population-coded spiking networks. The baseline conversion consists of 200 neurons for the input and output gate, 1,000 for the forget gate, and 2×2,000 and 2×10,000 for the hidden and carry state, respectively.

	μ_{acc}	σ_{acc}	Min	Max
Complete conversion	78.0%	1.35	72.8%	80.6%
Halved populations	77.7%	2.99	67.2%	84.4%
Doubled populations	78.1%	0.73	76.2%	80.4%

As before, the experiment was performed 50 times with randomly initialized thresholds for each run. To evaluate the impact of different population sizes, the experiment was also repeated with half and twice the number of neurons, to gather a better understanding of how the population size affects the performance of the converted network.

As per the results recorded in Table 5.2, the converted network had a mean accuracy of 78.0% with a standard deviation of 1.35%. This resulted in a conversion loss of approximately 13.2% compared to the original ANN, which had an accuracy of 89.8%. Although the mean accuracy remained roughly the same when both the population sizes were halved and doubled, the halving of population sizes resulted in slightly lower mean accuracy and more extreme outliers.

The results suggest that the use of larger populations can greatly reduce the divergence in the network's performance. In this particular case, doubling the population resulted in an approximate halving of the standard deviation. This highlights the importance of determining the most suitable population size to maximize the network's performance and minimize the deviation in its accuracy.

5.4 Discussion of the Results

We introduced a novel conversion approach to convert sigmoid and tanh activation-based neural networks into population-coded SNNs that are characterized by the use of randomly initialized, normally distributed spiking thresholds. The conversion methodology was tested on a Long-Short Term Memory (LSTM)-based recurrent neural network for sentiment classification.

The first step of the conversion process involved the evaluation of individual activation functions. The results of this experiment showed that the conversion of the tanh activation functions resulted in the largest loss of accuracy. Therefore, improving this specific conversion process could lead to the most improvement in performance. One possible approach for improvement is to use other cell types such as Gated Recurrent Units (GRUs), which only contain a single tanh activation function. Another approach could be to train the network using the cumulative normal distribution function for the gates, which could significantly reduce the conversion loss and improve the performance.

In the second step of the experiment, the entire converted network was evaluated. To achieve reasonable performance, each LSTM cell was replaced by 25,400 neurons, resulting in a total of 635,000 neurons in our example. The accuracy of the converted network showed a conversion loss of 13.2%. However, doubling the size of the population reduced the scattering of the accuracy in different trials of the same experiment, without increasing the overall performance.

One of the main benefits of the conversion approach is that it can leverage the potential drawback of neuromorphic hardware as an advantage. The use of population-coded SNNs eliminates the need for multiple time steps to process the inputs, unlike rate-coded SNNs, where inputs have to be presented multiple times. The conversion process can result in a simple method for future sub-threshold-operating analog neuromorphic hardware.

However, the sheer number of necessary neurons makes this approach impractical in a simulation environment. The small example network in our experiment increased the computational requirements several times, making it challenging to implement on large-scale compatible hardware. A comprehensive evaluation of the conversion approach would require access to compatible hardware, which is currently not available on this scale. Nonetheless, our approach shows promising results and provides a simple conversion methodology for future neuromorphic hardware implementations.

6 Temporal-Coded Conversion

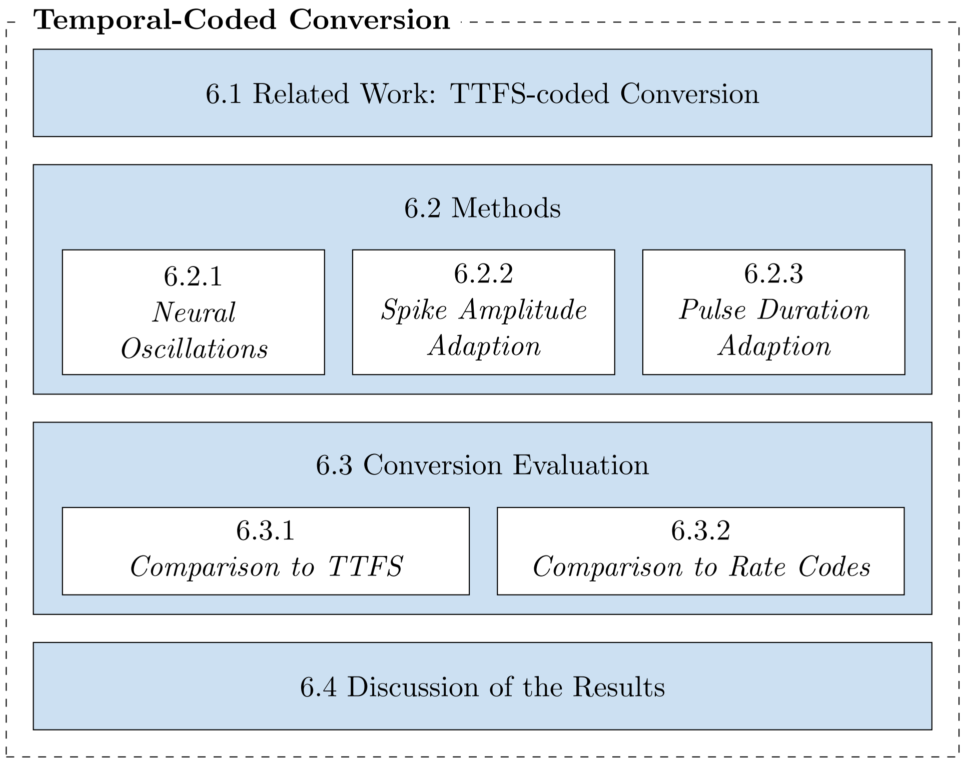

Temporal-Coded Conversion
6.1 Related Work: TTFS-coded Conversion
6.2 Methods
6.2.1 Neural Oscillations
6.2.2 Spike Amplitude Adaption
6.2.3 Pulse Duration Adaption
6.3 Conversion Evaluation
6.3.1 Comparison to TTFS
6.3.2 Comparison to Rate Codes
6.4 Discussion of the Results

Figure 6.1: Chapter structure.

In the previous chapters, we showed the conversion using rate and population coding. Both approaches need multiple pulses to encode the information, the number being limited by the runtime in rate-coded networks and in the size of the population in population-coded networks. In this chapter we introduce a sparse approach that encodes the information in the pre-spiking interval, allowing neurons with a positive activation to fire only once. The two temporal coding mechanisms that can be used for conversion derived in Section 3.1 are TTFS and phase coding. We give a brief overview of previous work about the TTFS-coded conversion and propose a conversion approach with a phase encoding.[1]

[1]This approach has been previously published in [45].

6.1 Related Work: TTFS-coded Conversion

In the field of converting ReLU-activated ANN to temporally encoded SNNs, Rueckauer et al. presented the *TTFS base* approach [38]. This method models incoming action potentials as a weighted permanent input current, triggering a spike when the resulting membrane potential surpasses a threshold, resetting the neuron's state to zero (mechanism illustrated in Figure 6.2). The membrane potential can be calculated with

$$u_i(t) = \sum_{j \in \Gamma_i^<} w_{ij}(t - t_j^{(0)}) + b_i t \tag{6.1}$$

with Γ_i denoting the set of pre-synaptic neurons. The time when neuron i emits a spike can be determined with

$$t_i^{(0)} = \frac{1}{\mu_i} \left(\theta + \sum_{j \in \Gamma_i^<} w_{ij} t_j^{(0)} \right) \tag{6.2}$$

with $u_i(t_i^{(0)}) = \theta$. Then, the corresponding instantaneous firing rate r_i is equal to $1/t_i^{(0)}$. In order to prevent further spikes, the refractory period is prolonged longer than the runtime of the simulation.

However, one notable drawback of the *TTFS base* method is the potential premature firing of spikes, which can occur when a positive input elevates the membrane potential above the threshold prior to the arrival of a balancing negative input. To address this issue, Rueckauer et al. proposed the *TTFS dyn thres* approach that implements dynamic thresholds, which are increased in proportion to the magnitude of missing inputs. Neurons, therefore, fire twice: first, to signal a missing input and second, to contain the encoded information.

As the authors mention, the approach suffers from the premature firing of spikes, such as when a brief positive input surges the membrane potential past the threshold, despite an impending negative input that would overall prevent the neuron from firing. To reduce this problem, a second method is proposed, called *TTFS dyn thresh*. There, the neurons are equipped with a dynamic threshold that increments based on the magnitude of the missing input. Therefore, neurons fire twice: once to signal a missing input and again to contain the temporal information.

In a comprehensive evaluation of the *TTFS base* and *TTFS dyn thres* methods, Rueckauer et al. conducted experiments on the MNIST dataset using a 5-layer LeNet5-CNN. The results of these experiments showed that the *TTFS base* method resulted in roughly 1% conversion loss, whereas the *TTFS dyn thres* method yielded a slightly improved result with 0.8% loss. A third presented approach, named *TTFS clamped*, does

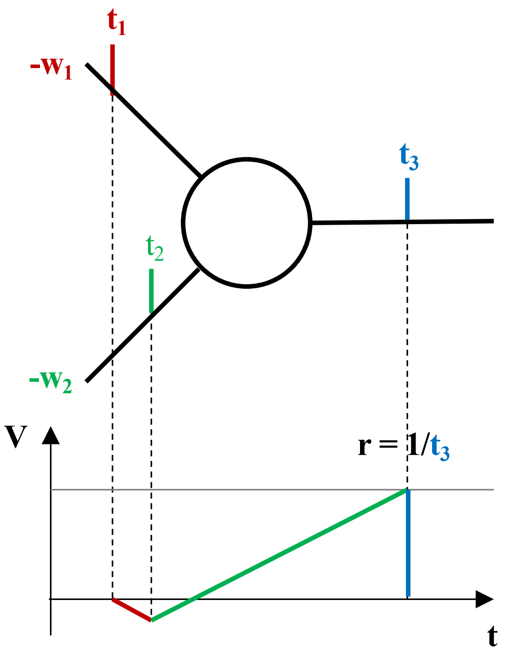


Figure 6.2: Spike generation with TTFS spike encoding. Figure from [38]

not rely on additional spikes but on the re-training of the entire network with a modified ReLU activation function, which clips the lower activation values, leading to a further improved performance. As this is a *constrain-then-train* approach (see Section 2.3.3), it is not in the scope of this work.

