

École polytechnique de Louvain

Training ultra-low-power spiking neural networks for neuromorphic IoT vision sensing and recognition

Author: **Thibault PIRSON**

Supervisors: **David BOL, Charlotte FRENKEL**

Readers: **Christophe DE VLEESCHOUWER, Martin LEFEBVRE**

Academic year 2018–2019

Master [120] in Electrical Engineering

Abstract

Computer vision and **machine learning** benefit from a growing interest in solving complex problems where computers are expected to automatically learn from data [1]. Despite impressive performances [2, 3], these algorithms are often energy-intensive [2, 4, 5] and fall far short of the robustness and versatility of biological systems [6, 7, 8, 9, 10, 11, 12, 13]. Over the last decades, **computers have become more powerful** [14]. However, this exponential growth in technology scaling seems to come to an end [14, 15], calling for new computing systems and processing algorithms. Indeed, current von Neumann architectures [16] have inherent hardware limitations [10] preventing them from efficiently implementing general-purpose real-time autonomous and energy-efficient systems, unlike biological brains. This drives a paradigm shift towards alternative architectures that could bring silicon systems closer to the efficiency of biological ones.

To address these challenges, an ongoing research field called **neuromorphic engineering** aims at developing practical solutions by drawing inspiration from the brain [17, 18]. Although neuromorphic sensors and processors already exist [7, 18, 19, 20, 21, 22, 23, 24], conventional imaging algorithms *cannot* be directly mapped to these hardware platforms as their architecture is *fundamentally different* from von Neumann computers [3]. Consequently, new algorithms are needed to process data efficiently on neuromorphic platforms.

This thesis aims at investigating several ways of training spiking neural networks (SNNs) in order to classify accurately samples from real-world neuromorphic datasets. At the same time, neuromorphic processor(s) developed at UCLouvain come as a support hardware framework, which defines chip related constraints. The **main achievements** of this thesis range from the development of a Python spiking simulator for inference in the spiking domain, the generation of SSP-MNIST which is a synthetic spiking version of MNIST, to an adapted integrate-and-fire neuron model thought to improve the classification accuracy of SNNs. To the best of our knowledge, this was the first time that the proposed approach was applied to the real-world neuromorphic dataset N-CARS. Finally, even though the proposed approach *did not* outperform state-of-the-art techniques, it achieved good classification accuracy despite being subject to hardware constraints (e.g. 98.19% on SSP-MNIST, 96.83% on N-MNIST and 81.97% on N-CARS).

Acknowledgements

I would like to thank my supervisors, David Bol and Charlotte Frenkel for being always available, helpful, inspiring, for their valuable feedback and for sharing their interest and passion for their fields. Moreover, I would like to thank them for their open-mindedness regarding the organization and the progression of this work which ran currently with an exchange program at the University of Melbourne, Australia. A special thanks goes to Charlotte Frenkel who has shared her exciting work and *always* answered very kindly and precisely to a dense stream of questions throughout the realization of this master thesis.

I would like to thank Amos Sironi (PROPHESSEE) for his time and answers about the N-CARS dataset. In addition, he shared a rescaled and downsampled version of the original N-CARS dataset, which was easier to handle and used in this work.

Additionally, I would like to thank Brigitte Dupont and Mathieu Xhonneux for the help provided in order to log in and connect to the UCLouvain machine learning servers, which was really useful to speed up simulations.

Finally, a special acknowledgement goes to family and friends for their support and encouragement.

Nomenclature

Acronyms

ADAM	Adaptive Moment Estimation
AER	Address Event Representation
AE	Address Event
AI	Artificial Intelligence
AIF	Adapted Integrate and Fire (neuron model)
ANN	Artificial Neural Network
AP	Action Potential
ATIS	Asynchronous Time-based Imaging Sensor
AVG	Averaging/Average
BP	Backpropagation
CNN/ConvNet	Convolution Neural Network
Conv	Convolution
CPU	Central Processing Unit
DBN	Deep-Belief Network
DL	Deep Learning
DNN	Deep Neural Network
DVS	Dynamic Vision Sensor
EB	Event-Based
EPSP	Excitatory Post-Synaptic Potential
FB	Frame-Based
FC	Fully Connected
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
IF	Integrate-and-Fire
IoT	Internet-of-Things
IPSP	Inhibitory Post-Synaptic Potential
LCD	Liquid Crystal Display
LIF	Leaky Integrate-and-Fire
LSTM	Long Short-Term Memory
ML	Machine Learning

Acronyms

MNIST	Modified National Institute of Standards and Technology (dataset)
MORPHIC	Neuromorphic processor (UCLouvain)
MSE	Mean Square Error
N-CARS	Real-world neuromorphic dataset (PROPHESSEE)
N-MNIST	Neuromorphic-MNIST (dataset)
NN	Neural Networks
ODIN	ONline-learning DIgital spiking Neuromorphic processor (UCLouvain)
PCA	Principal Component Analysis
PSNR	Peak Signal to Noise Ratio
PSP	Post-Synaptic Potential
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
S-SDSP	Stochastic-SDSP
SDSP	Spike-Driven Synaptic Plasticity
SGD	Stochastic Gradient Descent
SNN	Spiking Neural Network
SOM	Self Organizing Maps
SSP-MNIST	Synthetic SPiking MNIST (dataset)
STDP	Spike-Timing-Dependent Plasticity
SVM	Support Vector Machines
VNB	von Neumann Bottleneck
WTA	Winner-Take-All

Symbol	Units	
T_{frame}	s	Time interval between two successive frames
$\tau_{reality}$	s	Time constant of changes in the observed scene
$w_{M,J}$	/	Synaptic weight between neurons M and J
I_i	/	Neuron input i
TF_J	/	Transfer Function of the neuron J
AF_J	/	Activation Function of the neuron J
y_J	/	Output of the neuron J
k_a, k_b, k_l, k_i	pixel	Kernel shape
A, B	pixel	Input shape
M, N	/	Number of channels/feature maps
H, C	/	Number of neurons in the hidden,output layer
$\mathbf{e} = [x, y, t, p]$	/	Event = [x-coordinate, y-coordinate, timestamp, polarity]
Na^+	/	Sodium ion
K^-	/	Potassium ion
Cl^-	/	Chlorine ion
Ca^{2+}	/	Calcium ion
$V(t)$	mV	Membrane potential at time t
L	mV	Leakage parameter
$X(t)$	mV	Summation of all synaptic inputs at time t ($X_J(t) = \sum_i w_{i,J} I_i(t)$)
θ_{thresh}	mV	Membrane firing threshold
$V_{resting}$	mV	Resting membrane potential
C_m	F	Membrane capacitance
R_m	Ohm	Membrane resistance
$I(t)$	A	Input current at time t
t_{ref}	ms	Refractory period
θ_{thresh}^{down}	mV	Upper bound membrane firing threshold
θ_{thresh}^{up}	mV	Lower bound membrane firing threshold
q	/	Number of Conv layers in the network
r	/	Number of FC layers in the network
g	/	Pixel index
p	/	Sample index
dt	ms	Accumulation time window
$s = \{t_i^{g,p}, \dots, t_n^{g,p}\}$	ms	Spike train for pixel p of sample p
$C_{dt}^{g,p}$	/	Spike count for pixel p of sample p , over an accumulation time window dt

Symbol	Units	
$\tilde{C}_{dt}^{g,p}$	/	Normalized spike count for pixel p of sample p , over an accumulation time window dt
t_{pre}	ms	Timestamp of the pre-synaptic spike
t_{post}	ms	Timestamp of the post-synaptic spike
Δ_t	ms	Time interval between the pre- and post- synaptic spike ($\Delta_t = t_{post} - t_{pre}$)
τ	ms	Time constant
$\theta_1, \theta_2, \theta_3$	/	Calcium thresholds
Θ_m	mV	Thresholds vector
l	/	Index of the layer in the network
$\theta_m^{layer_l}$	mV	Membrane firing threshold of neurons in layer l
β_1	/	Exponential decay rate for the first moment estimates (ADAM parameter)
β_2	/	Exponential decay rate for the second-moment estimates (ADAM parameter)
ϵ	/	Very small number to prevent any division by zero (ADAM parameter)
η	/	Learning rate
ζ, ξ	/	Parameters for learning rate
N_{cores}		Number of cores in the multi-cores spiking simulator
$N_{synapses}$	/	Number of synaptic weights in the Conv part
$N_{neurons}$	/	Number of neurons in the Conv part
$R_{synapses}$	bit	Resolution of the synaptic weights in the Conv part
$R_{neurons}$	bit	Resolution of the neurons in the Conv part

Contents

Abstract	i
Acknowledgements	iii
List of Symbols	vii
1 Introduction	1
1.1 Goals	2
1.2 Main achievements	2
1.3 Structure of this work	3
2 Fundamentals	5
2.1 Context and motivations	5
2.1.1 Frame-based processing	5
2.1.2 Basics of machine learning and computer vision	7
2.1.2.1 Objectives and applications	7
2.1.2.2 Architectures and learning methods	8
2.1.2.3 Generalization	10
2.1.3 End of Moore's law and limitations of von Neumann architectures .	11
2.1.4 Neuromorphic engineering: Nature as an inspiration	11
2.2 Neuromorphic cameras	13
2.2.1 Working principles and properties	13
2.3 Address-event representation (AER)	15
2.4 Neuromorphic chips	16
2.4.1 ODIN (UCLouvain)	16
2.4.2 MorphIC (UCLouvain)	17
2.4.3 Currently available neuromorphic chips	18

2.5	Neuromorphic datasets	18
2.5.1	Generate a neuromorphic dataset	18
2.5.2	What is a good neuromorphic dataset ?	20
2.5.3	Handwritten-digits neuromorphic datasets	20
2.5.3.1	MNIST-DVS	21
2.5.3.2	N-MNIST	21
2.5.4	Real-world neuromorphic dataset: N-CARS (PROPHESSEE)	23
2.5.5	Other neuromorphic datasets	25
3	State of the art: training SNNs	27
3.1	Neural information processing	27
3.1.1	Overview of biological neural networks	27
3.1.2	Neural coding	29
3.1.2.1	Rate-based coding	29
3.1.2.2	Population coding	30
3.1.2.3	Temporal coding	30
3.1.3	Synaptic plasticity	31
3.1.4	Neural models	31
3.1.4.1	Leaky integrate-and-fire model	31
3.1.4.2	Adapted leaky integrate-and-fire model	32
3.1.4.3	Other neuron models	33
3.2	Discrepancies between training ANNs and SNNs	34
3.3	Training SNNs in the spiking domain	35
3.3.1	Hebbian learning	35
3.3.2	Spike-timing-dependent plasticity (STDP)	36
3.3.3	Spike-driven synaptic plasticity (SDSP)	37
3.3.4	Spiking backpropagation	38
3.4	Offline training of SNNs: mapping ANN-to-SNN	39
3.4.1	Method overview	39
3.4.2	Converting an event-based dataset to a frame-based dataset	41

3.4.3	Threshold balancing algorithm	43
4	Methodology	45
4.1	Choosing an appropriate SNN architecture and a training method	45
4.1.1	SNN architecture	45
4.1.2	Training method	48
4.2	Methodology overview: a three-step strategy	48
4.3	Generation of a synthetic spiking dataset: SSP-MNIST	50
4.4	Spiking simulator	52
4.4.1	Assumptions	52
4.4.2	Structure	53
5	Simulation results	55
5.1	SSP-MNIST	55
5.1.1	Network structure	55
5.1.2	Frame-based reconstructed dataset	56
5.1.3	Training on KERAS	56
5.1.4	Thresholds balancing	57
5.1.5	Results	59
5.1.5.1	IF implementation (basic IF neuron model)	60
5.1.5.2	AIF implementation (adapted IF neuron model)	62
5.1.5.3	Hybrid implementation (trade-off)	63
5.1.5.4	Winner-Take-All (WTA) implementation	64
5.1.6	Discussion	65
5.2	N-MNIST	67
5.2.1	Network structure	67
5.2.2	Frame-based reconstructed dataset	68
5.2.3	Training on KERAS	69
5.2.4	Thresholds balancing	70
5.2.5	Results	72

5.2.5.1	Inference using the MNIST weights and the hybrid implementation (trade-off)	73
5.2.5.2	IF implementation (basic IF neuron model)	74
5.2.5.3	AIF implementation (adapted IF neuron model)	74
5.2.5.4	Hybrid implementation (trade-off)	75
5.2.5.5	Accuracy-Latency trade-off in SNNs	76
5.2.6	Discussion	79
5.3	N-CARS	81
5.3.1	Network structure	81
5.3.2	Frame-based reconstructed dataset	82
5.3.3	Training on KERAS	82
5.3.4	Thresholds balancing	84
5.3.5	Results	85
5.3.5.1	IF implementation (basic IF neuron model)	85
5.3.5.2	AIF implementation (adapted IF neuron model)	86
5.3.5.3	Hybrid implementation (trade-off)	86
5.3.6	Discussion	88
5.4	Hardware implementation: careful considerations	90
5.4.1	Resources distribution	90
5.4.2	Influence of the weights quantization on the classification accuracy .	91
5.4.3	Memory and power considerations	93
5.5	Comprehensive discussion	95
6	Conclusion	97
A	Appendix	99
A.1	Architecture of the Python scripts	100
A.2	Supplementary material	102
A.2.1	DVS working principle	102
A.2.2	N-MNIST rescaling	102
A.2.3	Block chain	103

A.2.4	Memory and power considerations	103
A.2.5	Error metrics	103
A.2.6	Quantization	104
A.3	SPARTAN-VII: data sheets	106
List of Figures		125
List of Tables		127

Introduction

“The eye is the window through which the mind perceives the world around it. It is also a window through which to discern the workings of the brain” [25]. Vision is without a doubt one of the most critical senses in human and animal perception. Little wonder that engineers have developed **cameras** to sense the world and algorithms capable of exploiting the data produced by such sensors. Indeed, computer vision and **machine learning** benefit from a growing interest in solving complex problems where computers are not explicitly programmed but rather have to automatically learn from data [1]. Despite genuine impressive performances [2, 3] in specific domains such as detection, tracking and pattern recognition of visual features, these algorithms are often energy intensive [2, 4, 5] and fall far short of the robustness and versatility of biological systems [6, 7, 8, 9, 10, 11, 12, 13].

Over the last decades, **computers have become more powerful** [14], which has provided a favourable basis for running large-scale simulations, leveraging huge databases and facilitating electronic integration in every day life devices (e.g. cell phones, laptops, interconnected devices, ...). However, this exponential growth in technology scaling known as Moore’s law [14] seems to come to an end [14, 15], calling for new computing systems and processing algorithms. Moreover, the need for **real-time processing** and **general-purpose autonomous and energy-efficient systems** is increasingly present in modern societies. This is driven by the desire of direct *interaction* with the environment which requires to extract the useful information on-the-fly after having collected the data in order to react directly and safely to external stimuli. This capacity meets great interest and widespread adoption in several applications such as healthcare, autonomous vehicles/devices and industrial monitoring. Unfortunately, current von Neumann architectures [16] have inherent hardware limitations [10] (e.g. separation between memory and processing structures, memory-related constraints as silicon area and access time, ...) that prevents them from efficiently meeting these requirements in practice. This drives a **paradigm shift** towards alternative architectures that could bring silicon systems closer to the efficiency of biological ones.

To address these challenges, an ongoing research field called **neuromorphic engineering** aims at developing practical solutions by drawing inspiration from nature and especially

from the brain [17, 18]. Although neuromorphic sensors (e.g. silicon retinas [12, 19, 25, 26, 27], silicon cochleas [18, 28]) and neuromorphic processors already exist [7, 20, 21, 22, 23, 24], conventional imaging algorithms *cannot* be directly mapped to these hardware platforms as their architecture is *fundamentally different* from von Neumann computers [3]. Consequently, new algorithms are needed to handle and process data efficiently on neuromorphic platforms and hopefully to reduce the gap between current computers and biological brains.