6.2 Methods

However, this model also presents certain limitations that may not be beneficial for temporal conversion.

- *No temporal information:* The removal of the leak in the IF neuron model also removes the temporal information of the input.

  Figure 2.5c and Figure 2.5d demonstrate that the membrane potential remains constant following every input spike, while the input timing can be freely shifted before the third spike occurs, without affecting the overall output timing.

- *Multiple pattern recognition:* Different input patterns can result in a spike at the same time, leading to multiple pattern recognition by a single neuron. On the other hand, ANN neurons are trained to recognize specific patterns and only one linearly independent input can maximize the activation.

This neural system enables multiple input patterns to converge on a single spike time, such as a cluster of large spikes or a series of small ones. Consequently, an individual neuron can be leveraged to recognize a wide range of patterns across various hierarchical levels. In contrast, traditional ANN neurons are typically trained to recognize very specific patterns and respond maximally to only one linearly independent input

- *Discarding low activations:* When a spiking neuron receives insufficient input, the membrane potential fails to cross the threshold and therefore remains silent. For instance, if either the first or second input in Figure 2.5b were zero, the threshold would not be reached. In contrast, ReLU neurons will output a non-zero value as long as the total input is positive.

- *Premature spiking:* When an action potential is triggered, information from subsequent inputs is lost [38]. For instance, as seen in Figure 2.5c and Figure 2.5d, a large negative input at 9ms has no effect on the spike timing, despite the aggregate input being insufficient to trigger a response.

6.2.1 Neural Oscillations

The lossless conversion of ReLU-activated ANNs into temporal coded SNNs presents a unique challenge in ensuring the preservation of information without incurring any loss in the process. To achieve this, it is crucial that no spikes are emitted until all the inputs to a neuron have arrived. To this end, our method aims to address this issue by utilizing two distinct time windows within each layer of the network.

The first time window is designated for receiving incoming spikes, while the second window is used for emitting outgoing spikes (if any). This approach guarantees that the neuron will spikes exclusively if and when its membrane potential is positive. This approach depends on a "driving force" to ensure that the spiking threshold is reached to emit a spike.

The two-time window method provides a systematic and controlled method for converting ReLU-activated ANNs to temporal coded spiking networks, achieving lossless information transfer on the condition that the temporal resolution of the spike time is as high as the precision of the ReLU activation function.

In order to compensate for the limitations posed by IF neurons (see Section 2.1), it is essential to meet certain requirements for the globally referenced temporal encoded conversion.

Fixed Windows.

The first requirement is the use of fixed windows in order to address the issue of prematurely emitted spikes in IF neurons. This is achieved by utilizing a two-phase approach in processing the input signals. The first phase, referred to as the *listening phase*, is designed to allow the input signals to influence the membrane potential of the neuron without eliciting an action potential.

In the second phase, referred to as the *transmitting phase*, an action potential is generated by the neuron with the exact timing being dependent on the membrane potential achieved during the previous listening phase. This phase must have a duration equal to that of the listening phase to ensure that all input signals have been received before an action potential is generated.

Normalization. The weights of the original ANN are normalized to guarantee that no spikes are generated within the first time window. This is achieved by normalizing the weights such that, when a neuron receives its maximum possible input within the initial time frame, the neuron's membrane potential stays precisely at or below the spike threshold. During the transmission phase, the membrane potential undergoes a linear increase. As a result, neurons with high membrane potential emit an action potential early in the time frame, while those with lower potential emit action potentials later in the same window. Neurons with resting or negative potentials are silenced until after the time frame has ended and thus do not influence subsequent neurons in the network.

Neural Oscillation. During the transmitting phase, the membrane potential remains constant due to the absence of an input current and no leakage of the neuron. To push the positive membrane potential above the spiking threshold, a linear increase is required. This is achieved through the implementation of a globally referenced neural oscillation in the form of a continuous input current that periodically switches from positive to negative polarity after each phase. This oscillation is configured so that in the absence of any input, the membrane potential will merely reach the threshold without crossing it or triggering an action potential by the end of the transmitting phase. If the membrane potential has a positive value after the listening phase, it will cross the threshold, with higher potentials resulting in earlier spiking and lower potentials resulting in later spiking.

Bias. The bias in rate-coded conversions is commonly modeled as a constant current injected into the neuron. However, with neural oscillations, this method would result in premature spiking during the listening phase if a bias with a positive value is present. To avoid this issue, an additional, fully activated neuron is employed in each layer. The transmitted spike is weighted by the corresponding value of the bias and propagated to all neurons in the downstream layer. This methodology has been visualized and depicted in Figure 6.3. To prevent subsequent spikes within subsequent cycles, the refractory period of the neurons is set to a duration exceeding the total execution time of the network.

Refractory Period. In order to prevent the occurrence of additional spikes in subsequent oscillating cycles, a refractory period is established. It is set longer than the total runtime of the network.

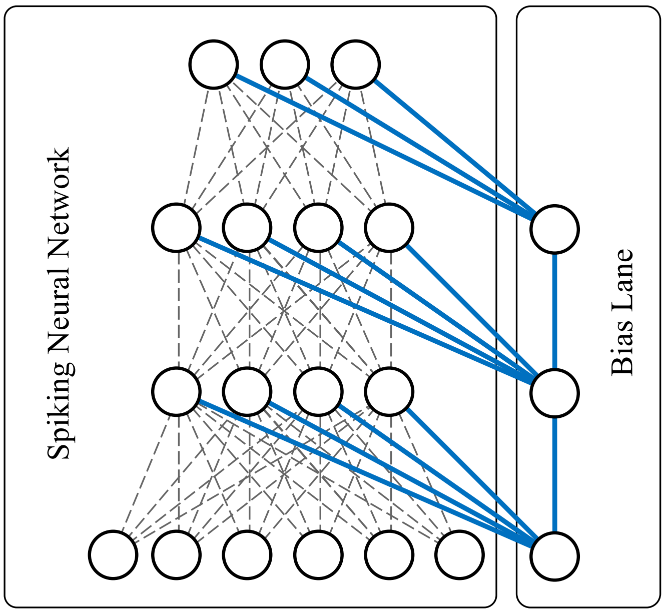


Figure 6.3: Bias lane for temporal-coded conversion. One additional neuron per layer that injects an additional pulse weighted by the value of the bias (Figure previously published in [45])

Weighting of Input. As the IF neurons do not incorporate any leakage, it is ambiguous for the membrane potential whether the input occurs early or late. Therefore, the arriving spikes must be adjusted according to their precise timing within the listening phase.

The voltage response of spiking neurons is dependent on the cumulative charge of the spike, which can be represented mathematically as $q = \int I(t)dt$ [61]. To adapt the incoming spikes to the IF neurons, two options are available. The first option is to adjust the amplitude of the incoming spike, the second option is to adjust the duration of the pulse, as discussed in the next section.

Adaption of the spikes in a neural network can be accomplished through two methods: Firstly, by either reducing the amplitude of the spike or secondly, by adapting the duration of the spike.

$$a_i^l := \max\left(0, \sum_{j=1}^{l-1} W_{ij}^l a_j^{l-1} + b_i^l\right) \tag{6.3}$$

6.2.2 Spike Amplitude Adaption

The amplitude of a spike has to decrease the later its arrival time becomes. An adaptive resistor integrated with the neural oscillation, for example, can be employed to reduce the amplitude of the spike as it occurs later in the listening phase. The membrane potential $u_i(t)$ of the neuron i is determined by the following equation:

$$u_i(t) = \sum W_{ij} \frac{t_j}{T} \delta_j + b_i^l \tag{6.4}$$

where T is the duration of the listening phase, δ is the Dirac delta function and W and b are the weights and biases, respectively.

The listening phase starts at a value of $-u_t hr$, reaches 0 between phases, and grows to $u_t hr$ by the end of the second phase. The membrane potential in subsequent layers must be phase-shifted.

The time at which a resulting spike is generated, relative to its time frame, can be calculated using the equation:

$$t_i(u) = (u_{thr} - u_{l+1})T \tag{6.5}$$

where $ul + 1$ represents the membrane potential at the beginning of the transmission phase in layer l and the beginning of the listening phase in layer $l + 1$, respectively.

6.2.3 Pulse Duration Adaption

With pulse duration adaptation, a constant current is fed to the neuron following an input event for the remaining portion of the listening phase. As a result, early spikes carry more weight in affecting the final membrane potential compared to those that spike later. This can be achieved by extending the pulse duration to the length of a single oscillation cycle. The input current alternates between positive and negative, with a negative current during the listening phase and a positive one during the transmitting phase.

For connecting layers, the sign of the pulses and threshold are reversed and set to $-u_{thr}$. The membrane potential of the neuron i at the conclusion of the listening phase can then be computed as follows:

$$u_i(t) = \sum W_{ij}(T - \Delta t_j)q_p + b_i^l q_p \tag{6.6}$$

where Δt_j represents the time of the incoming pulse relative to the start of the phase, and q_p represents the total charge of a pulse. The spike time can be calculated as expressed in equation 6.5.

6.3 Conversion Evaluation

It is important to note that while the approach used in this study is mathematically the same as traditional ML, as long as the temporal resolution is as high as the precision of the ReLU activation function. Therefore, comparing the accuracy of the results obtained through simulation is not a meaningful measure. Hence, instead of evaluating the accuracy, we opt to compare the inference time of the current approach to previous methods. In addition, we establish a link between the spiking operations employed in our methodology and existing research evaluating the performance of rate-coded conversion techniques"

In addition, we also establish a connection between the spiking operations performed in our approach and existing research evaluating the performance of rate-coded converting techniques. By doing so, we aim to provide a comprehensive understanding of the efficiency and effectiveness of our approach and its relationship with existing methods.

6.3.1 Phase Coding Compared to TTFS Coding

The results from Rueckauer and Liu [38] showed that their method based on TTFS led to premature spiking of neurons. To address this issue, an adapted version that uses dynamic thresholds was proposed, leading to improved performance but also doubling the number of spikes. We compare our approach to the two TTFS conversion methods by training two ANNs with different depths on the MNIST dataset [187].

As shown in Table 6.1, we achieved similar performance to the previous study by recreating their results using the LeNet-5 network architecture. Our approach showed the same accuracy of 98.7% compared to the original ANN. On the other hand, the TTFS base and TTFS dyn thresh methods showed a loss of 3.85% and 1.77%, respectively. The number of spikes roughly doubled for TTFS dyn thresh and our approach falls in between the two.

As we scale the network to a deeper architecture, specifically a 9-layer CNN built upon LeNet-5 with additional convolutional layers, both TTFS base and TTFS dyn thresh methods exhibit a substantial decline in performance. This is due to the TTFS base method suffering from premature spiking, as pointed out by Rueckauer et al. The TTFS dyn thresh method partially alleviates the issue, but does not fully eliminate it, since the first spike is only emitted when the initial input arrives, resulting in lost information when a neuron in a subsequent layer has already fired its second spike.

As the network is scaled, this issue worsens, with the interval between spike times across different layers growing, leading to the suppression of lower-activated inputs as the network becomes deeper. Notably, in the deep network, the TTFS base method required the fewest spikes, whereas the TTFS dyn thresh method demanded the most.

Table 6.1: Comparison to previous temporal coding approaches. Comparison of of our method to previous temporal coding approaches [38] on the MNIST dataset. The LeNet-5 Architecture contains 7620 neurons (7625 including the bias lane) and the 9-layer CNN (a variant of the LeNet-5 architecture with additional convolutional layer) in a total 13900 neurons (13909 including the bias lane) (Table previously published in [45])

Model	**Network**	**Loss**	**# spikes**
TTFS base	LeNet-5	3.85%	2011
TTFS dyn thresh	LeNet-5	1.77%	3867
This work	LeNet-5	-	2854
TTFS base	9-layer CNN	9.18%	4670
TTFS dyn thresh	9-layer CNN	6.36%	9132
This work	9-layer CNN	-	8234

Our approach is once again in between the two, but closer to the TTFS dyn thresh method.

6.3.2 Phase Coding Compared to Rate Coding

For a thorough comparison with rate-coded conversions, we trained a neural network on a subset of the speech commands dataset [222], employing a network architecture analogous to that previously described by Blouw and Eliasmith [223]. This architecture consisted of an input layer with 3920 neurons and two hidden layers each containing 256 neurons. Following the methodology of the original work, we employed a Hybrid SNN approach, in which the input was digitally processed and only spikes generated in the first layer were counted.

In terms of accuracy, our ANN model achieved a comparable 82.1% compared to the 81.8% reported in the original work. Unlike the rate-coded approach, our approach retained the same accuracy after conversion, while the rate-coded approach suffered a decrease in accuracy to 81.0%. In terms of computational efficiency, the original work required 61,362 SOPs, while our approach reduced this amount by a factor of approximately 15 to an average of 4081 SOPs over the test set. It is worth noting that almost all neurons in the input layer fired a spike, while the hidden and classification layers collectively produced fewer than 200 spikes.

The original study assessed the energy efficiency of the NN when operating as an ANN running on a neural accelerator or as a converted SNN on an Intel Loihi chip. The results showed that the SNN had a $4.11\times$ lower energy consumption compared to the original network. However, it is crucial to keep in mind that the reduction in spikes in our approach, by a factor of 15, is accompanied by an additional need for a global

Table 6.2: Comparison to rate coded approaches. Comparison of our method to rate coded approaches on the speech recognition dataset. The architecture contains an input layer with 3920 neurons and two hidden layers with 256 neurons. Our approach needs one additional neuron per layer (Table previously published in [45])

Model	ANN	SNN	# spikes
(Blouw et al. 2020)	81.8%	81.0%	61362
This work	82.1%	82.1%	4081

reference, which consumes additional power and therefore, cannot be directly used as a factor for potential energy reduction.

6.4 Discussion of the Results

In this chapter, we presented a novel method for converting conventional, activation-based neural networks into SNNs that are temporally coded. Our approach leverages the use of globally referenced neural oscillations to convert the ReLU activation output to a listening phase and a subsequent spike emission during a transmission phase. This mathematical equivalence ensures a lossless conversion from activation-based to spiking networks.

We evaluate the performance of our conversion method through two experimental studies and demonstrate that it is scalable for deep network architectures, resulting in a reduction of spikes by a factor of 15 compared to rate-coded conversion methods.

While our method is lossless in theory and simulation, practical implementation on neuromorphic hardware may introduce accuracy loss. Digital neuromorphic hardware quantizes the network, making accuracy dependent on the time steps of the simulation. Analog neuromorphic hardware is susceptible to noise interference, which can also impact accuracy and may require a slower operating speed.

Additionally, our method requires the use of a global reference, which may not be compatible with current neuromorphic hardware. Furthermore, the use of neural oscillations contributes to an increase in energy consumption. In conclusion, our method offers a lossless conversion approach for neural networks to be encoded temporally, providing new opportunities for implementation on neuromorphic hardware.

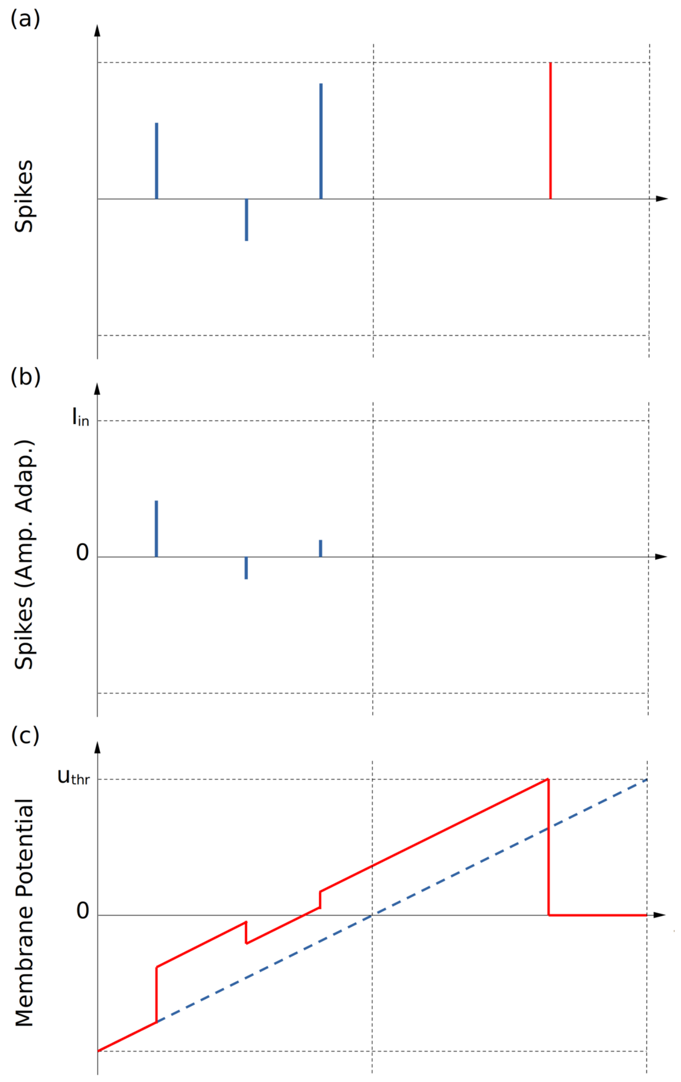


Figure 6.4: Neural oscillation method with pulse duration adaption. (Left half: listening phase, right half: transmission phase). (a) Input spikes during listening (left) and action potential generated from the two methods (right). (b) Input and (c) resulting membrane potential with spike adaption method. (d) Input and (e) resulting membrane potential with pulse duration adaption (Figure previously published in [45])

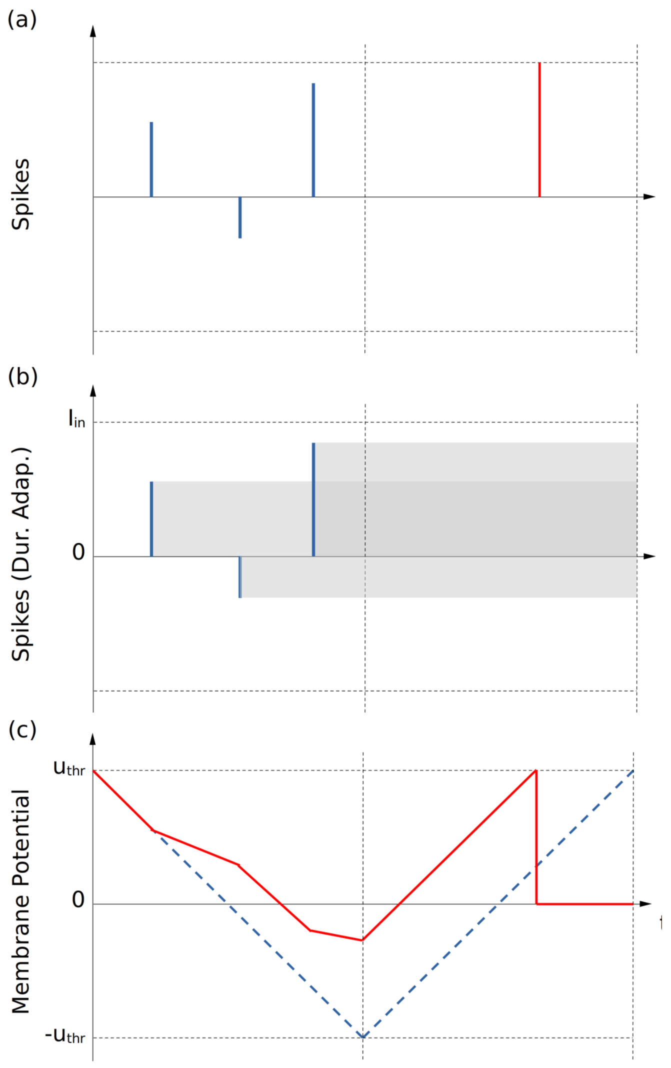


Figure 6.5: Neural oscillation method with pulse duration adapation. (Left half: listening phase, right half: transmission phase). (a) Input spikes during listening (left) and action potential generated from the two methods (right). (b) Input and (c) resulting membrane potential with spike adaption method. (d) Input and (e) resulting membrane potential with pulse duration adaption (Figure previously published in [45])

7 Summary and Conclusion

The conversion of Artificial Neural Networks (ANNs) to Spiking Neural Networks (SNNs) has been recognized as the most effective training approach for achieving high accuracy in SNNs on common datasets [224]. Recent advancements in hardware accelerators for ANNs have made it essential to minimize the number of spikes generated by neuromorphic hardware to remain competitive in the field.