1.1 Goals

This thesis aims at investigating several ways of **training spiking neural networks** (SNNs) in order to **classify** accurately samples from **real-world neuromorphic datasets**. At the same time, neuromorphic processor(s) developed at UCLouvain come as a **support hardware framework**. Thereby, chip related *constraints* have to be taken into account when designing the SNN architecture in order to ensure a possible hardware implementation.

The binary classification task presented by the real-world neuromorphic dataset **N-CARS** provided by the French company PROPHESSEE is used as a benchmark, alongside handwritten datasets (e.g. SSP-MNIST, a synthetic spiking version of MNIST generated in this work, and N-MNIST). In addition, this work aims at proposing an **overview** of fundamental existing concepts, devices and algorithms in the current landscape of **neuromorphic engineering**.

1.2 Main achievements

The main achievements of this thesis are as follows:

- The generation of a **synthetic neuromorphic dataset** based on MNIST, called SSP-MNIST
- The development of a multi-cores Python **spiking simulator** for inference in the spiking domain
- The development of **Python scripts** to handle spiking data
- The reconstruction by accumulation of spikes of a **frame-based version** of the N-MNIST and N-CARS datasets

- An **adapted integrate-and-fire neuron model** thought to improve the classification accuracy of SNNs
- **First attempt** to apply an approach similar to ours on the real-world neuromorphic dataset **N-CARS**, to the best of our knowledge
- **Good classification accuracy** on benchmark datasets (e.g. 98.19% on SSP-MNIST, 96.83% on N-MNIST and 81.97% on N-CARS) while the networks were subject to hardware constraints.

1.3 Structure of this work

This work is structured as follows:

- Chapter 2 is organized along **two main axes**. The *first one* introduces the **context**, the current limitations and motivations that drive a paradigm shift towards new architectures, such as neuromorphic computing. A brief reminder about machine learning is also given. The purpose of the *second axis* is to present the **available hardware**. Neuromorphic chips and sensors are used as a framework that sets up constraints on the practical reality behind the simulations and solutions presented in this work. Currently-available spiking datasets are also presented.
- Chapter 3 gives a non-exhaustive literature overview of the state-of-the-art processing methods available so far in **neuromorphic engineering**. This chapter focuses mainly on how SNNs can be trained. It forms the basis for the theoretical concepts used in subsequent chapters.
- Chapter 4 describes the **methodology** established in order to come up with a working solution, including the chosen SNN architecture and training method. Details about the spiking simulator and the synthetic dataset are given.
- Chapter 5 presents and discusses the simulation **results** regarding the datasets considered in this thesis. It gives also considerations about the practical implementation of the proposed solution on available hardware. Finally, a comprehensive discussion closes this chapter.
- Chapter 6 provides the **conclusions** to the work carried out in this thesis.

Appendix [A.1](#) exhibits the global architecture of all Python scripts created for this work. Appendix [A.2](#) contains supplementary material. Appendix [A.3](#) gathers some data sheets of the SPARTAN-VII FPGA device. Figures in this work are best seen in color.

Fundamentals

This chapter introduces the framework of this thesis. Context and motivations are outlined to understand the substrate driving a paradigm shift towards new technologies, such as neuromorphic engineering. Then, a brief reminder on the basics of machine learning is provided. Afterwards, the neuromorphic hardware at disposal for this work is presented. It is used to set up constraints on the practical reality behind the simulations and proposed solutions. Finally, some currently available spiking datasets are also introduced.

2.1 Context and motivations

2.1.1 Frame-based processing

Conventional frame-based cameras are extensively used in the everyday life for several tasks such as industrial monitoring, recording videos or taking pictures, security, self-driving vehicles, pattern recognition, space observations, to name a few. They are in charge of the first step in visual applications, i.e. data acquisition. Each image is a regular grid of pixels with corresponding intensity or color values [29], often called frame. Computer vision and machine learning are fields strongly dependent on frame-based processing, where images are at the core of artificial vision [30].

Even though both frame-based sensing and processing could be improved for applications where high performance is needed, **three key drawbacks** are inherent to the hardware implementation. **First**, this kind of cameras *encode the scene at a pre-defined fixed acquisition rate* $\frac{1}{T_{frame}}$ [31] (usually around $30 - 60fps$ [30, 32]) which means that each pixel of the sensor is updated at the same time as its neighbours, which creates frames that need to be buffered in a memory. However, hardware limitations prevent these sensors to reach higher and higher acquisition rate which can lead to motion blur. Moreover, it has been observed that most of the information of everyday scenes is located around $1kHz$ [33] which is far beyond the best acquisition rates for current imagers, i.e. around $60fps$ [30] up to $300fps$ [34]. **Second**, a fundamental limitation of conventional cameras is the *lack*

of *dynamic content* and the acquisition of both relevant and non-relevant data [30]: frames are completely *independent* of what is going on in the observed scene [35, 36]. In other words, frame-based sensing and processing assume that the time constant $\tau_{reality}$ related with the changes in reality and T_{frame} are similar ($T_{frame} \simeq \tau_{reality}$) but this is often *not* true in practice [37]. If almost nothing happens in front of the camera ($T_{frame} \ll \tau_{reality}$), it keeps taking and saving frames every T_{frame} seconds, which is useless in terms of new information [37]. In fact, successive frames contain a huge amount of redundant data as most pixels have similar values from one frame to the next [36]. On the other hand, if something happens too fast ($T_{frame} \gg \tau_{reality}$), the limited and fixed acquisition rate does not allow to accurately capture what happened [37]. Consequently, the too dense or too low information produced at each frame is *not* an efficient way to encode natural scenes [31]. This is illustrated in Figure 2.1. Even if specialized algorithms can achieve impressive compression ratios (up to 50 for 1 [38]), the acquired data has been unnecessarily sampled, digitized, and transmitted, which is a waste of resources [36]. **Finally**, intra-frame dynamic range is usually limited around $60dB$ [39] with conventional imagers. It limits the ability of capturing scenes with high contrast in uncontrolled environments with natural lighting, which can compromise performance [27].



FIGURE 2.1 – *Identification of redundant and interesting data in an image obtained with a conventional camera (frame-based sensing).* The sky, trees and grass are unnecessary oversampled whereas the swing of the club and the movement of the player/ball are undersampled. Image adapted from [40].

With the growing use of the Internet-of-Things (IoT) pushing towards more and more inter-connected portable and autonomous devices, *real-time processing ability* and *energy consumption* are **two key requirements** that have to be taken into account in order to develop efficient and sustainable smart sensor nodes. The previously discussed three key drawbacks of frame-based sensing and processing prevent them from efficiently meeting these two requirements. In fact, notwithstanding the advances in computational power, pixel resolution and frame rate acquisition over the last decades, even the state-of-the-art methods available nowadays are much less effective than biological vision systems when it comes to robustness, reliability and energy consumption [8, 13, 41].

2.1.2 Basics of machine learning and computer vision

This section aims at reminding some basics of machine learning and computer vision as they are extensively used throughout this work. Explanations given in the following sections are *superficial*: the goal of this reminder is not to propose an extensive summary of state-of-the-art machine learning techniques but rather to outline some critical concepts that will be used throughout this work, such as some objectives of machine learning, several architectures and learning methods but also well-known trade-offs.

2.1.2.1 Objectives and applications

The development of computing and communication over the last decades has pushed our society to produce more and more *information* [42]. As a matter of fact, exponential growth is observed in the size of the data produced by several systems, devices and data sources [43] (double every 20 months [42]). For instance, more than $2.5 \cdot 10^{30}$ bytes is being created *every single day* and 90% of the total amount of stored data in the world has been created over the past two years [43, 44, 45]. This huge volume of wide variety data produced at a frenetic rate is often called the *Big Data* [45] and this trend could be even amplified with the growing use of the IoT. However this collected data is information in its raw form (i.e. recorded facts) and even if some information contained in there can be interesting, it is locked up in huge databases [43]. Extracting useful information underlying this huge amount of data is the objective of **data mining** which aims at developing automatic sifting in order to find strong patterns that can be generalized and then be used to make better prediction on future data [42].

To reach this goal, it uses **machine learning** as a technical basis [42]. It consists in developing efficient computational methods where computers are not explicitly programmed but rather have to automatically learn from data [1] in order to improve performance [46] and is mainly used for prediction (i.e. forecasting), classification, clustering and solving complex problems [10, 41, 42, 47, 48, 49]. Machine learning is based on **three essential axes**: data availability, computing power and algorithmic innovation. Nowadays, it is applied in a wide range of applications ranging from fundamental physics, healthcare, politics, self-driving vehicles to economics, IoT and communication [42, 45], to name a few.

2.1.2.2 Architectures and learning methods

A **neural network** is made up of at least **two key components**: the processing units (i.e. neurons) and the links that connect them together (i.e. weights) [50]. This has been loosely inspired by neural architectures in the brain [4, 51, 52] as shown in Figures 2.2 and 2.3. Input and output neurons are connected to the external environment whereas hidden neurons are connected to neighboring neurons and their output is used as input for other neurons in the network [50, 53]. Every neuron receives input(s), processes information using their transfer function and then produces an output depending on to their activation function, as shown in Figure 2.3. In this Figure, the transfer function TF_J is the weighted sum of the inputs $I_{1 \rightarrow M}$ with respect to the weights $w_{1 \rightarrow M, J}$ of the links connected to the neuron J . Accordingly, $net_J = TF_J(I, w) = \sum_{i=1}^M I_i \cdot w_{i, J}$. The output y_J of the J^{th} neuron is expressed as $y_J = AF_J(net_J)$ where the activation function AF_J is often the sigmoid, tanh or ReLu function [37], among others. A wide variety of powerful machine learning **architectures and algorithms** have been investigated [50] and some of the most popular [5] are Principal Component Analysis (PCA) [54], K-means clustering [46], decision trees [55], auto-encoders [2, 48], Deep Belief Networks (DBNs) [2, 41], Restricted Boltzmann Machines (RBMs) [2, 41], Self Organizing Maps (SOMs) [46], Support Vector Machines (SVMs) [56] and **Artificial Neural Networks** (ANNs) [2, 5]. The focus is made on the latter one in this work. An ANN is mainly characterized by **three elements**: the *architecture* (i.e. connections between neurons), the *learning method* (i.e. algorithm that determines the weights of the connections) and the *activation function* (AF_J , i.e. the function used to produce the output of a neuron given its inputs, e.g. sigmoid, tanh, ...).

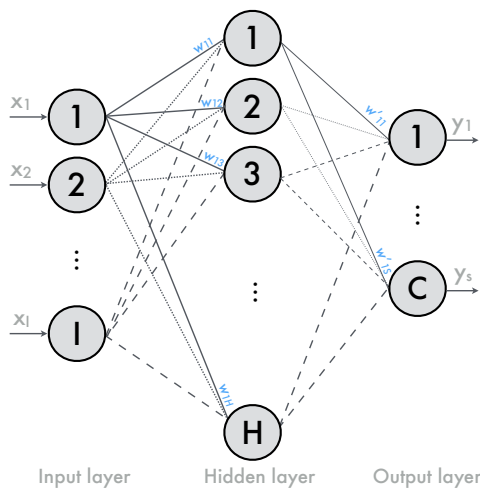


FIGURE 2.2 – **Fully-connected network**. Illustration of a three layers FC network with I input neurons, H hidden neurons and C output neurons.

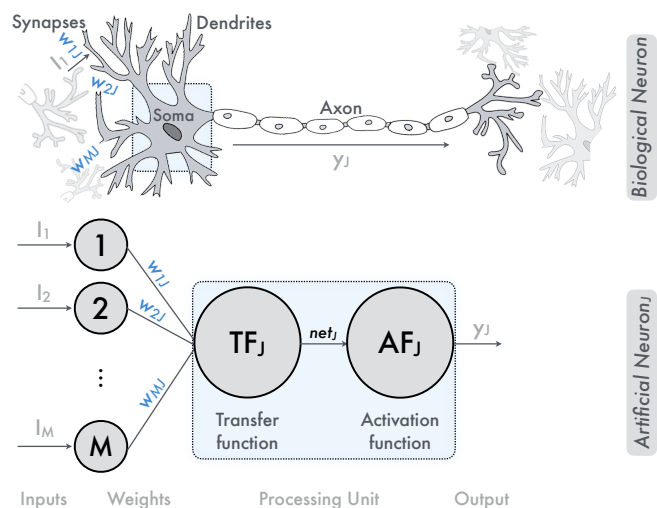


FIGURE 2.3 – **Processing unit (artificial neuron)**. Similarities between an artificial and a biological neuron. Image inspired by [37, 57, 58].

A **fully-connected (FC) layer** is a layer where every single neuron belonging to the layer is connected to all neurons in the previous layer [53] as can be observed in Figure 2.2. This architecture requires to store and learn a large number of parameters, often leading to copious power/energy consumption [53]. On the other hand in a **convolution (Conv) layer**, neurons are connected following a structure inspired by the organization of the animal visual cortex [36, 53, 59]. Neurons in a given layer are only connected to a subset of neurons in the previous layer, called the *local receptive field* (shape $k_a \times k_b$) [48], and are effective at extracting similar patterns in data with specific topological structures [48]. Figures 2.4 and 2.5 help to better visualize and understand such networks. Each connection in this subset of neurons has a particular weight and the whole subset of weights is called the *kernel/filter* (shape $k_a \times k_b$) [49]. It acts as a local feature extractor and transforms the input into a *feature map* [48, 60, 61], through the convolution operation presented in Figure 2.5. **Convolution Neural Networks** (CNN, ConvNets [61]) are a particular case of ANNs which have much fewer connections and parameters making them easier to train while maintaining similar performances [49]. The capacity of the network can be controlled [49] by varying parameters $A, B, M, N, H, k_1, k_2, k_3$ and k_4 as detailed in Figure 2.4, where (A, B) is the shape of the input, M and N are the number of channels in the first and second convolution layers respectively, H is the number of neurons in the hidden layer and $k_{1 \rightarrow 4}$ are the kernel sizes for the convolution and pooling filters. Varying the depth and breadth of CNNs has a strong impact on the capacity [3, 49, 53], i.e. the ability of solving complex tasks. Shapes (A_l, B_l) are defined by the kernel size k_l and the type of padding (e.g. valid or same). For a valid padding and k_l uneven, $(A_l, B_l) = (A_{l-1} - k_l - 1, B_{l-1} - k_l - 1)$, where l is the index of the layer in the network.

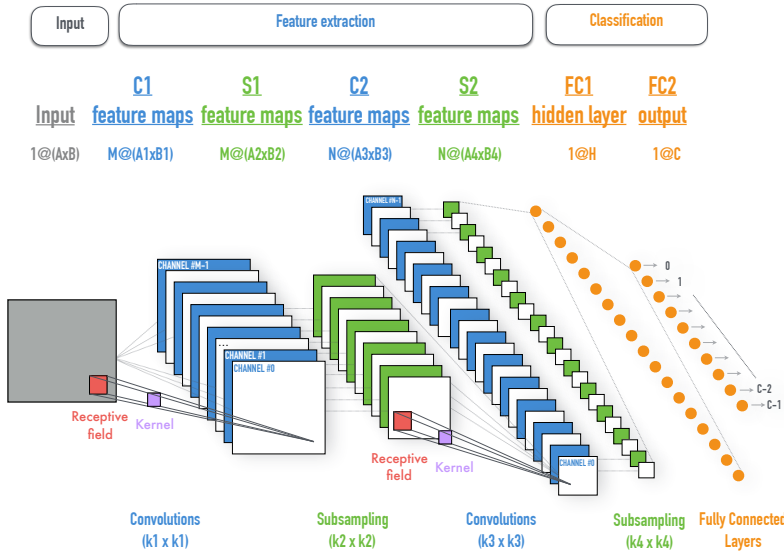


FIGURE 2.4 – **Convolution network.** Example of a basic two layers convolution-pooling network followed by two FC layers.