To address this challenge, this thesis aimed to explore methods for optimizing the conversion of ANNs to SNNs for potentially energy-efficient operation on specialized hardware.

In the first chapter, we provided a rationale for the study and formulated Research Questions (RQs) that guided our investigation. Chapter 2 provided an overview of the background and foundational concepts relevant to our research, including neuron models, neuromorphic hardware, and SNN learning algorithms. In Chapter 3, we discussed the conditions required to convert ANNs to SNNs, highlighting the spike encodings, network architectures and normalization algorithms that influence the performance and accuracy of the resulting SNNs. Chapter 4 demonstrated the optimization of rate-coded SNNs for development and inference. Building on this, Chapter 5 presented novel methods for energy-efficient conversion to specialized hardware using population codes. Finally, Chapter 6 explored the use of sparse temporal codes for conversion to SNNs, which are effective in reducing energy consumption while maintaining high accuracy.

In this chapter, we provide a summary of the key findings and conclusions derived from each chapter, highlighting the contributions of this research to the field of SNNs.

7.1 Theoretical Background

In this research, Chapter 2 serves as the theoretical foundation for the investigation. Specifically, the chapter discusses the various neuron models that are relevant to the study. First, biologically-plausible models like Hodgkin-Huxley and Morris-Lecar were introduced. These models aim to replicate the behavior of biological neural systems in a biologically realistic way.

Next, the chapter introduces biologically-inspired models, which attempt to mimic the behavior of biological neural systems, although not necessarily with biological plausibility. Examples include the Fitzhugh-Nagumo, Hindmarsh-Rose, and Izhekevich neuron models.

The chapter then discusses the Integrate-and-Fire (IF) model family, which is a simpler group of neurons based on the integration of their input currents. Examples of this family include, among others, the simple IF neuron, the Leaky Integrate-and-Fire (LIF), and Resonate-and-Fire (RF) neuron models.

In the following section, the chapter provides an overview of the neuromorphic hardware landscape. Here, the distinction is made between digital implementations, which are further subcategorized into flexible Field-Programmable Gate Arrays (FPGAs) and custom Application-Specific Integrated Circuits (ASICs) like Loihi and TrueNorth, and analog implementations, which can be further divided into superthreshold operating ones like BrainScaleS and subthreshold operating ones like Neurogrid.

Next, the chapter reviews the training algorithms for SNNs. This includes biologically-plausible learning algorithms like Spike-Timing-Dependent Plasticity (STDP), supervised learning algorithms that attempt to adapt spiking variants of backpropagation, and the conversion methods that form the basis of this research.

Conversion approaches typically involve conventionally training an ANN and then adapting the network to work with spiking neurons. With this approach, the best-performing SNNs to date have been created for common datasets. Two subcategories of conversion methods are distinguishable: firstly, constrain-then-train, where the ANN is adapted to the properties of the spiking neurons before training, so no further adaption is needed in the conversion process. Secondly, binarization of the ANN, where the activation function of the ANN can only take binary values and therefore translates into single spikes per spiking neuron after conversion, which does not require further modifications in the conversion process.

As the goal of this research was to accelerate the development process of SNNs, the approach that does not require the retraining of the ANN is focused on, as the training process is typically time-consuming, and computationally expensive.

7.2 Conversion Challenges and Opportunities

In Chapter 4, we analyze the landscape for the possibilities of converting ANNs to SNNs. Firstly, we explore the different spike encodings that can be used in SNNs and determine which ones are suitable for conversion. After an analysis, we found that count rate coding, density rate coding (for evaluation purposes), population coding, and two temporal encodings (Time-to-First-Spike (TTFS) and phase coding) are effective methods that can be used for conversion, and will serve as the foundation for the subsequent Chapters 4 to 6.

Additionally, we examine the current state-of-the-art neural network architectures and their corresponding common datasets. For example, Convolutional Neural Networks (CNNs) have shown outstanding performance in image classification, with Residual Neural

Networks (ResNets) showing the best accuracy to date, while for object detection, the implementation of Feature Pyramid Networks (FPNs) have been found to be the most effective. Meanwhile, transformer networks have emerged as the standard for processing sequential data, particularly for Natural Language Processing (NLP) tasks. On the other hand, Recurrent Neural Networks (RNNs) with memory cells are used for sequential data with unknown input lengths, such as voice assistants.

We also discuss the normalization process in the ANN-to-SNN conversion process. In order for the conversion to work, the weights of the ANN, which are usually trained with backpropagation, must be normalized so that they do not exceed the maximum firing rate of the spiking neurons in the SNN. A simple method involves passing the dataset through the ANN to detect the highest firing rates, which can then be used for normalization. However, this method may lead to very long convergence times of the converted SNN if the weights are normalized from single outliers. Hence, commonly robust normalization is implemented, where only the p^{th} percentile of the maximum activation of the original ANN is used for normalization, with p being commonly set between 99% to 99.999%. Furthermore, we present an approach, which involves the use of a synthetic dataset based on noise to approximate the best value for p. Additionally, we introduce an extension to the normalization process, namely channel-wise normalization, which aims to improve the normalization accuracy and efficiency by normalizing each channel of the ANN independently.

7.3 Methods and Evaluation

7.3.1 Rate-Coded Conversion

Chapter 4 focuses on the topic of rate-coded conversion, which is currently considered the de facto standard in conversion techniques. This is primarily because both the Rectified Linear Unit (ReLU) activation function and the firing rate of spiking neurons increase linearly with their input. While previous studies have explored the potential of rate-coded conversion, they often relied on simple networks and datasets to validate their approaches. Our work seeks to evaluate whether these approaches are scalable to very deep networks with large datasets. To accomplish this, we first evaluate optimization algorithms for simulating SNNs and present our unbiased quantization method.

Subsequently, we assess the optimization algorithms on an image classification dataset using state-of-the-art ResNets with a depth of up to 101 layers. This is, to the best of our knowledge, the deepest SNN to the date of publication. By evaluating these optimization approaches, we can determine the most effective methods for SNN conversion. We then apply these approaches to the conversion of a spiking FPN with a ResNet as its backbone on a large object detection dataset. As a result, we achieved the highest accuracy for spiking object detection on that dataset.

Given the increasing interest in transformer networks, we propose an approach for the conversion of both NLP and vision transformer networks. Our vision transformer approach demonstrates comparable performance for image classification to state-of-the-art CNN conversion, while our NLP transformer can outperform previous conversion approaches for a sentiment classification dataset. However, the embedding layers cannot be directly converted to spiking neurons and therefore must be preprocessed in conventional ways.

7.3.2 Population-Coded Conversion

Chapter 5 explores the possible implementation of population codes into the conversion approach. Population codes and count rate codes both average spikes over a certain period, but population codes use a group of neurons, whereas count rate codes use a single neuron. Therefore, population codes can produce results more quickly, as the neurons in the population can produce spikes in parallel, while count rate-coded neurons require multiple timesteps. Therefore, population codes offer an advantage in processing speed, especially when every layer produces at most one spike per neuron. As a result, a large number of neurons is required to represent the activation functions in the original ANN and would favor the use of subthreshold analog neuromorphic hardware due to their small neuron size in silicon.

The expected variability of transistor dimensions due to manufacturing inaccuracies in small process nodes leads to variability of spike activity over a population of neurons. Since the cumulative normal distribution exhibits an S-shape, we can use the fact that the thresholds are spread around a target value to convert similar-shaped activation functions in ANNs to populations of spiking neurons with normally distributed spiking thresholds. Therefore, we can take advantage of this property and convert s-shaped activation functions, like sigmoid and tanh, to populations of spiking neurons with normally distributed spiking thresholds.

Most common network architectures nowadays are implementing the ReLU activation function, except for RNNs with memory cells that consist of gates and states activated by sigmoid or tanh functions. Therefore, we evaluated our approach on RNNs with Long Short-Term Memorys (LSTMs) as memory cells for an NLP task. As the cumulative normal distribution and the S-shaped curves of both sigmoid and tanh do not perfectly match, we experienced a comparatively high conversion loss of 13.2% between ANN and SNN. Nevertheless, this marks the first successful conversion of RNNs with memory cells to SNN to the best of our knowledge.

7.3.3 Temporal-Coded Conversion

Recent research has shown that modern hardware accelerators for ANNs are highly energy-efficient, to the point where an equivalent SNNs should require no more than 1.72 spikes per neuron for optimal efficiency. However, we could not achieve this efficiency with rate codes or population codes. Thus, we turn our attention to temporal codes for conversion, which encode information in the time between spikes rather than the number of spikes.

In our background analysis, we evaluated two temporal coding methods for conversion: TTFS and phase codes. While TTFS has been used previously, it failed to scale to deeper networks due to premature spiking of neurons. Attempts to improve this by including a second spike per neuron also failed. Therefore, we explore the use of phase coding.

In phase coding, a global reference signal (in our case an oscillation) creates time windows for each layer. Each layer has a listening phase, during which it receives all input spikes from previous neurons, and a transmitting phase, during which it emits a spike whose timing is based on the membrane potential from the received spikes. We demonstrate that this approach is mathematically equivalent to linear activations in ANNs and can be converted losslessly. However, there is a trade-off between speed and accuracy when implementing this approach in hardware, as accuracy depends on how precisely spike times can be measured. Slower networks may result in higher accuracy, but additional energy is required to implement the oscillating signal.

7.4 Research Questions

For this work, we raised four research questions. These have already been introduced in the first chapter, but we will briefly recap these and show the contribution of this work:

RQ 1 (Spike Encodings)
Which spike encodings can be used for mapping the activation function of ANNs to spiking neurons?

and

RQ 2 (State-of-the-Art Architectures)
Which state-of-the-art neural network architectures and operators are missing a conversion approach?

The first two questions focus on spike encodings and today's commonly applied architectures, respectively. In Chapter 3, we analyzed the spike encoding landscape and identified suitable encodings for the conversion approach. In Chapters 4 to 6, we evaluated existing conversion methods based on these encodings or presented our approaches. We also analyzed today's state-of-the-art neural network architectures, including ResNets, FPNs for object detection, transformer networks for NLP, and RNNs in combination with memory cells.

RQ 3 (Rapid Prototyping)
Can the accuracy of the converted networks already be estimated before the conversion process?

The third research question concerns the accuracy estimation of converted networks before the conversion process. With the activation of linear activation functions like ReLU and the firing rate of spiking neurons increasing linearly to their input, the accuracy for converted networks should be the same. However, since spiking neurons have a maximum firing rate, normalization is necessary, introducing a speed-accuracy trade-off that requires careful tuning of a normalization hyperparameter. We present an approach in section 3.3 and evaluate it in Section 4.3 that uses ReLU1 to approximate the resulting accuracy of the converted SNN, making it possible to evaluate the hyperparameter before conversion.

RQ 4 (Scalability and Optimization)
Do the existing approaches scale to very deep Neural Networks (NNs) with large datasets and can they be further optimized?

The fourth research question concerns the scalability and optimization of existing approaches for very deep neural networks with large datasets. Since simulations of SNNs are usually more computationally demanding than running ANNs, common conversion approaches limit their evaluations to simple network architectures or datasets. However, we show the conversion of converted SNN with over 100 layers in section 4.2 and demonstrate the use of FPNs for object detection on a large dataset in section 4.3.

The overall question

> *How can ANN-to-SNN conversion approaches be used for faster development of energy-efficient, state-of-the-art spiking networks?*

can carefully be answered with *yes*. We showed improvements in the speed of the development process, presented approaches for state-of-the-art neural networks, and demonstrated high energy efficiency. However, recent hardware improvements for running ANNs might surpass the efficiency of SNNs. Nonetheless, the development of neuromorphic hardware is still in its early stages and may offer even higher efficiency in the future.

7.5 Conclusion

This research work aims to evaluate the process of converting ANNs to SNNs using different encoding schemes. The state-of-the-art conversion approach relies on rate encodings. However, in this study, we proposed novel optimization methods for improving the inference time of spiking networks. Additionally, we introduced an approach to estimate the SNN accuracy from the ANN. One of the main challenges in the production of subthreshold neuromorphic hardware is the creation of memory cells in RNNs. In this regard, we presented a solution that makes use of the resulting inhomogeneities for approximating the memory cells in RNNs. Furthermore, we introduced an approach for utilizing temporal coded neurons with an oscillating reference signal. The purpose of this approach is to produce a lossless conversion process that results in less than one spike per neuron.

Regarding the conversion of ANN to SNN, we evaluated the performance of different encoding schemes. We demonstrated that the state-of-the-art conversion approach based on rate encodings can be optimized for increased inference time. Moreover, we proposed an approach to estimate the accuracy of the converted SNN from the original ANN, which can aid in the development process.

In the context of subthreshold neuromorphic hardware, we addressed the challenges related to creating memory cells in RNNs. Our approach leverages the inhomogeneities in the hardware for approximating the memory cells. This solution offers an efficient and effective way to overcome the limitations of subthreshold neuromorphic hardware.

Finally, we introduced a novel approach for using temporal coded neurons with an oscillating reference signal. This approach offers a lossless conversion process that results in less than one spike per neuron.

8 Outlook

In this thesis, we have presented and evaluated various methods and approaches for converting ANNs to SNNs that can be used to simplify the development of neuromorphic hardware. Rate-coded network hyperparameters can be determined and evaluated prior to conversion to save time and computational power, resulting in a near-lossless conversion with a very high inference speed suitable for use with available low-powered neuromorphic hardware.

Furthermore, we have demonstrated an effective method for circumventing manufacturing inaccuracies that arise during the development of subthreshold analog neuromorphic hardware. We have leveraged the resulting inhomogeneities to our advantage for processing sequential data, resulting in enhanced data processing efficiency.

Finally, we have showcased how implementing a reference oscillation in neuromorphic hardware can lead to information processing with less than a single spike per neuron. Our results indicate the potential to achieve unprecedented levels of energy efficiency.

8.1 Rapid Prototyping of Rate-Coded Spiking Networks

Conversion approaches for generating SNNs have become a popular choice in the neuromorphic hardware field. Although common datasets are designed for ANNs, they typically require additional processing steps for usage within spiking networks. ANNs can be trained using many tools, and their usage is prevalent among data scientists. However, converting the existing networks to SNNs via a direct method can boost neuromorphic hardware adoption.

Our approach proposes a novel method for converting ANNs to SNNs, providing the added benefit of estimating the accuracy of the converted network before conversion and speeding up the development time of SNNs. We offer an open-sourced Python toolbox that integrates with the Machine Learning (ML) framework TensorFlow, facilitating the process for trained ML engineers. However, as more and more graphical approaches are developed for training ANNs, which can be utilized by individuals with little to no programming background, we suggest extending our approach with a Graphical User Interface (GUI) to facilitate the process.

The GUI will offer a more user-friendly interface, enabling individuals without a programming background to easily convert ANNs to SNNs. In addition, the GUI will democratize the process of converting ANNs to SNNs, making it more accessible to a broader range of individuals. We anticipate this approach will encourage more individuals to adopt neuromorphic hardware and further enhance the development of SNNs.

8.2 Population-Coded Converted Networks

The development of subthreshold analog neuromorphic hardware has the highest potential in terms of energy efficiency. Silicon transistors have demonstrated similar behavior to biological neurons when operated in subthreshold mode, making it possible to incorporate many neurons on a single chip using existing manufacturing processes. However, inhomogeneities of the silicon transistors can lead to inconsistency in the spiking threshold of the neurons, resulting from device mismatch due to process variance and shot and thermal noise. The creation of a uniform spiking threshold has emerged as a major challenge in the development of subthreshold analog neuromorphic hardware.

Our approach aims to leverage less-than-perfect hardware developments in implementing converted NNs with populations of spiking neurons. While S-shaped activation functions such as Sigmoid and Tanh are no longer as popular as they once were due to the increased implementation of ReLU, they still serve as a crucial component of memory cells in RNNs such as LSTM and Gated Recurrent Unit (GRU). Despite the increasing popularity of transformer networks, RNNs are still commonly used for data with an unknown input length, such as in voice assistants or text-to-speech applications. These fields would significantly benefit from the implementation of subthreshold analog neuromorphic hardware because of its energy-efficient nature, especially considering their usual usage in low-powered environments like smartphones or in cases where faster offline processing to bypass additional time requirements incurred from uploading and downloading data to a server would benefit the user experience.

Given the points addressed, we would recommend a constrain-then-train approach that effectively utilizes the distribution of subthreshold neuron thresholds as activation functions in the original ANN to reduce conversion loss.

8.3 Temporal-Coded Converted Networks

Among the various coding approaches in neuroscience, temporal coding holds the most potential for energy efficiency when compared to rate and population coding strategies, as it encodes information in the absence of spikes, rather than by counting the number of spikes, thus minimizing energy consumption. Given the benefits of rapid prototyping and energy-efficient neuromorphic hardware, a promising next step is to develop an efficient implementation of temporal coding for maximum energy efficiency.

Our analysis of different encoding schemes has revealed that only time-to-first-spike and phase coding are viable options for effective temporal coding of converted spiking networks, yet only the latter has the potential to be scaled to deep NNs. However, phase coding requires a global reference oscillation to function, which is currently not implemented in available neuromorphic hardware.

Developing a reference oscillation in different types of neuromorphic hardware could enable the use of very sparsely activated converted SNNs, offering further energy savings. Our proposed approach has already been implemented in follow-up research with the use on neuromorphic research hardware [225].

Bibliography

[1] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[2] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," 2011.

[3] A. Esteva, B. Kuprel, R. A. Novoa, *et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[4] R. H. Davies, J. B. Augusto, A. Bhuva, *et al.*, "Precision measurement of cardiac structure and function in cardiovascular magnetic resonance using machine learning," *Journal of Cardiovascular Magnetic Resonance*, vol. 24, no. 1, p. 16, 2022.

[5] L. Geex, *Comparing the performance of artificial intelligence to human lawyers in the review of standard business contracts*, `https://images.law.com/contrib/content/uploads/documents/397/5408/lawgeex.pdf`, 2018.

[6] Q. Le and B. Zoph, *Using machine learning to explore neural network architecture*, 2017.

[7] D. Silver and D. Hassabis, *AlphaGo: Mastering the ancient game of Go with Machine Learning*, 2016.

[8] J. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[9] Deci, *An Overview of State of the Art (SOTA) DNNs*, https://deci.ai/blog/sota-dnns-overview/, 2022.

[10] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," 2017.

[11] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. S. Schoenholz, and J. Pennington, *Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks*, 2018. arXiv: `1806.05393`.

[12] T. B. Brown, B. Mann, N. Ryder, *et al.*, “Language models are few-shot learners,” *arXiv:2005.14165 [cs]*, 2020. arXiv: `2005.14165 [cs]`.

[13] L. F. W. Anthony, B. Kanding, and R. Selvan, *Carbontracker: Tracking and predicting the carbon footprint of training deep learning models*, 2020. arXiv: `2007.03051`.

[14] D. So, Q. Le, and C. Liang, “The evolved Transformer,” in *Proceedings of the 36th International Conference on Machine Learning*, PMLR, 2019, pp. 5877–5886.

[15] E. Strubell, A. Ganesh, and A. McCallum, *Energy and policy considerations for deep learning in NLP*, 2019. arXiv: `1906.02243`.

[16] D. Amodei and D. Hernandez, *AI and compute*, https://openai.com/blog/ai-and-compute/, 2018.

[17] J. Koomey, *Our latest on energy efficiency of computing over time, now out in Electronic Design*, https://www.koomey.com/post/153838038643, 2016.