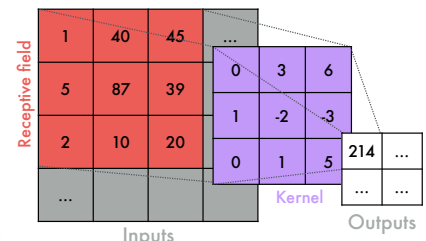


FIGURE 2.5 – **Illustration of the convolution operation.**

For the **learning process**, a common use is to split unevenly the training dataset in two parts: the *training set* and the *validation set* [49]. The first one is used to train the weights while the second one is used to assess intermediate performance. When the best weights have been found, the final performance is obtained by assessing how well the method performs on the *test set*, which has not been used before [1]. It exists several common learning methods in conventional machine learning but the three most famous categories [2] are *supervised*, *unsupervised* and *reinforcement learning* [32, 46, 51]. They differ mainly in the use of a teacher (e.g. labelled data) during the learning process [62]. In most cases, ANNs are trained using stochastic gradient descent (SGD) optimization and *backpropagation* algorithm [50, 53]. The basic idea behind this gradient-based method is to update individual weights in order to minimize an error function at the output layer by successively computing local derivatives with respect to the weights, for each neuron and each layer from the output to the input, hence the name backpropagation [4, 53]. This requires differentiable activation functions [3, 52]. In order to speed up the convergence of the training process, the data is usually divided in batches on which parameters are updated multiple times [53]. The learning rate η allows to control the updating speed of the weights [53].

2.1.2.3 Generalization

Finding an appropriate network architecture, choosing an appropriate training method and tuning the (hyper-)parameters to get the best performances for a given application are **key challenges** in machine learning [1, 2]. After the training phase, the network is expected to generalize well and to perform accurately on new samples. However, there exists a **trade-off** between *dataset size* and *complexity*, which impacts the generalization performance of the network [46]. **Overfitting** is a well-known issue in machine learning that occurs in supervised learning when (large and deep) networks memorize well the training set but deliver poorer accuracy on unknown samples, i.e. validation or test set [1, 4, 63]. This is often attributed to an insufficient size of the labeled training set compared to the size and capacity of the network [49]. This is a key reason underlying the motivation to store more and more data: the more data available, the more accurate and the more robust the results that machine learning techniques can provide [43]. This explains also why enormous datasets are needed in conventional machine learning. Thereby, a **major drawback of machine learning** is that it needs a *massive amount of energy* in order to learn from so many examples, which is even worse with *deep learning* [2, 4, 5]. Mobile and embedded applications are thereby limited [11]. Regularization [1, 4] (e.g. dropout [4, 63, 49], batch normalization [64]) and data augmentation [49] are frequently used to tackle overfitting.

2.1.3 End of Moore's law and limitations of von Neumann architectures

Separation between memory and processing structures is one of the **main limiting factor** in current von Neumann architectures [16], as well as memory-related constraints as silicon area and access time [10]. This issue is known as the *von Neumann bottleneck* (VNB) [10]: this is a throughput limitation due to unmatched rate of data transfer between the memory and the central processing unit (CPU) as shown in Figure 2.6. In fact, the CPU has to wait idle for data to be fetched from memory, which has a lower transfer speed [16]. Thereby, *latency* is unavoidable and *electrical power* is wasted while waiting [10]. Some efficient solutions exist (e.g. caches, distributed computing among multiple cores in GPUs [10]) but it does not solve the fundamental limitation of this computing substrate [10]. In addition, the end of Moore's law [14, 15] calls for new computing architectures and information encoding in order to compensate for the slow down in technology scaling.

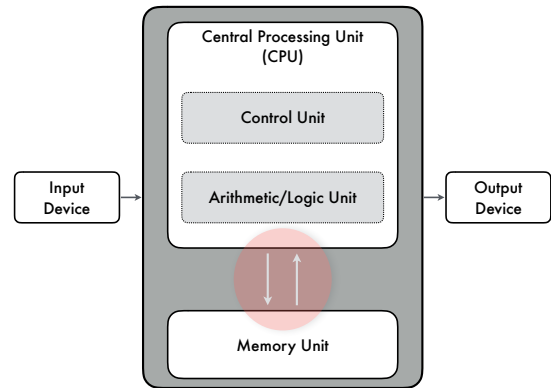


FIGURE 2.6 – *von Neumann architecture scheme*. The red disc highlights where the VNB occurs. Image inspired from [65].

A **paradigm shift is required** to take up that challenge, leading to alternative non von Neumann architectures that could bring silicon systems closer to the efficiency of biological ones.

2.1.4 Neuromorphic engineering: Nature as an inspiration

Nature provides a large amount of energy-efficient, adaptive, compact and intelligent systems [6] which is why it has been a key *source of inspiration* for development of scientific systems. Inspired by neuro-biological systems, **neuromorphic engineering** (sometimes called *event-based processing*) follows a bottom-up approach and is a research field that aims at designing and fabricating new efficient systems made of electronic neuron and synapse circuits [17, 18, 66, 67]. More generally, it refers to a range of electronic systems that aim at mimicing and modelling biological systems in analogue and/or digital signal [8, 17, 32]. Several devices that emulate sensory transduction such as silicon cochleas [18, 28] and silicon retinas [12, 26, 19, 25, 27] (which are the focus of this work) have been developed for a wide range of applications [18, 19].

Two main differences with conventional architectures have to be highlighted. The **first one** is related to *processing and memory distribution* [67, 68]. In opposition with the current von Neumann architectures, processing (i.e. neurons) and memory (i.e. synapses) are not split from each other which allows for a *massively-parallel* data handling [67, 68, 69]. **The second one** is related to the way information is represented: only *spikes* are used to encode and process the data in the time domain, in a totally *asynchronous* way [7, 32]. Moreover, spikes are *sparse* in time, offering an interesting property to spiking networks: by using few spikes which have high information content, energy consumption can be reduced [6, 10, 70, 71, 72, 73].

In addition, it is worth noticing that working principles of current processing methods and neuro-biological systems are **fundamentally different** [12, 18]. In fact, the latter work with slow signals (several *Hz*), inhomogeneous components (electro-chemical propagation, complex biology, living substrate, ...) and deal with noise as an inherent feature in their behavior [25, 69], while traditional computers make use of high-speed clock rates (*MHz* – *GHz* [74]) with mostly synchronous logic and try to get rid of noise as much as possible [8, 25]. Furthermore, even though current computers are useful for biophysics simulations, they seem to be far below the performances of biological brains when it comes to energy consumption, scale, adaptability and efficiency in real-world tasks [6, 7, 8, 9, 10, 11, 12, 13, 18, 25], as indicated in Figure 2.7. For instance, a typical room-size supercomputer is about 10^3 heavier, 10^4 bigger and 10^6 more power consuming than the cantaloupe-size lump of neural tissue that composes a human brain [12].

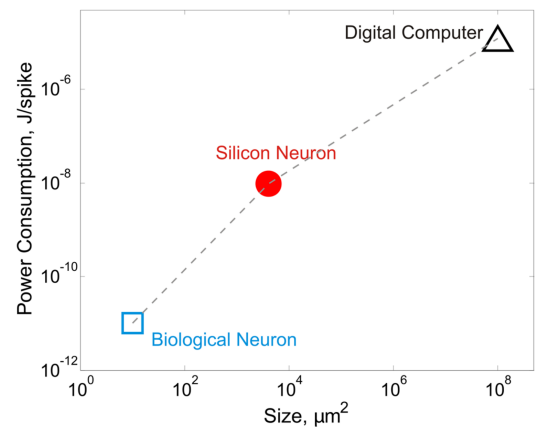


FIGURE 2.7 – *Comparison of power and space efficiency between digital computers and biological/silicon neurons. Image adapted from [9].*

To **sum up**, the motivation to develop *real-time processing* and *ultra-low power consumption* devices supports a paradigm shift to the neuromorphic processing. Getting closer to biological performance is consequently an ongoing challenge not only for neuromorphic engineering, but also for neuroscientists.

2.2 Neuromorphic cameras

The main objective of this section is to understand how does the neuromorphic vision sensor behave in order to be able to make the most of the collected data. In fact, changing the sensor from a frame-based camera to an event-based camera is not only changing the encoding of the data but also the whole processing chain coming behind it.

2.2.1 Working principles and properties

Silicon retinas [12, 25] or event-based cameras [19, 26, 27], such as the Dynamic Vision Sensor (DVS) [19] and the Asynchronous Time-based Imaging Sensor (ATIS) [26], were developed on the basis of a **biological retina** [12, 26]. Their working principle is fundamentally different from current imagers: each pixel is **independent** from its neighbours and **react to a change in illumination** by asynchronously generating an event [27, 32], hence the name *event-based* technology. The frame principle is thereby completely abandoned [27]. The term *event* refers to a spike defined by the spatial location of the illumination change on the silicon retina also called *address-events* (x, y) , a timestamp t , and the polarity p of the spike [75], as given by the expression $\mathbf{e} = [x, y, t, p]$. Polarity can be either positive or negative (ON/OFF, respectively): if the change of local logarithmic illumination has increased (resp. decreased) above (resp. below) a pre-defined threshold, an event is generated with the corresponding polarity [32, 76] as shown in Figure 2.8. Supplementary material is given in Appendix A.2.1.

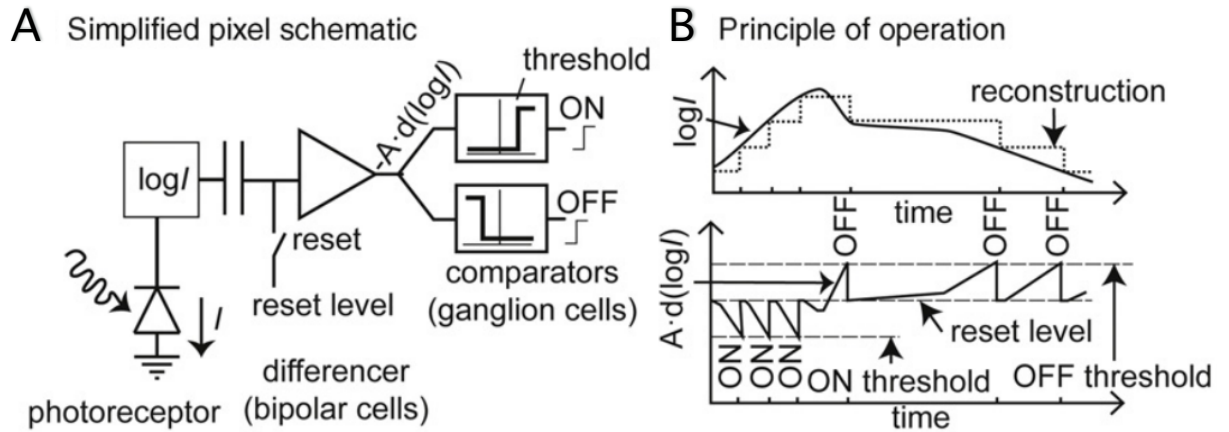


FIGURE 2.8 – **General working principle of a DVS camera.** (A) *Simplified pixel schematic* (B) *ON/OFF spikes generation principle*. Image adapted from [6, 27].

The output of such sensors is consequently a **stream of asynchronous events** triggered by a change in temporal contrast [36], which makes the output data-rate dependent on the content of the scene [19]. It is a more efficient encoding that reduces spatio-temporal redundancy [32], leading to savings in both bandwidth and power consumption [19]. Figure 2.9 displays typical outputs produced by a DVS camera when spikes are accumulated over a time interval (usually tens of *ms*). The *contrast sensitivity* of event-based cameras is often set about 1.5 – 15% and has a strong influence on the quality and richness of the observed scene but also on the output spike rate of the sensor [19, 36], as exhibited in Figure 2.10.

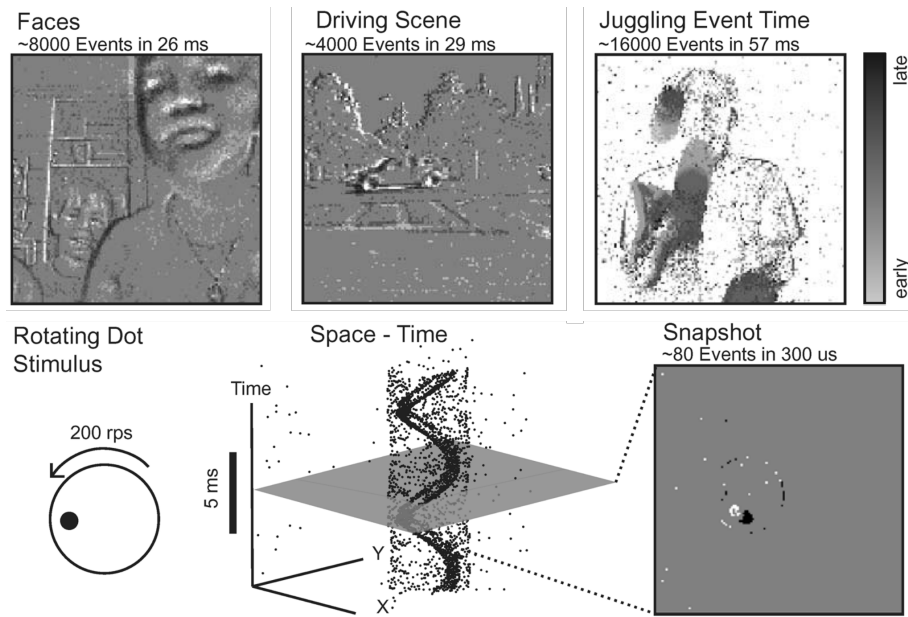


FIGURE 2.9 – **Data obtained from a DVS sensor.** Different visualization techniques are possible: histogram of events in 2D-space over a time window which forms a contrast image based on ON/OFF events, or a gray scale reconstruction showing the relative event time, or even a space-time representation to view the spatio-temporal structure. Image adapted from [6, 27].

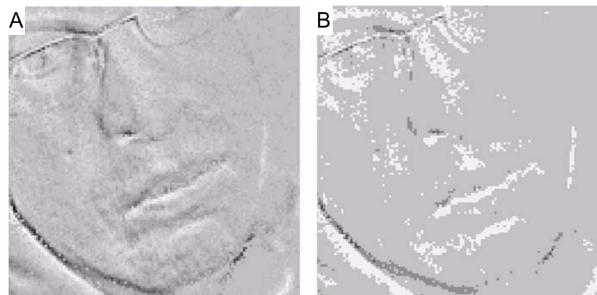


FIGURE 2.10 – **Impact of the contrast sensitivity** of a DVS sensor on the quality of a reconstructed scene. The accumulation is carried out by accumulating events over a period of 30ms. Contrast sensitivity is set to 1.5% in (A) and 10% in (B). Image adapted from [19].

The **effective sampling rate** of neuromorphic vision sensors is also improved and can reach several tens of kHz [19, 35], which allows to avoid motion blur and makes them really attractive and well-suited for high speed motion and real-time applications. Figure 2.11 illustrates how event-based sensors differ from their frame-based counterparts. In addition, **dynamic range** in illumination of such sensors is improved up to $140[dB]$ thanks to the logarithmic sensitivity of each pixel [39]. When compared with the figures of merit of traditional cameras (around $60[dB]$ [39] and roughly $60Hz$ [30] resp.), these numbers speak for themselves. When an image is captured with a conventional camera, the temporal information of the scene is *de facto* lost [76]. Several videos can be found in [77] for readers interested in better visualizing the features specific to neuromorphic sensors.