[18] Y. LeCun, J. Denker, and S. Solla, “Optimal brain damage,” in *Advances in Neural Information Processing Systems*, vol. 2, Morgan-Kaufmann, 1989.

[19] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[20] J. Frankle and M. Carbin, “The Lottery Ticket Hypothesis: Finding sparse, trainable neural networks,” *arXiv:1803.03635 [cs]*, 2019. arXiv: `1803.03635 [cs]`.

[21] A. G. Howard, M. Zhu, B. Chen, *et al.*, *MobileNets: Efficient convolutional neural networks for mobile vision applications*, 2017. arXiv: `1704.04861`.

[22] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An extremely efficient convolutional neural network for mobile devices,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[23] A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.

[24] A. Shafiee, A. Nag, N. Muralimanohar, *et al.*, “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” p. 13, 2016.

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

[26] C.-S. Poon and K. Zhou, “Neuromorphic silicon neurons and large-scale neural networks: Challenges and opportunities,” *Frontiers in Neuroscience*, vol. 5, 2011.

[27] M. Davies, N. Srinivasa, T.-H. Lin, *et al.*, “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[28] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface — Science,” *Science*, vol. 345, no. 6197, 2014.

[29] Mikroelektronikforschung, *KI-ASIC*, `https://www.elektronikforschung.de/projekte/ki-asic`, 2019.

[30] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[31] E. Izhikevich, “Which model to use for cortical spiking neurons?” *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[32] W. Maass, “Networks of spiking neurons: The third generation of neural network models,” *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[33] G.-q. Bi and M.-m. Poo, “Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type,” *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[34] S. Song, K. D. Miller, and L. F. Abbott, “Competitive Hebbian learning through spike-timing-dependent synaptic plasticity,” *Nature Neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[35] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, “Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning,” *Neural Computation*, vol. 18, no. 6, pp. 1318–1348, 2006.

[36] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, “Event-driven contrastive divergence for spiking neuromorphic systems,” *Frontiers in Neuroscience*, vol. 7, 2014.

[37] E. O. Neftci, H. Mostafa, and F. Zenke, “Surrogate gradient learning in spiking neural networks,” *arXiv:1901.09948 [cs, q-bio]*, 2019. arXiv: `1901.09948 [cs, q-bio]`.

[38] B. Rueckauer and S.-C. Liu, “Conversion of analog to spiking neural networks using sparse temporal coding,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[39] E. Mueller, D. Auge, and A. Knoll, “Normalization hyperparameter search for converted spiking neural networks,” 2021, p. 2.

[40] E. Mueller, D. Auge, and A. Knoll, “Exploiting inhomogeneities of subthreshold transistors as populations of spiking neurons,” 2022, p. 10.

[41] D. Auge, J. Hille, E. Mueller, and A. Knoll, "A survey of encoding techniques for signal processing in spiking neural networks," *Neural Processing Letters*, vol. 53, no. 6, pp. 4693–4710, 2021.

[42] E. Mueller, J. Hansjakob, and D. Auge, "Faster conversion of analog to spiking neural networks by error centering," in *Bernstein Conference 2020*, 2020, p. 2.

[43] E. Mueller, J. Hansjakob, D. Auge, and A. Knoll, "Minimizing inference time: Optimization methods for converted deep spiking neural networks," in *International Joint Conference on Neural Network*, 2021, p. 8.

[44] E. Mueller, V. Studenyak, D. Auge, and A. Knoll, "Spiking Transformer networks: A rate coded approach for processing sequential data," in *2021 7th International Conference on Systems and Informatics (ICSAI)*, Chongqing, China: IEEE, 2021, pp. 1–5.

[45] E. Mueller, D. Auge, S. Klimaschka, and A. Knoll, "Neural oscillations for energy-efficient hardware implementation of sparsely activated deep spiking neural networks," 2022, p. 7.

[46] E. Mueller, *Convert2SNN: Toolbox for the ANN-to-SNN Conversion*, `https://github.com/EtienneMueller/Convert2SNN`, 2021.

[47] D. Auge and E. Mueller, "Resonate-and-fire neurons as frequency selective input encoders for spiking neural networks," Tech. Rep., 2020, p. 8.

[48] D. Auge, J. Hille, and E. Mueller, "End-to-end spiking neural network for speech recognition using resonating input neurons," in *30th International Conference on Artificial Neural Networks (ICANN)*, 2021, p. 12.

[49] D. Auge, J. Hille, E. Mueller, and A. Knoll, "Hand gesture recognition in range-doppler images using binary activated spiking neural networks," in *2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021)*, Jodhpur, India: IEEE, 2021, pp. 01–07.

[50] D. Auge, P. Wenner, and E. Mueller, "Hand gesture recognition using hierarchical temporal memory on radar sequence data," in *Bernstein Conference 2020*, 2020, p. 2.

[51] V. Nair and G. E. Hinton, "Rectified Linear Units improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Conference on Machine Learning*, 2010, p. 8.

[52] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," 1952.

[53] C. Morris and H. Lecar, "Voltage oscillations in the barnacle giant muscle fiber," *Biophysical Journal*, vol. 35, no. 1, pp. 193–213, 1981.

[54] R. Fitzhugh, “Impulses and physiological states in theoretical models of nerve membrane,” *Biophysical Journal*, vol. 1, no. 6, pp. 445–466, 1961.

[55] J. Nagumo, S. Arimoto, and S. Yoshizawa, “An Active Pulse Transmission Line Simulating Nerve Axon,” *Proceedings of the IRE*, vol. 50, no. 10, pp. 2061–2070, 1962.

[56] J. L. Hindmarsh and R. M. Rose, “A Model of Neuronal Bursting Using Three Coupled First Order Differential Equations,” *Proceedings of the Royal Society of London. Series B, Biological Sciences*, vol. 221, no. 1222, pp. 87–102, 1984. JSTOR: 35900.

[57] E. M. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[58] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, 2002.

[59] P. E. Latham, B. J. Richmond, P. G. Nelson, and S. Nirenberg, “Intrinsic Dynamics in Neuronal Networks. I. Theory,” *Journal of Neurophysiology*, vol. 83, no. 2, pp. 808–827, 2000.

[60] R. Brette and W. Gerstner, “Adaptive exponential integrate-and-fire model as an effective description of neuronal activity,” *Journal of Neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.

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

[62] Ş. Mihalaş and E. Niebur, “A Generalized Linear Integrate-and-Fire Neural Model Produces Diverse Spiking Behaviors,” *Neural Computation*, vol. 21, no. 3, pp. 704–718, 2009.

[63] F. Grassia, T. Levi, T. Kohno, and S. Saïghi, “Silicon neuron: Digital hardware implementation of the quartic model,” *Artificial Life and Robotics*, vol. 19, no. 3, pp. 215–219, 2014.

[64] L. Abbott, “Lapicque’s introduction of the integrate-and-fire model neuron (1907),” *Brain Research Bulletin*, vol. 50, no. 5-6, pp. 303–304, 1999.

[65] D. Monroe, “Neuromorphic Computing Gets Ready for the (Really) Big Time,” *Commun. ACM*, vol. 57, no. 6, pp. 13–15, 2014.

[66] N. Izeboudjen, C. Larbes, and A. Farah, “A new classification approach for neural networks hardware: From standards chips to embedded systems on chip,” *Artificial Intelligence Review*, vol. 41, no. 4, pp. 491–534, 2014.

[67] C. D. Schuman, T. E. Potok, R. M. Patton, *et al.*, “A survey of neuromorphic computing and neural networks in hardware,” *arXiv:1705.06963 [cs]*, 2017. arXiv: `1705.06963 [cs]`.

[68] G. Indiveri, “Computation in Neuromorphic Analog VLSI Systems,” in *Neural Nets WIRN Vietri-01*, R. Tagliaferri and M. Marinaro, Eds., ser. Perspectives in Neural Computing, London: Springer, 2002, pp. 3–20.

[69] C. Mayr, S. Hoeppner, and S. Furber, *SpiNNaker 2: A 10 million core processor system for brain simulation and machine learning*, 2019. arXiv: `1911.02385`.

[70] *Human Brain Project*, `https://www.humanbrainproject.eu/en/`.

[71] *Brain Initiative*, `https://braininitiative.nih.gov/`.

[72] N. T. Carnevale and M. L. Hines, *The NEURON Book*. Cambridge University Press, 2006.

[73] J. M. Bower and D. Beeman, *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*. Springer Science & Business Media, 2012.

[74] M. Stimberg, R. Brette, and D. F. Goodman, “Brian 2, an intuitive and efficient neural simulator,” *eLife*, vol. 8, F. K. Skinner, R. L. Calabrese, F. K. Skinner, F. Zeldenrust, and R. C. Gerkin, Eds., e47314, 2019.

[75] M. Diesmann and M.-O. Gewaltig, “NEST: An Environment for Neural Systems Simulations,” *Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis*, vol. 58, pp. 43–70, 2001.

[76] R. A. Tikidji-Hamburyan, V. Narayana, Z. Bozkus, and T. A. El-Ghazawi, “Software for brain network simulations: A comparative study,” *Frontiers in Neuroinformatics*, vol. 11, 2017.

[77] E. Falotico, L. Vannucci, A. Ambrosano, *et al.*, “Connecting artificial brains to robots in a comprehensive simulation framework: The neurorobotics platform,” *Frontiers in Neurorobotics*, vol. 11, 2017.

[78] M. Abadi, P. Barham, J. Chen, *et al.*, “TensorFlow: A system for large-scale machine learning,” in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[79] A. Paszke, S. Gross, F. Massa, *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[80] A. Cassidy, A. G. Andreou, and J. Georgiou, “Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis,” in *2011 45th Annual Conference on Information Sciences and Systems*, 2011, pp. 1–6.

[81] Q. Wang, Y. Li, B. Shao, S. Dey, and P. Li, "Energy efficient parallel neuromorphic architectures with approximate arithmetic on FPGA," *Neurocomputing*, vol. 221, pp. 146–158, 2017.

[82] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio, and L. Raffo, "An FPGA platform for real-time simulation of spiking neuronal networks," *Frontiers in Neuroscience*, vol. 11, 2017.