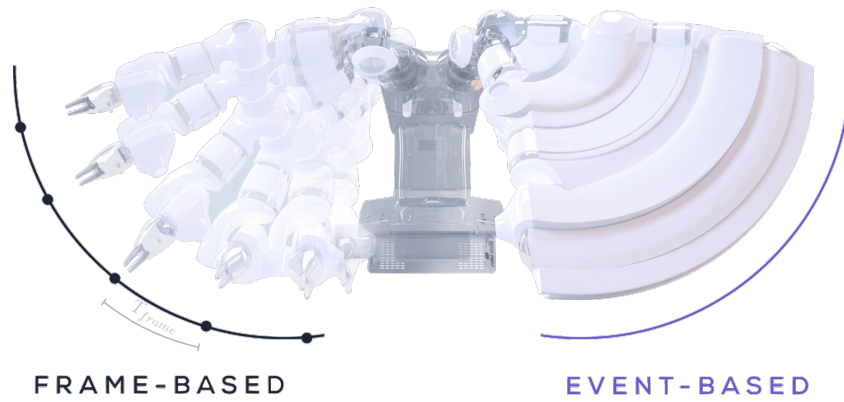


FIGURE 2.11 – *Schematic comparison of sampling rate performance between frame-based and event-based imagers. Image adapted from [78].*

2.3 Address-event representation (AER)

A single neuron is not interesting when it comes to realize complex calculations. In fact, the computational power of the network resides in the interconnection between multiple neurons [3]. Consequently, some challenges arise when designing huge arrays of interconnected neurons on a silicon chip. Wiring and circuit design are fixed after chip fabrication and design is limited to 2D substrate [6], contrary to biological brain where neurons are interconnected in a 3D world and where neural plasticity allows on-the-fly wirings adjustments [79]. However, there is something to gain thanks to the mobility of electrons in a transistor which is roughly 10^6 times greater than the ions mobility in the biological neurons, resulting in a significant speedup [6, 25]. Using that speed advantage of electronics, an alternative signal transmission protocol called **address-event representation (AER)** can be implemented. It allows to **simulate a point-to-point connectivity** from each pixel of the sensor to an external device via a narrow channel by giving an ad-

dress to every single pixel/neuron on-chip. It can be seen as a virtual bundle of axons (i.e. a silicon optic nerve) connecting two neuromorphic devices [12]. An event $\mathbf{e} = [x, y, t, p]$ is sent serially through the bus only when a spike is generated, as shown in Figure 2.12. As it takes only a few nanoseconds for these addresses to be transmitted and thanks to the sparse nature of the output data of the sensor, AER can handle multiple spikes addresses transfer [6]. The typical throughput of multi-channel AER systems is about tens of *Mevents/s* [13, 80]. Multiplexing and arbitration can occur but only when the spike rate exceeds the transmission capacity of the communication link [8]. This signal transmission scheme has been widely adopted in neuromorphic engineering thanks to its high speed potential and energy efficient nature. It is usually used to transfer events between silicon retinas/cochleas and neural processors as well as inter-devices communication [19, 27].

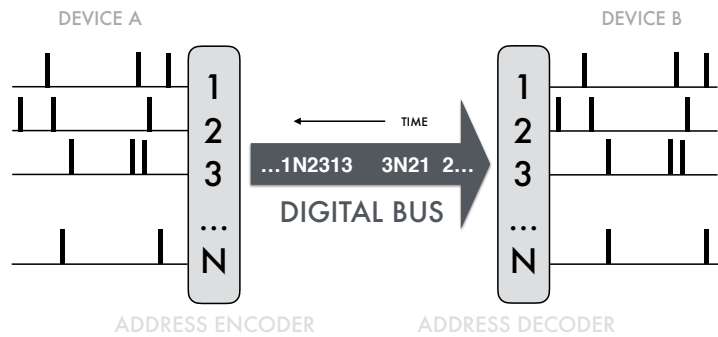


FIGURE 2.12 – **Schematic representation of the AER communication protocol.** The numbers in the address encoder/decoder are the neurons indexes. Image inspired by [8, 27].

2.4 Neuromorphic chips

If neuromorphic sensors are created to sense the real world while using sparsity of spike coding, neuromorphic chips are needed in order to keep spiking representation from sensing to processing [32]. This section focuses on existing neuromorphic chips, their strengths and weaknesses in order to have a better understanding of which applications can currently be supported by neuromorphic hardware.

2.4.1 ODIN (UCLouvain)

ODIN is an online-learning digital spiking neuromorphic processor in $28nm$ FDSOI CMOS that was developed at UCLouvain in 2018 [7]. It has a crossbar architecture. One particularity of this chip is that neurons can be programmed to emulate all the 20 Izhikevich behaviors, which are the 20 behaviors observed in biological neurons [7, 81]. A Python

simulator with a 1-to-1 mapping correspondence was developed. More details are given in Figure 2.13.

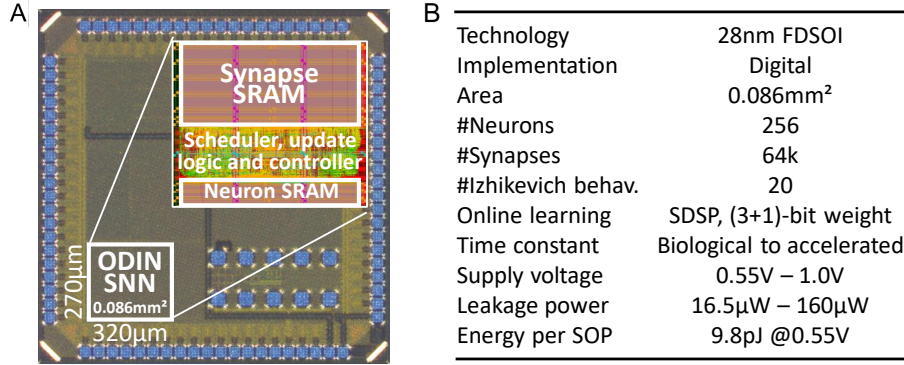


FIGURE 2.13 – **ODIN**. (A) Chip microphotograph with zoom on floorplan view. (B) Specifications and measurements. Image adapted from [7].

2.4.2 MorphIC (UCLouvain)

MORPHIC is a quad-core binary-weight digital neuromorphic processor embedding a stochastic version of the spike-driven synaptic plasticity (S-SDSP) learning rule [20]. This chip attempts to tackle the *connectivity challenge* for large-scale chip interconnection. Yet, connectivity between the four cores is limited as shown in Figure 2.14 and sets up constraints on the architectures that can fit on this processor. MORPHIC is also a crossbar architecture and embeds binary synapses, which drastically reduces the energetic footprint of inference [10]. A Python simulator with a 1-to-1 mapping correspondence was developed and is used in this work. The power consumption of this chip is about 5mW, as calculated from the specifications given in Figure 2.14.

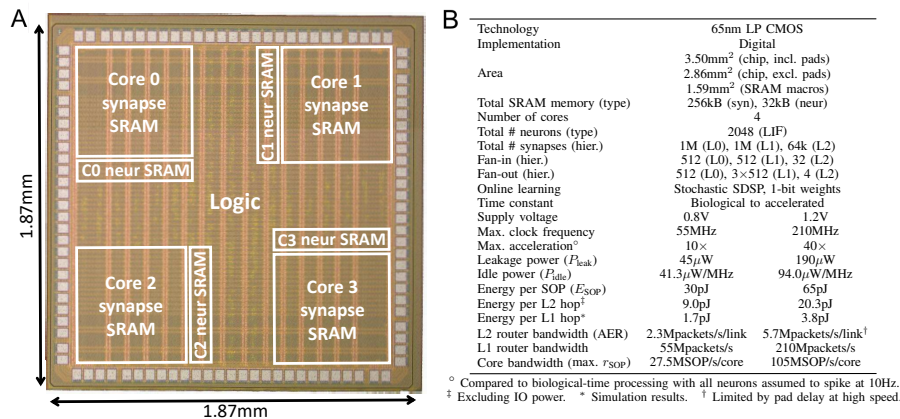


FIGURE 2.14 – **MorphIC specifications**. (A) Chip microphotograph showing the floorplan view and synapse SRAM macros of each core. (B) Specifications and measurements. Image adapted from [20].

2.4.3 Currently available neuromorphic chips

Other neuromorphic chips are given in Table 2.1. This table is a simplified version of the one available in [7] which is the most detailed and fair comparison between neuromorphic processors to the best of our knowledge. As it can be observed in Table 2.1, there exists several neuromorphic chips with different strengths and weaknesses. UCLouvain chips are required to be used in this work (the two last columns of the Table 2.1).

Author	Benjamin [24]	Qiao [82]	Moradi [83]	Painkras [21]	Akopyan [23]	Davies [22]	Frenkel [7]	Frenkel [20]
Publication	PIEEE, 2014	Front. NS, 2015	TBioCAS, 2017	JSSC, 2013	TCAD, 2015	IEEE Micro, 2018	TBioCAS, 2019	TBioCAS, 2019
Chip name	NeuroGrid	ROLLS	DYNAPs	SpiNNaker	TrueNorth	Loihi	ODIN	MorphIC
Implementation	Mixed-signal	Mixed-signal	Mixed-signal	Digital	Digital	Digital	Digital	Digital
Technology	0.18 μ m	0.18 μ m	0.18 μ m	0.13 μ m	28nm	14nm FinFET	28nm FDSOI	65nm LP CMOS
Neurosynaptic core area [mm ²]	168	51.4	7.5	3.75	0.095	0.4	0.086	0.715
# Izhikevich behaviors [†]	N/A	(20)	(20)	Programmable	11 (3 neur: 20)	(6)	20	3
# neurons per core	64k	256	256	max. 1000 [°]	256	max. 1024	256	512
Synaptic weight storage	Off-chip	Capacitor	12-bit (CAM)	Off-chip	1-bit (SRAM)	1- to 9-bit (SRAM)	(3+1)-bit (SRAM)	1-bit (SRAM)
Embedded online learning	No	SDSP	No	Programmable	No	Programmable	SDSP	S-SDSP
# synapses per core	-	128k	16k	-	64k	1M to 114k (1-9 bits)	64k	528k
Time constant	Biological	Biological	Biological	Bio. to accel.	Biological	N/A	Bio. to accel.	Bio. to accel.
Neuron core density [neur/mm ²] [*]	390	5	34	max. 267 [°]	2.6k	max. 2.6k	3.0k	716
Synapse core density [syn/mm ²] [*]	-	2.5k	2.1k	-	673k	2.5M to 285k	741k	738k
Supply voltage	3.0V	1.8V	1.3V-1.8V	1.2V	0.7V-1.05V	0.5V-1.25V	0.55V-1.0V	0.8-1.2V
Energy per SOP	941pJ [▲]	77fJ ^Δ	30pJ [▲] (at 1.3V)	26.6nJ [▲]	26pJ [▲] (at 0.775V)	>23.6pJ ^Δ (at 0.75V)	12.7pJ [▲] (at 0.55V)	51 pJ [▲] (at 0.8 V)

[†] By its similarity with the Izhikevich neuron model, the AdExp neuron model is believed to reach the 20 Izhikevich behaviors, but it has not been demonstrated in ROLLS and DYNAPs. The neuron model of TrueNorth can reach 11 behaviors per neuron and 20 by combining three neurons together. The neuron model of Loihi is based on an LIF model to which threshold adaptation is added: the neuron should therefore reach up to 6 Izhikevich behaviors, although it has not been demonstrated.

^{*} Neuron (resp. synapse) core densities are computed by dividing the number of neurons (resp. synapses) per neurosynaptic core by the neurosynaptic core area. Regarding the synapse core density, NeuroGrid and SpiNNaker use an off-chip memory to store synaptic data. As the synapse core density cannot be extracted when off-chip resources are involved, no synapse core density values are reported for these chips.

^Δ Incremental values describe the amount of energy paid per each additional SOP computation, they are measured by subtracting the leakage and idle power consumption of the chip although the exact power contributions taken into account in the SOP energy vary across chips.

[▲] Global values are obtained by dividing the total chip power consumption by the SOP rate.

TABLE 2.1 – *Chip characteristics and measured performances compared with the state of the art of spiking neuromorphic circuits. The original version is available in [7].*

2.5 Neuromorphic datasets

This section starts by giving some details on the different existing ways to generate a neuromorphic datasets. Then, criteria for a good neuromorphic dataset are provided and finally, some currently available spiking datasets are presented.

2.5.1 Generate a neuromorphic dataset

They are mainly **three ways** to get a spiking dataset [76] and they are presented in this section. How to generate such datasets is a crucial consideration as it modifies the structure onto the underlying data [8, 76].

The **first way** is to *convert an existing frame-based dataset* into an event stream through digital data processing [3] as shown in Figure 2.15. The *key advantage* is that the rule for conversion is a degree of freedom. For instance, a popular approach is to generate

synthetically a Poisson spike train [3, 4, 11, 41, 51, 84, 85, 86] for each particular pixel, where the rate of firing is proportional to the gray-level (i.e. illumination) of this pixel and repeat the process for each pixel of the image and for each image of the original dataset. The main *drawback* of this approach is that the spiking dataset produced is *not* equivalent to a realistic spiking dataset recorded with an event-based camera [3]. Moreover, noise present in the real world and in the neuromorphic sensor is *not* taken into account which might lead to optimistic performances [87]. Even worse, performances might collapse when switching to a non-synthetic real-life dataset captured with a neuromorphic sensor [3].

The **second way** to get a spiking dataset is to *convert a frame-based dataset by using a neuromorphic camera* in front of a screen on which images from the conventional dataset are displayed (with small saccades to simulate dynamics) [3, 87, 88]. This approach could be seen as a gentle mix between frame-based and event-based technology but still, it is not a fully event-based dataset as the dynamics has been simulated. However, noise in the neuromorphic sensor is now considered and the dataset is getting closer to a real-world spiking dataset [87].

The **third way** to generate a spiking dataset is to *record real-life scenes with an event-based camera* [31, 89, 90]. This is the best way to obtain a representative dataset as it ensures to capture the noise and the real dynamics of the observed scenes. It is hard to find that kind of dataset nowadays.

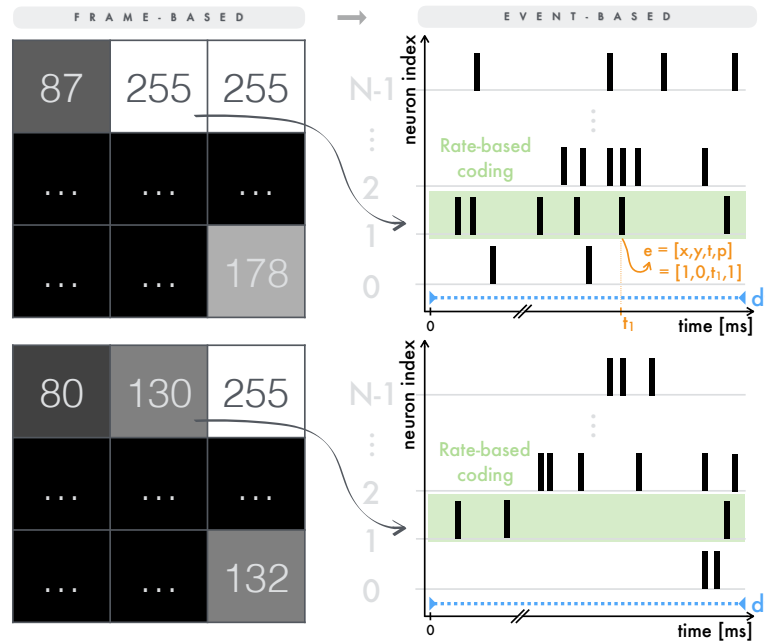


FIGURE 2.15 – *Frame-based to event-based dataset conversion principle. Illustration for the first method.*

2.5.2 What is a good neuromorphic dataset ?

Not having a reliable neuromorphic dataset is a **fundamental issue** limiting event-based object classification [31]. If the dataset is too small, it prevents an accurate evaluation of the proposed method. If the dataset is large enough but not really representative of the real dynamics of a scene nor of the inherent event-based temporal resolution, there is no guarantee at all that the method tested on this data will reach similar performances in real-world conditions [31]. When considering a dataset, it is fundamental to make sure that it is *sufficiently large* and presents a problem of *sufficient difficulty* [8, 76]. However, contrary to well-established fields as machine learning and computer vision which possess a large number of datasets, benchmarks and figures of merit useful to compare performance of algorithms and systems, the neuromorphic field is a quite novel field and **suffers from a lack of representative datasets and metrics** [8, 76].

What **criteria** define a good neuromorphic dataset is consequently a very important question in neuromorphic engineering [87]. An important feature that seems to stand out is that the dataset has to be inherently *spatiotemporal* [87]. In addition, *real-life dynamics* seems to be another crucial feature, i.e. scenes recorded over different rotations, lighting conditions, sizes, etc... Consequently, only the third method presented in the previous section seems to be really suited in order to generate a relatively good dataset. Neuromorphic datasets do not include only visual spiking datasets: other challenging candidates are audio spiking datasets [87], for instance.