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

[84] R. M. Wang, C. S. Thakur, and A. van Schaik, "An FPGA-based massively parallel neuromorphic cortex simulator," *Frontiers in Neuroscience*, vol. 12, 2018.

[85] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[86] J. Shen, D. Ma, Z. Gu, *et al.*, "Darwin: A neuromorphic hardware co-processor based on Spiking Neural Networks," *Science China Information Sciences*, vol. 59, no. 2, pp. 1–5, 2016.

[87] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Transactions on Biomedical Circuits and Systems*, pp. 1–1, 2018.

[88] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, 2018.

[89] S.-G. Cho, E. Beigné, and Z. Zhang, "A 2048-neuron spiking neural network accelerator with neuro-inspired pruning and asynchronous network on chip in 40nm CMOS," in *2019 IEEE Custom Integrated Circuits Conference (CICC)*, 2019, pp. 1–4.

[90] J. Park, J. Lee, and D. Jeon, "7.6 A 65nm 236.5nJ/classification neuromorphic processor with 7.5 percent energy overhead on-chip learning using direct spike-only feedback," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 140–142.

[91] J. Pei, L. Deng, S. Song, *et al.*, "Towards artificial general intelligence with hybrid Tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.

[92] J. Stuijt, M. Sifalakis, A. Yousefzadeh, and F. Corradi, "μBrain: An event-driven and fully synthesizable architecture for spiking neural networks," *Frontiers in Neuroscience*, Neuromorphic Engineering, 2021.

[93] Y. Kuang, X. Cui, Y. Zhong, *et al.*, "A 28-nm 0.34-pJ/SOP spike-based neuromorphic processor for efficient artificial neural network implementations," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[94] Y. Kuang, X. Cui, Y. Zhong, *et al.*, "A 64K-neuron 64M-1b-synapse 2.64pJ/SOP neuromorphic chip with all memory on chip for spike-based models in 65nm CMOS," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 7, pp. 2655–2659, 2021.

[95] Y. Zhong, X. Cui, Y. Kuang, K. Liu, Y. Wang, and R. Huang, "A spike-event-based neuromorphic processor with enhanced on-chip STDP learning in 28nm CMOS," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[96] G. Orchard, E. P. Frady, D. B. D. Rubin, *et al.*, "Efficient neuromorphic signal processing with Loihi 2," in *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, 2021, pp. 254–259.

[97] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 1947–1950.

[98] S. A. Aamir, P. Müller, A. Hartel, J. Schemmel, and K. Meier, "A highly tunable 65-nm CMOS LIF neuron for a large scale neuromorphic system," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016, pp. 71–74.

[99] B. V. Benjamin, P. Gao, E. McQuinn, *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[100] A. Neckar, S. Fok, B. V. Benjamin, *et al.*, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2018.

[101] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, 2017.

[102] N. Qiao, H. Mostafa, F. Corradi, *et al.*, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Frontiers in Neuroscience*, vol. 9, 2015.

[103] F. N. Buhler, P. Brown, J. Li, T. Chen, Z. Zhang, and M. P. Flynn, "A 3.43 TOPS/W 48.9 pJ/pixel 50.1 nJ/classification 512 analog neuron sparse coding neural network with on-chip learning and classification in 40nm CMOS," in *2017 Symposium on VLSI Circuits*, 2017, pp. C30–C31.

[104] E. Farquhar, C. Gordon, and P. Hasler, "A field programmable neural array," in *2006 IEEE International Symposium on Circuits and Systems*, 2006, 4 pp.–4117.

[105] M. Liu, H. Yu, and W. Wang, "FPAA Based on Integration of CMOS and Nanojunction Devices for Neuromorphic Applications," in *Nano-Net*, M. Cheng, Ed., vol. 3, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 44–48.

[106] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, *Event-Based Neuromorphic Systems.* John Wiley & Sons, 2014.

[107] S.-C. Liu, J. Kramer, G. Indiveri, T. Delbruck, and R. Douglas, *Analog VLSI: Circuits and Principles.* MIT Press, 2002.

[108] E. Chicca, D. Badoni, V. Dante, *et al.*, "A VLSI recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term memory," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1297–1307, 2003.

[109] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems* (Computational Neuroscience). Cambridge, Mass: MIT Press, 2003.

[110] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs," *Science*, vol. 275, no. 5297, pp. 213–215, 1997.

[111] S. M. Bohte, "Error-backpropagation in networks of fractionally predictive spiking neurons," in *Artificial Neural Networks and Machine Learning – ICANN 2011*, T. Honkela, W. Duch, M. Girolami, and S. Kaski, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2011, pp. 60–68.

[112] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short term memory and learning-to-learn in networks of spiking neurons," *arXiv:1803.09574 [cs, q-bio]*, 2018. arXiv: `1803.09574 [cs, q-bio]`.

[113] F. Zenke and S. Ganguli, "SuperSpike: Supervised learning in multi-layer spiking neural networks," *Neural Computation*, vol. 30, no. 6, pp. 1514–1541, 2018.

[114] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory.* New York: Psychology Press, 1949.

[115] N. T. Markov, J. Vezoli, P. Chameau, *et al.*, “Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex,” *Journal of Comparative Neurology*, vol. 522, no. 1, pp. 225–259, 2014.

[116] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, “Random synaptic feedback weights support error backpropagation for deep learning,” *Nature Communications*, vol. 7, no. 1, p. 13 276, 2016.

[117] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, “Event-driven random back-propagation: Enabling neuromorphic deep learning machines,” *Frontiers in Neuroscience*, vol. 11, 2017.

[118] J. Guerguiev, T. P. Lillicrap, and B. A. Richards, “Towards deep learning with segregated dendrites,” *eLife*, vol. 6, e22901, 2017.

[119] T. Masquelier, R. Guyonneau, and S. J. Thorpe, “Competitive STDP-based spike pattern learning,” *Neural Computation*, vol. 21, no. 5, pp. 1259–1276, 2009.

[120] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass, “Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity,” 2013.

[121] A. Coates, A. Ng, and H. Lee, “An analysis of single-layer networks in unsupervised feature learning,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 215–223.

[122] A. Dundar, J. Jin, and E. Culurciello, *Convolutional clustering for unsupervised learning*, 2016. arXiv: `1511.06241 [cs]`.

[123] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, “Stochastic synapses enable efficient brain-inspired learning machines,” *Frontiers in Neuroscience*, vol. 10, 2016.

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

[125] C. Lee, G. Srinivasan, P. Panda, and K. Roy, “Deep spiking convolutional neural network trained with unsupervised spike-timing-dependent plasticity,” *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 3, pp. 384–394, 2019.

[126] P. Panda and K. Roy, “Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition,” *arXiv:1602.01510 [cs]*, 2016. arXiv: `1602.01510 [cs]`.

[127] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data," *Frontiers in Neuroscience*, vol. 11, 2017.

[128] 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. arXiv: `1804.00227 [cs, q-bio]`.

[129] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[130] R. Gütig and H. Sompolinsky, "The tempotron: A neuron that learns spike timing-based decisions," *Nature neuroscience*, vol. 9, pp. 420–428, 2006.

[131] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.

[132] B. Schrauwen and J. V. Campenhout, "Improving SpikeProp: Enhancements to an error-backpropagation rule for spiking neural networks," *Proceedings of the 15th ProRISC workshop*, vol. 11, 2004.

[133] S. McKennoch, D. Liu, and L. Bushnell, "Fast modifications of the SpikeProp algorithm," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 2006, pp. 3970–3977.

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

[135] P. O'Connor and M. Welling, *Deep Spiking Networks*, 2016. arXiv: `1602.08323 [cs]`.

[136] H. Mostafa, "Supervised learning based on temporal coding in Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, 2018.

[137] Y. Jin, W. Zhang, and P. Li, "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018.

[138] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, *et al.*, "CAVIAR: A 45k neuron, 5M synapse, 12G connects\/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking," *IEEE Transactions on Neural Networks*, vol. 20, no. 9, pp. 1417–1438, 2009.

[139] J. A. Perez-Carrasco, B. Zhao, C. Serrano, *et al.*, “Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing-application to feedforward ConvNets,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2706–2719, 2013.

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

[141] Y. Hu, H. Tang, Y. Wang, and G. Pan, “Spiking deep residual network,” *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[142] S. Kim, S. Park, B. Na, and S. Yoon, “Spiking-YOLO: Spiking neural network for energy-efficient object detection,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 11 270–11 277, 2019.

[143] Y. Cao, Y. Chen, and D. Khosla, “Spiking deep convolutional neural networks for energy-efficient object recognition,” *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2014.

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

[145] A. J. Yu, M. A. Giese, and T. A. Poggio, “Biophysiologically plausible implementations of the maximum operation,” *Neural Computation*, vol. 14, no. 12, pp. 2857–2881, 2002.

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

[147] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, “Going deeper in spiking neural networks: VGG and residual architectures,” *Frontiers in Neuroscience*, vol. 13, 2019.

[148] Y. Li, D. Zhao, and Y. Zeng, “BSNN: Towards faster and better conversion of artificial neural networks to spiking neural networks with bistable neurons,” *Frontiers in Neuroscience*, vol. 16, 2022.

[149] G. Orchard, X. Lagorce, C. Posch, S. B. Furber, R. Benosman, and F. Galluppi, “Real-time event-driven spiking neural network object recognition on the SpiNNaker platform,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 2413–2416.

[150] M. Riesenhuber and T. Poggio, “Hierarchical models of object recognition in cortex,” *Nature Neuroscience*, vol. 2, no. 11, pp. 1019–1025, 1999.

[151] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, “HATS: Histograms of averaged time surfaces for robust event-based object classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1731–1740.

[152] D. Zambrano and S. M. Bohte, “Fast and efficient asynchronous neural computation with adapting spiking neural networks,” *arXiv:1609.02053 [cs]*, 2016. arXiv: `1609.02053 [cs]`.

[153] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, “Backpropagation for energy-efficient neuromorphic computing,” in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 1117–1125.

[154] S. K. Esser, P. A. Merolla, J. V. Arthur, *et al.*, “Convolutional networks for fast, energy-efficient neuromorphic computing,” *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11 441–11 446, 2016.