2.5.3 Handwritten-digits neuromorphic datasets

The Modified National Institute of Standards and Technology (MNIST) database [61] is well-known and ubiquitous in conventional machine learning and computer vision. It is used as a benchmark for almost all neural networks architectures [4]. It is composed of 60k training samples and 10k test samples and good classification accuracy is considered above 98 – 99[%], e.g. the best accuracy reaches 99.82[%] at the moment [47].

However, this dataset is *not* a neuromorphic dataset. In order to assess performances of SNNs architectures on this specific dataset, a **spiking version of MNIST** is therefore needed. This was the motivation behind the creation of the MNIST-DVS [91] and N-MNIST [88] datasets. They were both generated using the second conversion method presented previously, i.e. converting a frame-based dataset by using a neuromorphic camera.

2.5.3.1 MNIST-DVS

MNIST-DVS [91] was created by recording the original MNIST dataset with a DVS camera in front of a LCD monitor [3]. The DVS sensor and the screen were stationary whereas the image displayed was slowly moving on the screen to create dynamics [3]. Only 10k samples were converted (full MNIST is 70k samples) at three different scales which makes 30k spiking samples at the end. Each sample was recorded during 2 to 3 [s]. This dataset comes with some MATLAB scripts for pre-processing [3] including 75Hz LCD screen refresh rate harmonic elimination, moving digits stabilization and polarity bit removing.

2.5.3.2 N-MNIST

N-MNIST [88] stands for Neuromorphic-MNIST and was also created by recording the original MNIST dataset with a DVS camera in front of a screen [3]. However this time, both the screen and the image were static whereas the DVS sensor was moving. It was mounted on a pan-tilt unit which induced three sequential 100ms saccades at three different angles (0° , $+60^\circ$, -60°) [92], as clearly shown in the first subplot of Figure 2.16 and in Figure 2.19. Consequently, each sample of the original dataset was converted eventually to a 300ms event stream. The second subplot of Figure 2.16 displays a typical sequence of spikes for a particular pixel of the sensor. The two polarities can be identified (ON = red, OFF = blue) and spikes seem to occur in "small batches" with respect to the polarity. The dataset can be downloaded online at <https://www.garrickorchard.com/datasets/n-mnist>.

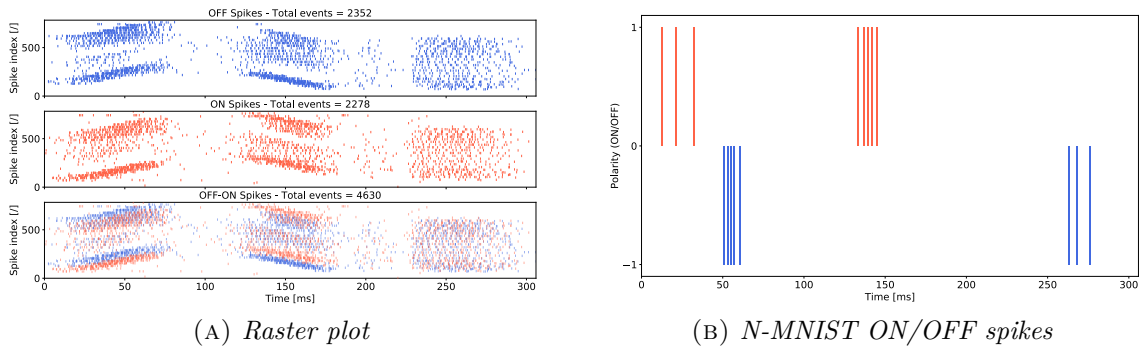


FIGURE 2.16 – (A) **N-MNIST input spikes raster plot**, for the first sample of the training set (label = 5). (B) **N-MNIST ON/OFF spikes**, for the first sample of the training set (label = 5) and pixel index = 290, i.e. $[x, y] = [10, 10]$.

This conversion process has been discussed because of the dynamics introduced by the pan-tilt movement, which is not really representative of a real-life dynamics because it is too deterministic, limited to two dimensions and restricted to ideal environment [87]. Moreover, N-MNIST contains less information than MNIST, because of the noise and

movements in the image [87]. In fact, N-MNIST can at best emulate the original MNIST dataset with that conversion method [87]. Nevertheless, the noise introduced in the dataset due to the neuromorphic camera is a good thing because it makes the dataset more realistic [87]. Images were reconstructed by accumulating spikes over a time interval, as show in Figure 2.17. It can be observed that if the accumulation window is too large, the reconstructed image looks really blurry (e.g. as in the first subplot of Figure 2.17). Each reconstructed image is normalized with the highest spikes count such that the brighter pixel has a value of 255. More details about frame-based reconstruction from event-based data are given in Section 3.4.2.

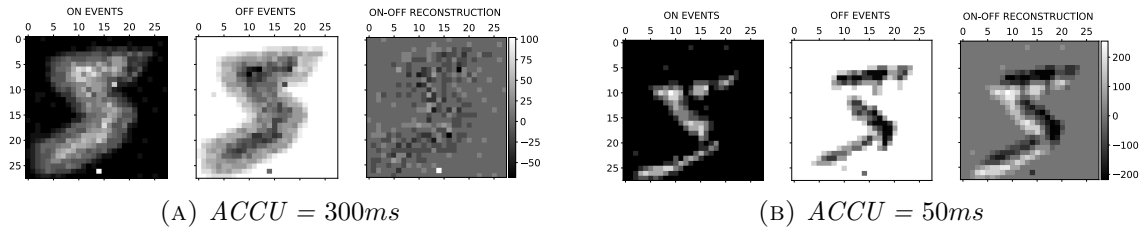


FIGURE 2.17 – *N-MNIST frame-based reconstruction based on accumulation of input spikes and ON-OFF reconstruction. This is done for the first sample of the training set (label = 5) with an accumulation window of (A) 300ms and (B) 50ms.*

The original N-MNIST dataset was cropped from a 34x34 input shape to a 28x28 shape (i.e. MNIST shape) in order to be able to transfer an architecture from MNIST to N-MNIST (more details are given in Section 5.2). This cropped version is used in the rest of this work. This modification can be noticed in Figure 2.17. No pre-processing was applied, only discarding some pixels to get the desired shape. By visual inspection of this dataset, events seemed to happen more frequently in the bottom part of the image. However, the cropping was carried out with respect to the center of the image, Appendix A.2.2. There are *in average* 4117 spikes per sample as shown in Figure 2.18.

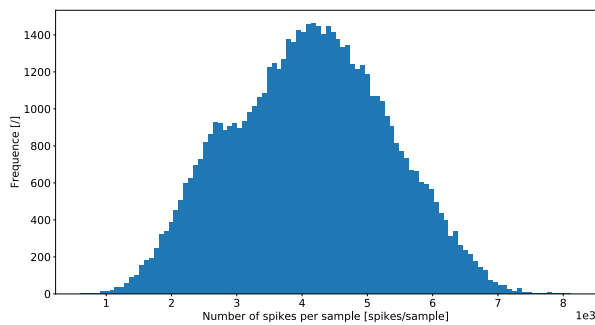


FIGURE 2.18 – *N-MNIST spikes per sample distribution, for the whole training set. Mean = 4117 spikes/sample.*

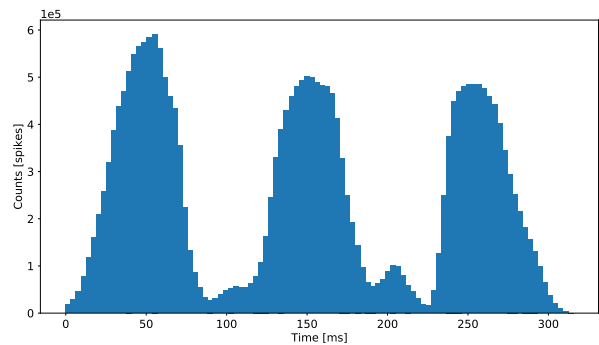


FIGURE 2.19 – *N-MNIST time distribution of spikes, for 6000 randomly chosen samples of the training set.*

2.5.4 Real-world neuromorphic dataset: N-CARS (Prophesee)

As explained previously in Section 2.5.1, the need for large real-world datasets is one of the main slowing factor for event-based vision and classification [31, 29].

To face that problem, the french company PROPHESEE [89] has created a brand new real-world dataset called **N-CARS**. Scenes have been recorded with an ATIS [26] event-based sensor in urban environments, i.e. embedding dynamics of the sensor as well as the one of the environment which is really attractive. Thanks to a semi-automatic protocol [31], PROPHESEE generated a two-class dataset (i.e. binary classification challenge) which includes 12,336 *car* samples and 11,693 non-cars samples (labeled as *background*). In order to have a training set and a test set, the dataset was split in 7940 car and 7482 background training samples, and 4396 car and 4211 background test samples [31]. Consecutive samples have no relation with each other, the dataset is simply a bunch of 100ms samples, representing either cars or background scenes. Time resolution is $1\mu\text{s}$. Figure 2.20 exhibits a few reconstructed images of samples from the original N-CARS dataset. The N-CARS dataset is available for download at <http://www.prophesee.ai/dataset-n-cars/> and supplemental material regarding the generation of this dataset is available at [90].

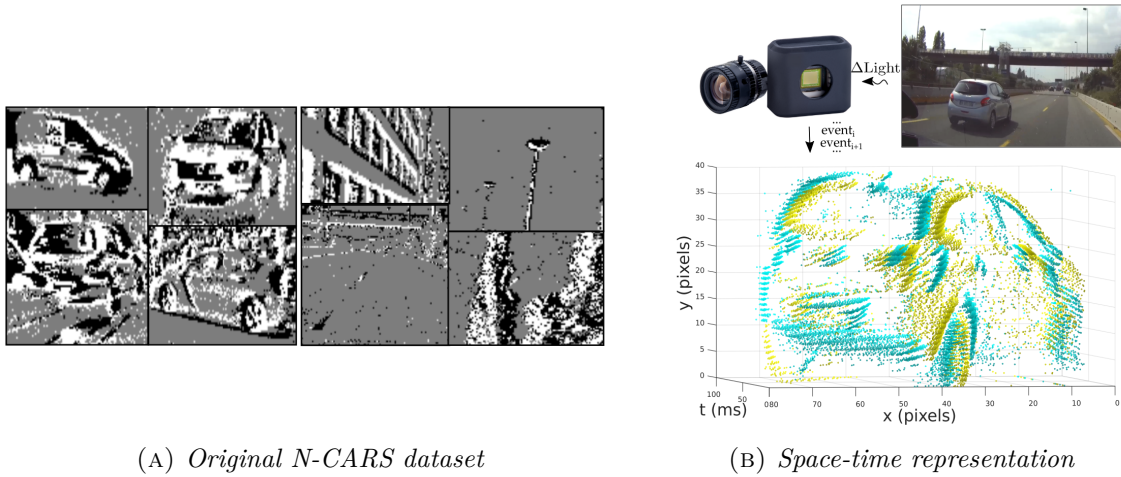


FIGURE 2.20 – (A) **Samples from the original N-CARS dataset**, reconstructed from accumulating spikes over a time interval. Image adapted from [31]. (B) **Space-time representation**: events received from the neuromorphic sensor are plotted through time. Image adapted from [31].

A member of PROPHESEE has very kindly accepted to **share a rescaled and down-sampled version** of the full N-CARS dataset, which is called the *UCLouvain version* in the rest of this work. More specifically, each sample from the original N-CARS dataset was rescaled to a spatial resolution of 50x40 pixels. The temporal resolution was also

downsampled by enforcing a refractory period of 1ms to every pixel, i.e. if two events occurred at the same pixel in a time interval of less than 1ms only the first event was kept and the second was filtered, as can be observed in Figure 2.21. After these processing steps, each sample contains at most 10^4 spikes as shown in Figure 2.22. As shown in Figure 2.23, the distribution of spikes through time is uniform, i.e. there is no specific time window where more spikes are usually concentrated, in opposition to the saccades in the N-MNIST dataset. PROPHESEE obtained about 75% classification accuracy with a FC network (4000-100-2) and about 95.5% with a ConvNet (2 channels input, two Conv layers with 20 and 60 channels followed by two FC layers of 500 and 2 neurons) both trained with backpropagation in the spiking domain. These results were not published but were kindly shared by a member of PROPHESEE. There is no human reference accuracy for the N-CARS dataset.

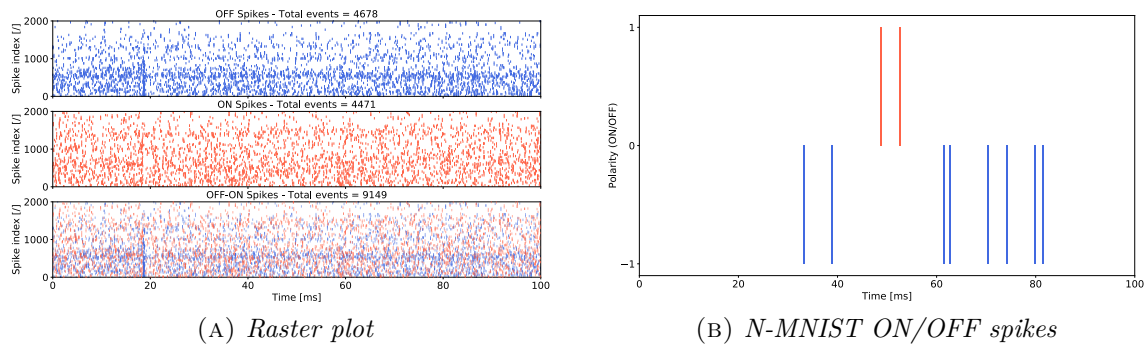


FIGURE 2.21 – (A) *N-CARS input spikes raster plot*, for the 54^{th} sample of the training set (label = 1). (B) *N-CARS ON/OFF spikes*, for the 54^{th} sample of the training set (label = 1) and pixel index = 470, i.e. $[x, y] = [20, 9]$.

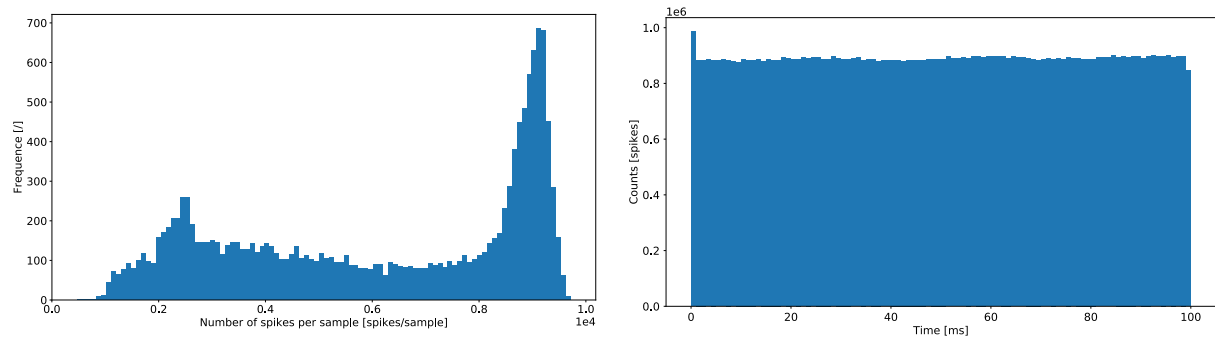


FIGURE 2.22 – *N-CARS spikes per sample distribution*, for the whole training set (UCL version). Mean = 6139 spikes/sample.

FIGURE 2.23 – *N-CARS time distribution of spikes*, for the whole training set (UCL version). The distribution is uniform.

In this work, the UCLouvain version is used instead of the original N-CARS dataset for **two** main reasons. **First**, one drawback of the original version is that all samples *do not* have the same shape, making it less convenient to use. That can however be solved by zero-padding. **Then**, even if both temporal and spatial resolution are better in the original dataset, this results in more spikes to process and thereby more simulation time. Figure 2.24 gives two different reconstructions of a sample labelled as a car (label = 1). As a matter of fact, it is less obvious to recognize a car compared to Figure 2.20. This is the main *drawback* of the UCLouvain version: there is a valuable loss of information due to the rescaling and downsampling.

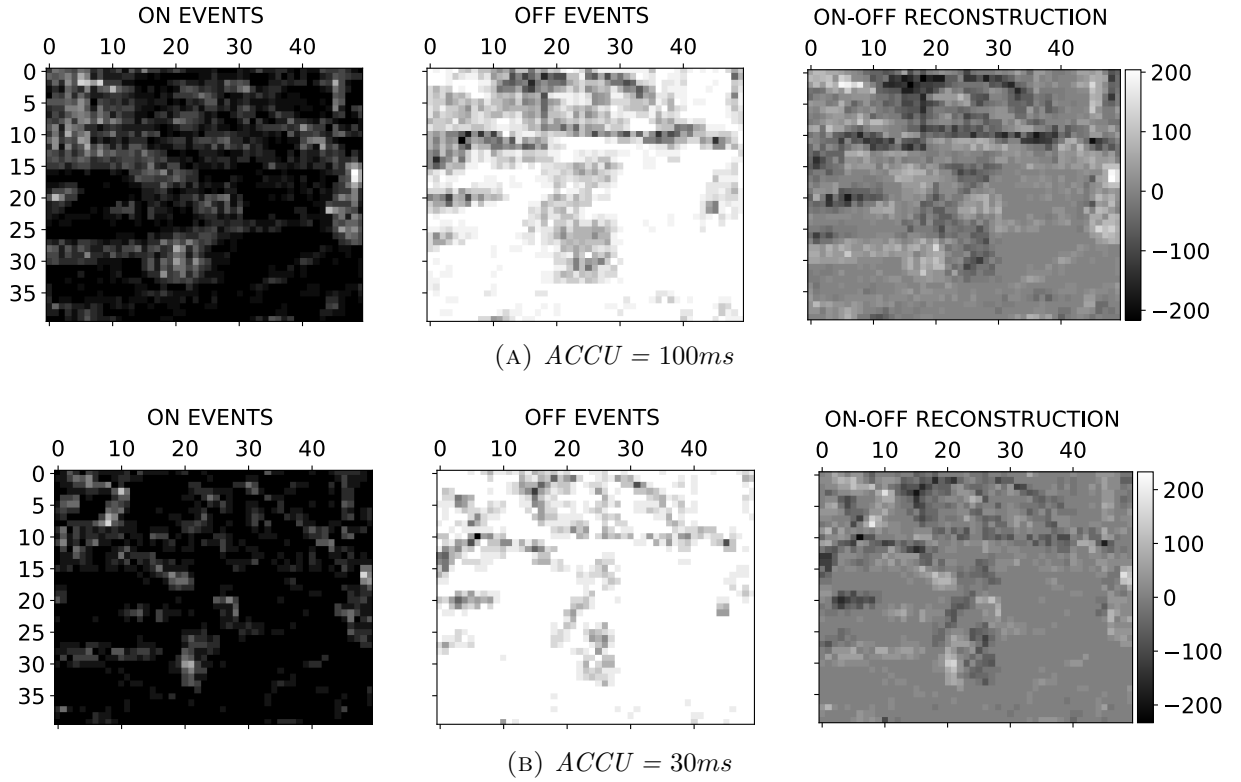


FIGURE 2.24 – *N-CARS frame-based reconstruction* based on accumulation of input spikes and ON-OFF reconstruction. This is done for the 54th sample of the training set (label = 1 = car) with an accumulation window of (A) 100ms and (B) 30ms.

2.5.5 Other neuromorphic datasets

There exists other neuromorphic datasets but as they will not be used in this work, only some references are given for readers information (Pips dataset [29], Poker-DVS dataset [3], Rotating barrel dataset [35], N-Caltech101 dataset [87] to name a few).

State of the art: training SNNs

This chapter introduces the basics of neural information theory that will be useful for the rest of this work. Then it focuses on the main differences between training ANNs and SNNs and finally, it presents a literature review about how SNNs can be trained.

3.1 Neural information processing

This section first gives some basic theoretical foundations about brain information processing in order to better understand the parallelism between biological networks and SNNs. Then, some well-known models for neuron behaviors and learning process are given.

3.1.1 Overview of biological neural networks

The **two key units** in biological neural networks are **neurons** and **synapses** as shown in Figure 3.1. The human brain is estimated to contain around 10^{11} neurons which are individual units interconnected together through axons, dendrites and especially adaptive synapses (roughly 10^{15} in the human brain) [12, 93, 94]. Typical synaptic fan-ins can go from 100 up to 10^4 synapses per neuron [66, 84]. A synapse connect the end of an axon from the pre-synaptic neuron to a dendrite of the post-synaptic neuron.

To **transmit information** from one neuron to another, the brain uses electro-chemical signals called *action potentials* (AP) or *spikes*, as shown in the first subplot of Figure 3.1. An AP is a fluctuation of about $100mV$ membrane potential that occurs over roughly $1ms$ [95]. This is actually one of the *most important properties of neurons*: their membrane is electrically excitable [96]. This electrical signal is of interest in the nervous system as it can be monitored: at rest, the potential inside the cell membrane is about $-70mV$ with respect to the external surrounding of the neuron [95], which is called the resting membrane potential $V_{resting}$. The cell is in a polarization state. When the running sum of changes in the membrane potential crosses a threshold (usually around $15mV$ above $V_{resting}$ as shown in the second subplot of Figure 3.1 [95]), an AP is generated. It is due to rapid changes of chemical concentrations caused by ions channels activity (mainly Na^+ , K^- and

Cl^-) in the cell membrane which induces sudden (de-, hyper-)polarization [95, 96]. This causes electric current to flow down the axon to the axon terminals (as fast as 120 [m/s] for myelinated axons [96]) where voltage-gated Ca^{2+} channels open. Ca^{2+} ions flow in the pre-synaptic terminal buttons and eventually stimulate vesicles to release neurotransmitters in the synaptic cleft by exocytosis [97]. These neurotransmitters propagate through the synapse and bind with receptors (chemical gates) present on the dendrites of the post-synaptic neuron. This allows ions to flow in and modify the post-synaptic potential (PSP). Depending on the type of neurotransmitters, the PSP can increase (excitatory, EPSP) or decrease (inhibitory, IPSP) [95]. The synaptic cleft is usually too wide for the direct spread of current from one cell to another: the action potential is prevented from passing electrically from one neuron to another which allows for all the chemical processing to take place in the synapse [97]. The absolute refractory period is the time interval (a few milliseconds) just after a spike over which the neuron cannot fire again. Both spatial and temporal summations are observed in the brain [97]. Spatial summation of the PSPs occurs when several different pre-synaptic neurons fire the same post-synaptic neuron, leading to a rapid modification of its membrane potential. Temporal summation occurs when the same pre-synaptic neuron fires multiple times over a short period of time [97]. As biological neurons are **dynamic cells** dealing with noise and randomness [97], they have stochastic behaviors. It has been observed that biological neurons could have up to 20 different behaviors, also called the *Izhikevich behaviors* [7, 81, 98].

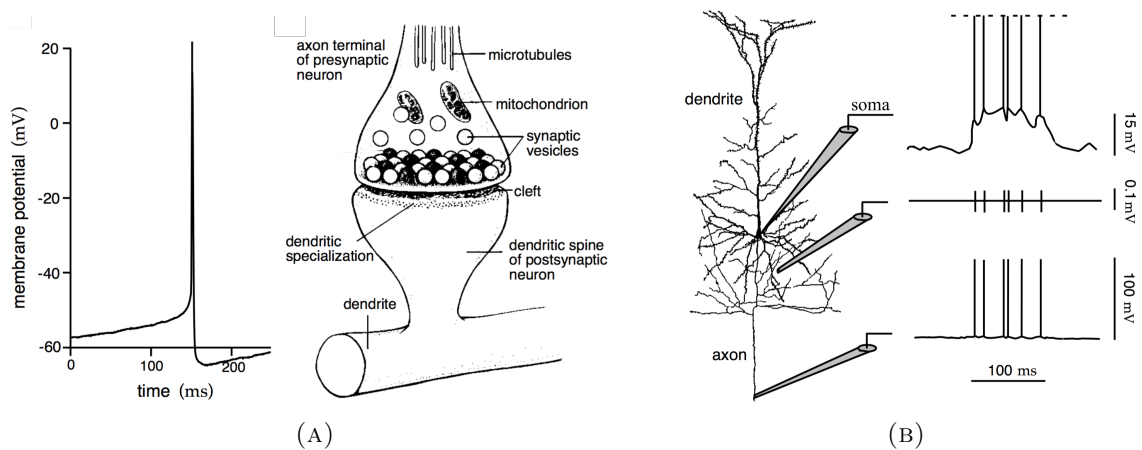


FIGURE 3.1 – (A) **Action potential measurement and synapse illustration.** On the left part, an AP recorded *in vivo* from a cultured rat neocortical pyramidal cell. On the right part, a schematic representation of a synapse. Adapted from [95]. (B) **Biological neuron illustration and firing measurements.** The dendritic tree is a branching topology specific to neurons, allowing to receive inputs from multiples other neurons through synaptic connections as shown in (A). Adapted from [95].

3.1.2 Neural coding

Neural coding is about **understanding how information is encoded in the nervous system** (network of interconnected neurons). It has been observed that all action potentials are the same: they follow a *all or none* law and when an AP is triggered, it is indistinguishable from others [84, 96]. Accordingly, *no information is contained in their shape*. Instead, the neural code seems to be carried in the rate of firing, the population of neurons that have spiked in response to a stimulus or in the pattern of spike times [99]. Even if many behaviors are still left unexplained in the brain, it is widely admitted that firing rates and temporal patterns of spike trains take a major role in carrying information about stimuli and computations [52]. However, an intense debate exists within the neuroscience community regarding whether biological neurons use rate or temporal codes and even on the definition of these terms [84, 95]. This disagreement would be related to whether or not correlation exists between successive spikes or different neurons, i.e. if this correlation could carry valuable information [95]. We redirect the reader to [95, 100, 101, 102, 103, 104] for further information on neural coding and on other types of spike codes.

3.1.2.1 Rate-based coding

The basis of rate coding comes from the observation of the relationship between the *stimulus intensity* and the *neuronal firing rate* [84, 105]. In rate-based coding, what matters is the **average rate of firing** (i.e. *spikes/s*) [87]. Individual timing of spikes is *not* relevant here. For instance, motor neurons seem to use that encoding: the higher the firing rate, the higher the contraction of the muscle [103]. The average could be done over time or over several repetitions of an experiment [99].

In order to **simulate** some neuron behaviors, a *Poisson process* is often used to generate spike trains [3, 4, 11, 41, 51, 84, 85, 86]. It is a process where events are assumed to be statistically independent [95] which gives a fairly good approximation of stochastic neuronal firing [84, 95]. That explains why a Poisson process is often used to convert the pixel illumination into a spike train: it produces *in average* a spiking rate proportional to the illumination of pixels while introducing noise. However, in its simplest form, a Poisson process cannot simulate bursts of events or refractory period [84].

3.1.2.2 Population coding

For that coding scheme, the information is contained in the **set of neurons** that are active in response to a particular stimulus [87]. It is a powerful way to understand which neurons are involved in a specific system (for instance, tuning curves for V1 neurons in the visual cortex) [84, 99].

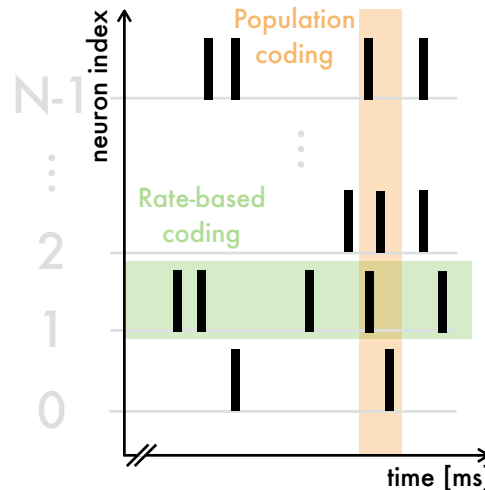


FIGURE 3.2 – *Rate-based and population coding, illustration.*

3.1.2.3 Temporal coding

It has been observed that neurons activity in the brain is really **sparse**: only around 1% (up to 4%) of the neurons are thought to be active at a time [72, 106]. Therefore, coding in the brain cannot be explained only by rate codes [72, 73]. In fact, as neural activity has been shown to be very costly for the brain [106], it has to rely on codes that can embed a valuable amount of information while using a few active neurons [107].

Temporal coding is a code that cares about individual timings of spikes, in opposition to rate-based coding. In fact, information is contained in the **precise (relative) timing of spikes** [33, 100]. Temporal coding holds generally detailed information about the phase of a periodic signal and is thought to be important in the sound localization pathway for instance [108, 109]. Latency coding is a form of temporal coding where information is contained in the time of first spike [3].

3.1.3 Synaptic plasticity

Learning in the brain is thought to occur mainly through the **plasticity of synapses** [17, 69, 110], which is one of their *key property*. They can be seen as a *local* memory that can be modified on-the-fly by the past and the current activity of the network. In other words, the strength of the connection allowing the communication between neurons is encoded in the synapse. This is one of the reason why they are thought to have a *key role* in learning processes [17]: modifying synapses modifies the future activity of the network for either a short or a long period of time (i.e. short-term/long-term synaptic plasticity), which in terms impacts synaptic plasticity [69].

Apart from synapses plasticity, several researches suggest that **propagation delays** in spiking patterns of neurons may be adaptive [69]. Memory and formation of neuronal groups might be related to this time information [69, 98].

Thereby, **learning mechanisms in biological systems are complex mechanisms**: a single neuron is sought by thousands of inputs every minute [111] and has to find out the ones that matter in order to focus on them and pass accurate messages to downstream neurons.

3.1.4 Neural models

Modeling is a really important step to understand, explain and predict behaviors. The models provided in this section allow the trade-off between *complexity* and *versatility*, depending on specific assumptions. These trade-offs will be highlighted in order to show how to select a suitable model for a given application.

3.1.4.1 Leaky integrate-and-fire model

The **leaky integrate-and-fire** (LIF) is a first-order model and one of the simplest and most extensively used neuron models thanks to its computational efficiency [32, 41]. The basic concept of the LIF model is that the neuron has a membrane potential that evolves through time due to incoming excitatory or inhibitory currents and a leakage parameter. Whenever the membrane potential of a neuron crosses its threshold, the neuron fires, i.e. emits a spike. Its membrane potential is then reset to its resting state. This behavior is characterized in Equation (3.1).

$$\begin{cases} V(t) = V(t-1) + L + X(t) \\ \text{If } V(t) \geq \theta_{thresh}, \quad \text{spike and reset} \quad V(t) = V_{resting} \\ \text{If } V(t) < V_{resting}, \quad \text{reset } V(t) = V_{resting} \end{cases} \quad (3.1)$$

where $V(t)$ is the membrane potential, L the leakage parameter, $X(t)$ the summation of all synaptic inputs at time t i.e. $X_J(t) = \sum_i w_{i,J} I_i(t)$, $V_{resting}$ is the resting membrane potential (usually set to 0 but not always) and θ_{thresh} is the membrane threshold [70]. If there is no leak, i.e. $L = 0$, this gives the integrate-and-fire (IF) neuron model. Equation (3.1) is a discrete time version of the LIF model. In continuous time, it can be expressed as

$$C_m \frac{dV(t)}{dt} + \frac{V(t)}{R_m} = I(t) \quad (3.2)$$

where C_m is the membrane capacitance, R_m the membrane resistance and $I(t)$ the input current at time t [86, 112, 113]. The membrane is consequently *not* a perfect insulator due to the leak (time constant $\tau_m = R_m C_m$) which reflects the diffusion of ions that occurs through the membrane when the cell is not at its resting state [86]. The leak has also a stabilizing influence on the neuron behavior [114] and is sometimes used to set up a time constant for the memory of the network [111]. The states of LIF neurons can be updated asynchronously by using only the timing of upcoming events [92]. This model can be implemented with electronic components as presented by the schematics in Figure 3.3.