[155] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[156] S. Davidson and S. B. Furber, “Comparison of Artificial and Spiking Neural Networks on Digital Hardware,” *Frontiers in Neuroscience*, vol. 15, 2021.

[157] S. Denève and C. K. Machens, “Efficient codes and balanced networks,” *Nature Neuroscience*, vol. 19, no. 3, pp. 375–382, 2016.

[158] S. Thorpe and J. Gautrais, “Rank Order Coding,” in *Computational Neuroscience: Trends in Research, 1998*, J. M. Bower, Ed., Boston, MA: Springer US, 1998, pp. 113–118.

[159] R. S. Johansson and I. Birznieks, “First spikes in ensembles of human tactile afferents code complex spatial fingertip events,” *Nature Neuroscience*, vol. 7, no. 2, pp. 170–177, 2004.

[160] T. Gollisch and M. Meister, “Rapid neural coding in the retina with relative spike latencies,” *Science*, vol. 319, no. 5866, pp. 1108–1111, 2008.

[161] J. J. Hopfield, “Pattern recognition computation using action potential timing for stimulus representation,” *Nature*, vol. 376, no. 6535, pp. 33–36, 1995.

[162] C. Kayser, M. A. Montemurro, N. K. Logothetis, and S. Panzeri, “Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns,” *Neuron*, vol. 61, no. 4, pp. 597–608, 2009.

[163] C. M. Gray, P. König, A. K. Engel, and W. Singer, "Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties," *Nature*, vol. 338, no. 6213, pp. 334–337, 1989.

[164] J. Gautrais and S. Thorpe, "Rate coding versus temporal order coding: A theoretical approach," *Biosystems*, vol. 48, no. 1, pp. 57–65, 1998.

[165] M. Zhang, N. Zheng, D. Ma, G. Pan, and Z. Gu, "Efficient spiking neural networks with logarithmic temporal coding," *arXiv:1811.04233 [cs]*, 2018. arXiv: `1811.04233 [cs]`.

[166] H. Hamanaka, H. Torikai, and T. Saito, "Quantized spiking neuron with A/D conversion functions," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 10, pp. 1049–1053, 2006.

[167] F. Ponulak and A. Kasiński, "Introduction to spiking neural networks: Information processing, learning and applications," p. 26, 2011.

[168] A.-M. M. Oswald, B. Doiron, and L. Maler, "Interval coding. I. Burst Interspike Intervals as indicators of stimulus intensity," *Journal of Neurophysiology*, vol. 97, no. 4, pp. 2731–2743, 2007.

[169] M. Li and J. Z. Tsien, "Neural Code—Neural Self-information Theory on how cell-assembly code rises from spike time and neuronal variability," *Frontiers in Cellular Neuroscience*, vol. 11, p. 236, 2017.

[170] S. Park, S. Kim, H. Choe, and S. Yoon, "Fast and efficient information transmission with burst spikes in deep spiking neural networks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, Las Vegas NV USA: ACM, 2019, pp. 1–6.

[171] L. Turnbull, E. Dian, and G. Gross, "The string method of burst identification in neuronal spike trains," *Journal of Neuroscience Methods*, vol. 145, no. 1, pp. 23–35, 2005.

[172] F. Zeldenrust, W. J. Wadman, and B. Englitz, "Neural coding with bursts—current state and future perspectives," *Frontiers in Computational Neuroscience*, vol. 12, 2018.

[173] S. Ahmad and L. Scheinkman, "How can we be so dense? The benefits of using highly sparse representations," 2019.

[174] B. A. Olshausen and D. J. Field, "Sparse coding of sensory inputs," *Current Opinion in Neurobiology*, vol. 14, no. 4, pp. 481–487, 2004.

[175] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers in Neural Circuits*, vol. 10, 2016.

[176] P. N. Steinmetz, A. Roy, P. J. Fitzgerald, S. S. Hsiao, K. O. Johnson, and E. Niebur, "Attention modulates synchronized neuronal firing in primate somatosensory cortex," *Nature*, vol. 404, no. 6774, pp. 187–190, 2000.

[177] C. M. Gray and W. Singer, "Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex.," *Proceedings of the National Academy of Sciences*, vol. 86, no. 5, pp. 1698–1702, 1989.

[178] E. I. Moser, E. Kropff, and M.-B. Moser, "Place cells, grid cells, and the brain's spatial representation system," *Annual Review of Neuroscience*, vol. 31, no. 1, pp. 69–89, 2008.

[179] J. O'Keefe and J. Dostrovsky, "The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat," *Brain Research*, vol. 34, pp. 171–175, 1971.

[180] M. Hough, H. de Garis, M. Korkin, F. Gers, and N. E. Nawa, "SPIKER: Analog waveform to digital spiketrain conversion in ATR's artificial brain (cam-brain) project," *In International conference on robotics and artificial life*, vol. 92, p. 4, 1999.

[181] B. Schrauwen and J. Van Campenhout, "BSA, a fast and accurate spike train encoding scheme," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 4, 2003, 2825–2830 vol.4.

[182] N. Sengupta, N. Scott, and N. Kasabov, "Framework for knowledge driven optimisation based data encoding for brain data modelling using spiking neural network architecture," in *Proceedings of the Fifth International Conference on Fuzzy and Neuro Computing (FANCCO - 2015)*, V. Ravi, B. K. Panigrahi, S. Das, and P. N. Suganthan, Eds., ser. Advances in Intelligent Systems and Computing, Cham: Springer International Publishing, 2015, pp. 109–118.

[183] N. Kasabov, N. M. Scott, E. Tu, *et al.*, "Evolving spatio-temporal data machines based on the NeuCube neuromorphic framework: Design methodology and selected applications," *Neural Networks*, Special Issue on "Neural Network Learning in Big Data", vol. 78, pp. 1–14, 2016.

[184] B. Petro, N. Kasabov, and R. M. Kiss, "Selection and optimization of temporal spike encoding methods for spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 2, pp. 358–370, 2020.

[185] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[186] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

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

[188] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.Sc. Thesis, University of Toronto, Department of Computer Science, 2009.

[189] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[190] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft COCO: Common Objects in Context," *arXiv:1405.0312 [cs]*, 2015. arXiv: `1405.0312 [cs]`.

[191] B. N. Esi Nyarko, W. Bin, J. Zhou, G. K. Agordzo, J. Odoom, and E. Koukoyi, "Comparative Analysis of AlexNet, Resnet-50, and Inception-V3 Models on Masked Face Recognition," in *2022 IEEE World AI IoT Congress (AIIoT)*, 2022, pp. 337–343.

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

[193] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1492–1500.

[194] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.

[195] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," *arXiv:1602.07261 [cs]*, 2016. arXiv: `1602.07261 [cs]`.

[196] A. Howard, M. Sandler, G. Chu, *et al.*, "Searching for MobileNetV3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[197] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *arXiv:1905.11946 [cs, stat]*, 2019. arXiv: `1905.11946 [cs, stat]`.

[198] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2980–2988.

[199] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[200] W. Liu, D. Anguelov, D. Erhan, *et al.*, "SSD: Single shot multibox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 21–37.

[201] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7263–7271.

[202] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint*, vol. arXiv:1804.02767 [cs], 2018.

[203] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 781–10 790.

[204] E. Kubera, A. Kubik-Komar, P. Kurasiński, K. Piotrowska-Weryszko, and M. Skrzypiec, "Detection and Recognition of Pollen Grains in Multilabel Microscopic Images," *Sensors*, vol. 22, p. 2690, 2022.

[205] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[206] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv:2010.11929 [cs]*, 2021. arXiv: `2010.11929 [cs]`.

[207] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv:1502.03167 [cs]*, 2015. arXiv: `1502.03167 [cs]`.

[208] B. Nessler, M. Pfeiffer, and W. Maass, "STDP enables spiking neurons to detect hidden causes of their inputs," *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds., pp. 1357–1365, 2009.

[209] P. U. Diehl, B. U. Pedroni, A. Cassidy, P. Merolla, E. Neftci, and G. Zarrella, "TrueHappiness: Neuromorphic emotion recognition on TrueNorth," in *2016 International Joint Conference on Neural Networks (IJCNN)*, Vancouver, BC, Canada: IEEE, 2016, pp. 4278–4285.

[210] A. S. Cassidy, P. Merolla, J. V. Arthur, *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–10.

[211] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, 2016, pp. 2818–2826.

[212] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: `1409.1556`.

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

[214] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," *Proceedings of the AISTATS*, 2015.

[215] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014.

[216] W. Xiao, H. Chen, Q. Liao, and T. Poggio, "Biologically-plausible learning algorithms can scale to large datasets," *arXiv:1811.03567 [cs, stat]*, 2018. arXiv: `1811.03567 [cs, stat]`.

[217] J. Wu, C. Xu, D. Zhou, H. Li, and K. C. Tan, "Progressive Tandem Learning for pattern recognition with deep spiking neural networks," *arxiv:2007.01204 [cs]*, 2020. arXiv: `2007.01204 [cs]`.

[218] E. Hunsberger and C. Eliasmith, "Spiking deep networks with LIF neurons," *arXiv:1510.08829 [cs]*, 2015. arXiv: `1510.08829 [cs]`.

[219] B. Han, G. Srinivasan, and K. Roy, "RMP-SNN: Residual Membrane Potential Neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 558–13 567.

[220] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and tools for the conversion of analog to spiking convolutional neural networks," *arXiv:1612.04052 [cs, stat]*, 2016. arXiv: `1612.04052 [cs, stat]`.

[221] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 2011, p. 9.

[222] P. Warden, "Speech Commands: A dataset for limited-vocabulary speech recognition," *arXiv:1804.03209 [cs]*, 2018. arXiv: `1804.03209 [cs]`.

[223] P. Blouw and C. Eliasmith, "Event-driven signal processing with neuromorphic computing systems," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain: IEEE, 2020, pp. 8534–8538.

[224] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.

[225] J. Lopez-Randulfe, N. Reeb, N. Karimi, *et al.*, "Time-coded spiking fourier transform in neuromorphic hardware," *arXiv:2202.12650 [cs, eess]*, 2022. arXiv: `2202.12650 [cs, eess]`.