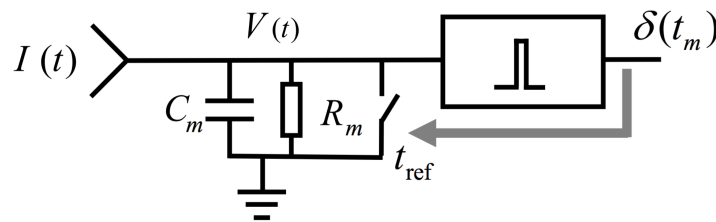


FIGURE 3.3 – *LIF single-cell electronic schematics*. Image adapted from [86, 112].

3.1.4.2 Adapted leaky integrate-and-fire model

The basic LIF model as presented in the previous section has a **main issue** when it is used for rate-based computations that happen close to zero: the output result is **dependent on the order of arriving spikes**, which is *not* desirable in a rate-based context. Specific care should be taken with low firing rates and low threshold values. To illustrate this, let's

imagine a LIF neuron_J with two different inputs I_i weighted by their respective weight w_i . In addition, consider $V_{resting} = 0$. Let's suppose $w_1 = a < 0$ and $w_2 = b > 0$. If I_1 spikes before I_2 , the output is $y_J = b$ as $y_J = (0 - a) + b = 0 + b = b$. However, if the order is reversed, $y_J = b - a$. This simple example highlights the limitations of this model. In practice, this phenomenon can happen really often and introduce a bias on the rate (higher rate). **We propose an adapted model to fix this issue:** setting the lower bound threshold θ_{thresh}^{down} to a negative value, $\theta_{thresh}^{down} = -\theta_{thresh}^{up}$ while keeping $V_{resting} = 0$ and $\theta_{thresh}^{up} = \theta_{thresh}$. Even if this modification does not really make sense from a biological point of view, it mathematically solves the problem *around zero*. As a matter of fact, $y_J = b - a$ in both cases now. Yet, the problem still exists around θ_{thresh}^{down} . V_m could be reset to $V_{resting}$ without firing or simply kept at θ_{thresh}^{down} (chosen in this work). Moreover, when $V(t)$ crosses the threshold, a *reset by subtraction* [11, 115, 116] can be used instead of a *reset to zero* in order to decrease the residual error [11]. Accordingly, the adapted model is given in Equation (3.3). It has been inspired by similar considerations [3, 11, 70]. If $\theta_{thresh}^{down} = 0$ and $\theta_{thresh}^{up} = \theta_{thresh}$, this model is equivalent to the basic LIF neuron model. In this work, the adapted leaky integrate-and-fire model is used without leak, i.e. $L = 0$ and is therefore called **adapted integrate-and-fire model (AIF)**.

$$\left\{ \begin{array}{l} V(t) = V(t-1) + L + X(t) \\ \text{If } V(t) \geq \theta_{thresh}^{up}, \quad \text{spike and reset} \left\{ \begin{array}{l} \text{to zero: } V(t) = V_{resting} \\ \text{by subtraction: } V(t) = V(t-1) - \theta_{thresh}^{up} \end{array} \right. \\ \text{If } V(t) < \theta_{thresh}^{down}, \quad \text{reset} \quad V(t) = \theta_{thresh}^{down} \\ \theta_{thresh}^{down} = -\theta_{thresh}^{up}, \quad V_{resting} = 0 \end{array} \right. \quad (3.3)$$

3.1.4.3 Other neuron models

Even if the two models given previously are interesting because of their simplicity and because of their **computational** abilities, they do not account for chemical pumping dynamics and do not model the AP dynamics. **More complex models exist** and attempt to better capture the biological neuron behavior as the Hodgkin and Huxley model [71, 117], the FitzHugh-Nagumo [118, 119] model and the Izhikevich model [71, 81], among others [104]. However, they are too complex for this application. In fact, they need a lot of computations (solving several differential equations) and it becomes even worse when network of neurons are considered as it is the case in this work. **Consequently, basic and adapted leaky integrate-and-fire models were chosen here.**

3.2 Discrepancies between training ANNs and SNNs

ANNs are usually trained with backpropagation algorithm using a huge amount of labeled training examples [4, 50, 53]. The resulting classification accuracy is genuinely impressive [2, 3], despite the energy required to train that kind of networks [41, 52]. Each sample is presented as a static input and values are propagated through the network, i.e. there is no time information involved during ANN training and neural nodes are analog values [51]. However, some architectures such as recurrent neural networks (RNNs) [2, 46, 120] and long short-term memories (LSTMs) [2, 120, 121] units aim at introducing the notion of time in conventional (deep) neural networks [120]. In fact, they are powerful at extracting patterns from temporal sequences [120]. In conventional neural networks, neurons are usually characterized by a single static and continuous non-linear activation function (e.g. sigmoid, tanh, ...) which is differentiable [52], allowing gradient-based optimization algorithms to be applied.

In opposition to ANNs, **SNNs** are more biologically inspired and they *inherently code more spatio-temporal information* [122]. Unfortunately, traditional ANN training algorithms using gradient-based optimization cannot be used directly to train SNNs [3] because activation functions of spiking neurons are not differentiable (due to threshold condition [3]). However, recent work has proposed some adaptations to make it possible [92]. **Two main trends** are usually observed in the literature in order to train SNNs [3]: either using *training algorithms in the spiking domain* or using an *ANN-to-SNN mapping approach*. These two approaches are truly different and are discussed in the next sections 3.3 and 3.4, respectively. Some **other methods** have been developed to perform machine learning on event-based data but they are not investigated in this work. They are given as supplementary reading for interested readers (e.g. ReSuMe [71, 52, 123], Tempotron [32, 71], Chronotron [71], DART [75], HOTS [30], HATS [31], HFIRST [35], SKIM [58], HMAX [32, 124] among others).

Training (deep) SNNs is in its early phases and still an ongoing challenge: understanding how to carry out training in these networks is consequently an important question, making the exact process still unclear [37]. However, an important part of the training process in both spiking and non-spiking architectures seems to reside in the *feature extraction hierarchy* which discovers more and more complex, abstract and invariant features [37]. Moreover, SNNs have the interesting opportunity to trade off classification error rate against the number of operations, which is not the case in deep conventional ANNs that need a fixed number of operations to reach their classification error rate [11].

3.3 Training SNNs in the spiking domain

In the context of neuronal networks, training is referred as the weights tuning process using a particular updating scheme which is called *learning rule* [125]. In order to capture all the statistics and information contained inside the spike trains, it is desirable *not to leave the spiking domain* when training the network [32]. Some bio-inspired models for learning have been derived as **Hebb’s rule**, **STPD** and **SDSP** learning rules. They are presented in the following sections.

3.3.1 Hebbian learning

One of the most famous learning rule is the *Hebb’s neurophysiological postulate of learning* which describes qualitatively the synaptic potentiation and suggests that synaptic plasticity is related to the **correlated activity of pre- and post synaptic neurons**. A simplified form is known as *"neurons that fire together wire together"* [12, 126, 127] whereas the original statement is [128]:

“When an axon of a cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth or metabolic change takes place in both cells such that A’s efficiency, as one of the cells firing B, is increased”

— D. O. Hebb

However, as this simple model is a *positive-feedback process* (effective synapses are strengthened and strengthened whereas ineffective synapses are weakened and weakened), it can lead to unbounded weights and *stability issues*, e.g. dead or saturated neurons [110, 129]. Using control mechanisms such as multiplicative or subtractive adjustments on the weights can introduce competition and stabilize the learning process [110, 125]. *Co-operativity* (i.e. pre- and post-synaptic highly correlated activity) and *locality* (i.e. change in synaptic weights depends only upon local variables) are **two key aspects** of Hebb’s principle [10, 129].

3.3.2 Spike-timing-dependent plasticity (STDP)

Unlike Hebbian learning where only a high firing rate for both pre- and post-synaptic neurons is of interest, STDP is a **synaptic local modification mechanism** driven by the **relative timing** ($\Delta_t = t_{post} - t_{pre}$) between spikes of a pre- and post-synaptic neuron, which indicates *causality* [111]. If the post-synaptic fires shortly after the pre-synaptic neuron, the weight of that particular synapse has to be strengthened as the pre-synaptic neuron helped the post-synaptic neuron to fire (*causal firing*). In the same way, if the ordering is reversed, the weight of that particular synapse has to be weakened (*anti-causal firing*). Equation (3.4) describes how weights are updated for the most common experimentally observed STDP rule, where w is the synaptic weight, ζ and ξ are constant parameters for learning rates and τ is the time constant (e.g. 15ms [71]) for the temporal learning window [71].

$$\begin{cases} w \rightarrow w + \zeta e^{\frac{-|\Delta_t|}{\tau}} & \zeta > 0, \text{ if } \Delta_t \geq 0 \quad (\text{causality}) \\ w \rightarrow w + \xi e^{\frac{-|\Delta_t|}{\tau}} & \xi < 0, \text{ if } \Delta_t < 0 \quad (\text{anti-causality}) \end{cases} \quad (3.4)$$

It introduces *competition* between synapses as the next spike time will be shifted forward (resp. backward) if the synapse has been strengthened (resp. weakened), which will affect the weights of downstream synapses, even if STDP is a local mechanism. There exists several variations and implementations of the STDP rules: we refer the reader to [17, 130, 131]. STDP is mainly a *positive feedback* procedure that could lead to instability and a bimodal distribution of the synaptic strengths [125].

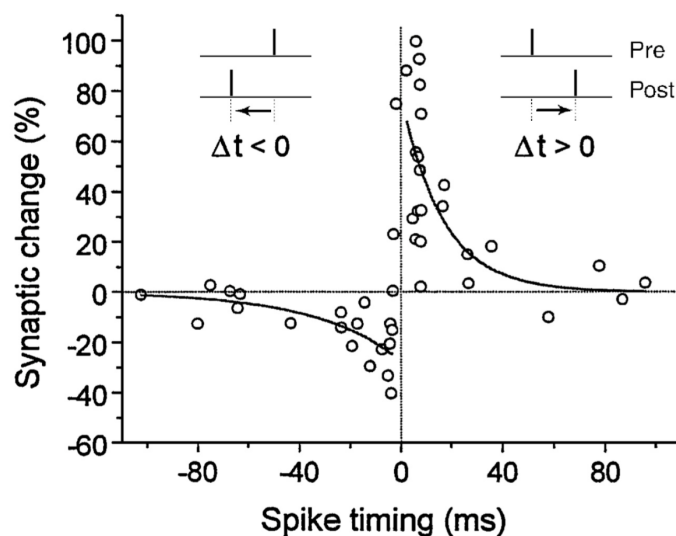


FIGURE 3.4 – *Asymmetric STDP learning window, characterized in hippocampal cultures. Image adapted from [125].*

3.3.3 Spike-driven synaptic plasticity (SDSP)

Whereas STDP relies on relative timing ($\Delta_t = t_{post} - t_{pre}$) between pre- and post-synaptic spike times, the SDSP learning rule proposed by *Brader et al.* [132] uses only **pre-synaptic spike as a trigger for updating the synapse**. In fact, the state of the post-synaptic neuron at the time of the pre-synaptic spike governs weight increment or decrement [7]. Thereby, conditions for synaptic updates can be computed by neurons, which prevents replicating the learning rule inside each synapse, in opposition to STDP. In fact, only the membrane potential V_{mem} and the Calcium concentration have to be compared to corresponding thresholds (θ_m and $\theta_{1 \rightarrow 3}$ respectively) for updating the weights, as shown in Equation (3.5) [7, 132, 133] and in Figure 3.5. The Calcium concentration is a key quantity correlated with the post-synaptic firing activity [10]. Moreover, SDSP embeds overfitting prevention mechanism thanks to stop-learning conditions performed by the Calcium variable [132].

$$\begin{cases} w \rightarrow w + 1 & \text{if } V_{mem}(t_{pre}) \geq \theta_m, \quad \theta_1 \leq Ca(t_{pre}) < \theta_3 \\ w \rightarrow w - 1 & \text{if } V_{mem}(t_{pre}) < \theta_m, \quad \theta_1 \leq Ca(t_{pre}) < \theta_2 \end{cases} \quad (3.5)$$

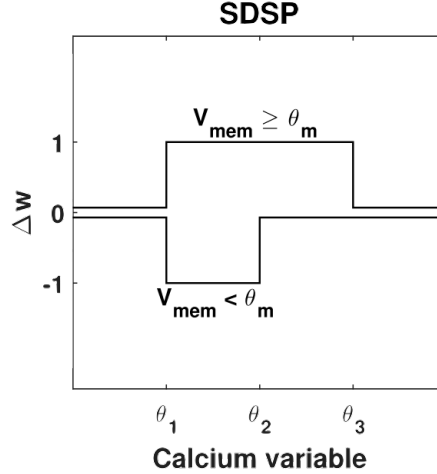


FIGURE 3.5 – **Digital SDSP rule**, proposed by *Brader et al.* [132]. Image adapted from [7].

3.3.4 Spiking backpropagation

Traditional ANNs training algorithms cannot be used directly to train SNNs and the implementation of gradient backpropagation in the spiking domain is especially challenging for reasons detailed in Neftci et al. [10], e.g. weight transport problem, multiplication with derivatives and activation functions and precision among others. However, some recent works have proposed some **adaptations such that backpropagation can be implemented in SNNs**. Several variants exist but only two are briefly presented in this section. We redirect interested readers to [10, 92, 107, 134, 135] for further information.

SPIKEPROP [107] appears as the first attempt to train a SNN using a spiking backpropagation algorithm [71]. Complex non-linear classification can be achieved with this method just as well as rate-coded networks, as it was demonstrated on different benchmark datasets [107]. Moreover, it demonstrates the potency of temporal coding for fast neural information processing by using less neurons than in instantaneous rate-coding [107].

The key idea in [92] is to generate a continuous and differentiable signal such that stochastic gradient descent (SGD) can be applied. Abrupt changes of the membrane potential is treated as noise during error backpropagation and spiking signals are filtered to be then added onto the membrane potential. This supervised method can be applied both on FC and Conv layers and also on SNNs with LIF neurons.

3.4 Offline training of SNNs: mapping ANN-to-SNN

The **basic idea** behind the mapping approach is to use the traditional ML training algorithms on a continuous-valued architecture such that optimal weights can be obtained and then transferred to an *equivalent rate-based SNNs architecture* [4, 10, 70]. In such rate-based spiking networks, the average firing rate of input neurons over a pre-defined time interval is approximately proportional to the magnitude of the original continuous inputs [51]. In the case of images, the magnitude is the pixel intensity of the original ANN inputs. The objective is thereby to shift the background developed in these fields to the event-based domain to get efficient SNNs. Even though this approach is *not* driven by the desire of understanding how the brain computes and processes information, it has demonstrated benchmark records [11, 135] in terms of SNNs accuracy for different architectures (e.g. FC networks, ConvNets [61], Inception-v3 [136], ResNet [137], ...) on several datasets (e.g. MNIST [61], CIFAR-10 [138], ImageNet [139, 140], ...). In this work, the focus is made on the mapping of a CNNs. Consequently, the terminology *mapping CNN-to-SNN* is rather used hereafter.

The next section gives an overview of the general process of mapping whereas the sections 3.4.3 and 3.4.2 brings more details about two specific and sensible steps in this approach.

3.4.1 Method overview

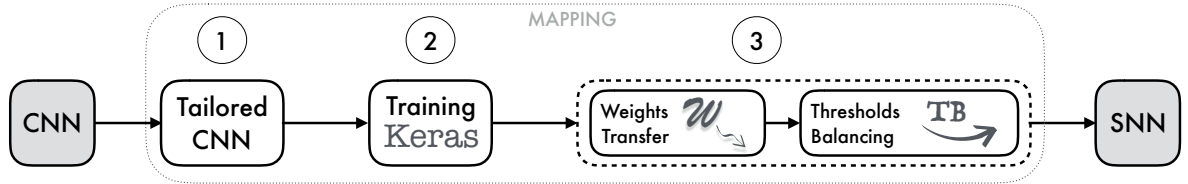


FIGURE 3.6 – *Methodology to convert a CNN to a SNN.*

The mapping approach can be summarized in **three main steps** [70, 4] as presented in Figure 3.6. To understand the first step, it is important to point out that there exists an important *parallelism* between ANNs using Rectified Linear Unit (ReLU) activations and IF rate-based SNNs. The ReLU neuron output y_J is given by $y_J = \max(0, \sum_i w_{i,J} I_i)$ [70] and it can be considered as a first order approximation of the behavior of the IF neurons in the rate-based domain [70, 4]. Accordingly, the output of the ReLU is proportional to the firing rate of an IF neuron [4]. This explains why this activation function is used in order to simulate an IF neuron, instead of the tanh or the sigmoid functions for instance. Moreover, the non-saturating nonlinearity of the ReLU function speeds up the training

time with gradient descent [139]. Accordingly, the **first step** ① of the mapping approach is to *convert* the original CNN into a *tailored CNN* where [70]:

- Inputs values are all positive. It can be done by adding an Abs function before the first layer of the network. This step can be skipped if the inputs is inherently positive, i.g. illumination level.
- ReLU activation function is used for all neurons
- Biases have been removed from all Conv and FC layers
- Spatial max-pooling has been replaced by spatial average pooling

Secondly ②, the tailored CNN is *trained* using conventional training methods (e.g. back-propagation). For the **third and last step** ③, weights are *transferred* $\mathcal{W}_{\text{c}}$ from the tailored CNN to the equivalent SNN architecture while performing a proper re-scaling of the thresholds (i.e. *thresholds balancing* $\xrightarrow{\text{TB}}$ as explained in Section 3.4.3).

This mapping approach works for both FC networks and ConvNets [70] but if the SNN neuron model is changed from the standard IF, the activation function in the tailored CNN needs to be changed accordingly. However, using the AIF model presented in Section 3.1.4.2 does not require to change the activation function as it is an attempt to better implement it in rate-based SNNs. Simulations and results are investigated and discussed later in Chapter 5.

The **main challenge** during the mapping approach is to **avoid a substantial drop in classification accuracy after conversion**. However, a source of accuracy loss in this method is that *only a fraction of the traditional ML tools are available* to implement such networks and algorithms in a spiking structure. Even if some alternatives have been proposed to allow conversion of nearly arbitrary CNN architectures by making available tools as max-pooling, softmax, batch-normalization and inception-modules in SNNs [11], common mappings methodology use a restricted number of tools. For instance, dropout is often used instead of batch-normalization regarding the regularization technique [51]. Another source of accuracy loss is the presence of *hardware constraints* set by the application (i.e. number of neurons, resolution of the synapses, connectivity, ...). Consequently, not every continuous-valued CNN can be mapped onto a SNN. For instance, even if negative values can be represented in SNNs using inhibitory neurons, this come with a high hardware cost, i.e. doubling the number of neurons and consequently the number of connections, the complexity and the power consumption which is not desirable [70].

3.4.2 Converting an event-based dataset to a frame-based dataset

There is still one **key limitation related to this general mapping approach**: how could an ANN be trained if the only available input is a neuromorphic dataset ? In fact, the problem is straightforward if a framed-based dataset is used to train the tailored CNN and is then converted to a spiking dataset as explained in Section 2.5.1. However, the problem is far more extensive if **only a spiking dataset is available**. Indeed, a *spikes-to-frame conversion* is required for training the tailored CNN but it requires to leave the spiking domain.

How to carry out the conversion is also a very pertinent question. The **main objective** behind that dataset conversion idea is to create a frame-based dataset that embeds as much as possible information from the spiking dataset. This should provide representative data to the tailored network in order to learn the weights that will be then transferred to the SNN to perform the inference. This is the main difference with Rebecq et al. [39] that proposes to reconstruct images (videos) from a stream of events at the best possible quality in order to apply conventional machine learning algorithms afterwards. Even if this approach seems to lead to impressive classification performances [39], inference is *not* performed in the spiking domain, which is of interest here. In addition, training the tailored network with these reconstructed images will not train weights properly for the spiking network as it is *not directly* representative of the spiking data.

A possible solution is to **reconstruct a gray-level image by accumulating ON/OFF events during a pre-defined time interval dt** (i.e. accumulation window) [6, 32, 87] as presented in Figures 3.7 and 3.8. This reconstruction method assigns a pixel intensity proportional to the spike rate of the corresponding pixel and is often called *accumulated spike count* or *histogram of events* in the literature [4, 6, 27]. If the accumulation window is small enough (i.e. order of ms or tens or ms depending on the dynamics of the scene), the sensor and the network are assumed to use both a rate-based encoding over small time bins. It is a good practice to *normalize* the spike count $C_{dt}^{g,p}$ by the maximum number of spikes that have been accumulated for each reconstructed image [87] in order to avoid introducing a bias related to the size of the accumulation window, as given in Equation (3.6) (adapted from [87]). Each sample p of the dataset is defined by the spike train $s^{g,p} = \{t_1^{g,p}, \dots, t_n^{g,p}\}$ for each of its pixels g [87]. This can be carried out for every sample p of the original dataset.

$$\tilde{C}_{dt}^{g,p} = \frac{C_{dt}^{g,p}}{\max_z \sum_i^n t_i^{z,p}} \quad \text{where} \quad C_{dt}^{g,p} = \sum_i^n t_i^{g,p} \quad (3.6)$$

Then, the reconstructed dataset can be used to train the tailored network with backpropagation. However, these reconstructed signals cannot embed all the information in spike trains that goes beyond ideal mean rate modeling [92]. How to choose the duration dt and the position of accumulation window within the whole duration of each spiking sample are important questions as well. The first subplot of Figure 3.7 highlights the fact that the reconstructed image is affected by these two parameters.

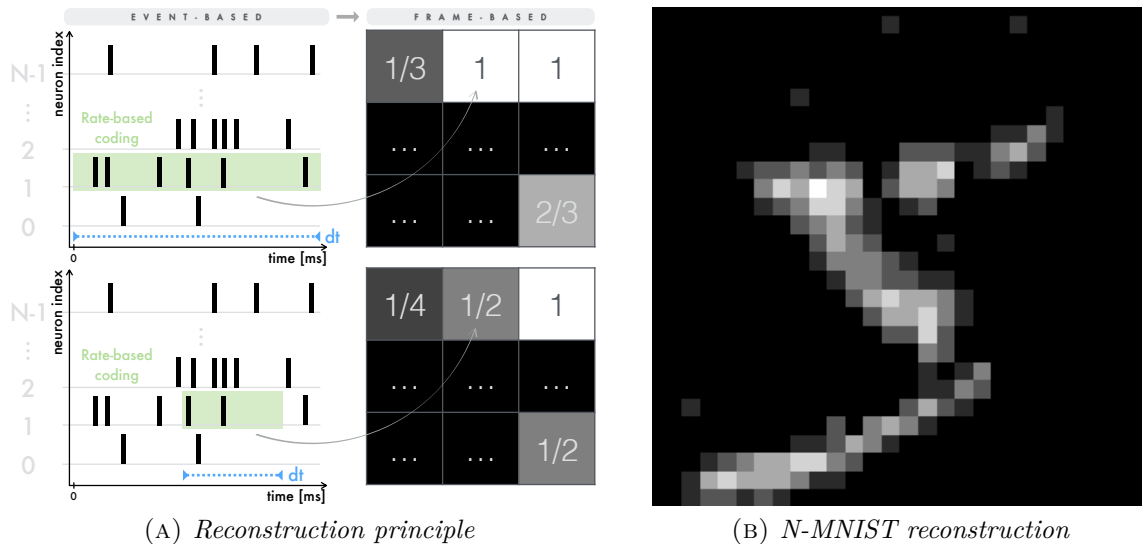


FIGURE 3.7 – (A) Illustration to explain the **gray-level reconstruction** principle of images by accumulation of spikes (only ON spikes are considered here). (B) **Result of a reconstruction gray-level image** using the first sample of the N-MNIST training set with an accumulation window $dt = 50ms$ (only ON spikes are considered).

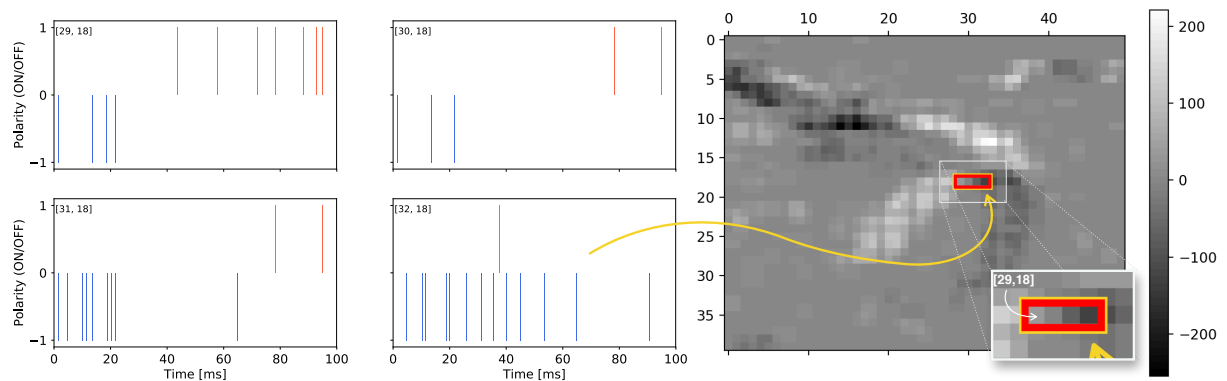


FIGURE 3.8 – Example of **gray-level (contrast) reconstruction** for the 19th sample of the N-CARS training set, label = 1 (car). Accumulation window is 100ms, i.e. the whole sample duration. The ON-OFF corresponding reconstruction is shown on the right. The spike trains for four consecutive pixels (e.g. $[x, y] = [29 \rightarrow 32, 18]$) are represented on the left.

Another solution is to **reconstruct a gray-level image where recent spikes have more intensity than older spikes** [6], as shown in Figure 2.9. This method has the advantage to show the relative event time but as it focuses on the timing of spikes, *this is not a rate-based encoding*. Hence, it appears less attractive for the mapping method as the target architecture is a rate-based SNN.

The choice of the **reconstruction method** is an important step in the mapping approach: the more it can embed spatio-temporal information from the spiking dataset while ensuring an accurate translation of the continuous activation, the better.

3.4.3 Threshold balancing algorithm

Given that ReLU in the CNN have been replaced with IF neurons in the SNN, the **remaining loss of performance** after the conversion can come from **three factors** [4] which can be addressed by *thresholds balancing* $\xrightarrow{\text{TB}}$ [4]:

- The input firing rates are too low such that the IF units do not cross their threshold, leading to a lower effective output (i.e. dead neurons)
- The input firing rate is so high that the IF units output too much spikes (i.e. saturated neurons), i.e. their firing rate is higher than they should be. This could also happen if the input weights are higher than the neurons threshold (if thresholds have been wrongly balanced)
- A specific feature set can be over- or under- activated due to the non-uniformity of spike trains (probabilistic nature of the spiking input)

Thereby, a proper re-scaling of the thresholds is necessary to ensure an accurate translation of continuous-valued neuron activations into firing rates of spiking neurons, which makes the **quality of the mapping** highly dependent on the **ratio between the neuron threshold and the synaptic weights** [51]. We propose the following notation (that will be used throughout this work): the thresholds vector Θ_m contains the firing thresholds $\theta_m^{layer_l}$ for each layer l of the network. For instance, a SNN with q Conv layers, each followed by an average pooling layer and r FC layers has a thresholds vector Θ_m defined by Equation (3.7).

$$\Theta_m = \underbrace{[\theta_m^{Conv_1}, \theta_m^{Pool_1}, \dots, \theta_m^{Conv_q}, \theta_m^{Pool_q}]}_{\text{Conv-Avg layers}} \underbrace{[\theta_m^1, \dots, \theta_m^r]}_{\text{FC layers}} \quad (3.7)$$

The **most formal way** to achieve a proper re-scaling is to use either a weight-threshold balancing algorithm (e.g. SPIKE-NORM proposed by Sengupta et al. [51] or *data-based normalization* algorithm proposed by Diehl et al. [4]) to obtain the best Θ_m or to develop analytic equations to obtain optimal balancing, i.e. *minimizing a specific cost function*.

We **propose** two other approaches in order to balance the thresholds.

- Although this is less elegant while consuming more resources and so arguably less effective (no guarantee to obtain an optimum), a **grid search algorithm** can be used to find Θ_m . It is quite simple to implement but the computational cost is quite high as inference has to be performed over a significant fraction of the training set and for several combination of thresholds, i.e. a grid search of Θ_m . This allows to identify good candidates for Θ_m , i.e. leading to the best classification accuracy on the grid search. Then, the best Θ_m is used to perform inference over the test set.
- Another possible approach could be to tune thresholds such that the output of each layer of the SNN is as close as possible to the output of the corresponding layer in the ANN for a subset of training samples. Images can be reconstructed in the spiking network using the method presented in the previous section and then compared with their counterparts in the continuous network, using an error metrics such as PSNR or RMSE, Appendix A.2.5. For the rest of this work, the reconstructed image used to represent the output of a channel in a particular layer will be called an *activation map* as it is a visual representation of the activation output where each neuron is represented by its corresponding pixel. The brighter the pixel on the activation map, the higher the activation corresponding to that neuron. In order to allow fair comparison between activation maps, each activation map is *normalized* by its maximal value in order to have values in $[0, 1]$.

All these approaches could be done with respect to the weights as well (*weights-normalization*). What matters is to have the **right balancing** (i.e. the right ratio) between the firing threshold and the input weights as it determines the output firing rate of the neuron [4] for given input spikes. Consequently, if balancing on thresholds is carried out (i.e. find the optimal Θ_m , *threshold-balancing*), weights can be imported $\mathcal{W}_\text{c}$ after learning without any modification. Both operations are equivalent from a mathematical point of view. Simulations and discussions are presented in Section 5.1.4.

Methodology

This chapter aims at motivating our choices regarding the chosen SNN architecture and the learning method as well as giving an overview of the general strategy that we proposed to reach our objective. In addition, it aims at presenting the **two main tools** that we have developed for this work, i.e. a *synthetic spiking dataset* (SSP-MNSIT) and a *spiking simulator*.

It is important for the reader to keep in mind that the ultimate objective was to use that SNN structure on the N-CARS dataset (UCLouvain version) to reach good classification accuracy (i.e. $> 80\%$, set arbitrarily). At the same time, the proposed solution had to account for hardware constraints as presented in Chapter 2 such that this work could be implemented on real hardware.

4.1 Choosing an appropriate SNN architecture and a training method

In the light of Sections 3.2, 3.3 and 3.4, it is time to detail our choices regarding the **SNN architecture** that was used for this work and most importantly, which approach was followed to **train** the SNN.

4.1.1 SNN architecture

Regarding the **architecture**, a SNN using only FC layers could not be used due to the limited fan-in and connectivity of both ODIN (256) and MORPHIC (512). As N-CARS dataset (UCLouvain version) has a 50x40 input shape, it cannot fit on the UCLouvain neuromorphic chips. A solution could have been to split the input in four patches (e.g. approach investigated on MNIST by Frenkel et al. [20]), resulting in four channels of shape 25x20 that could fit on the four cores of MORPHIC. However, this is without accounting for the two polarities (ON/OFF) inherent to spiking datasets, which double the size of the input shape making this solution impossible to implement in practice given these hardware constraints. *Rescaling* the N-CARS dataset could have been a solution

but as the version used in that work was already a rescaled and downsampled version of the original N-CARS dataset, too much information would have been lost. Consequently, a FC network could not be used alone to classify samples from N-CARS because this architecture could not fit on MORPHIC due to hardware constraints.

A way to **decrease the dimension of the input shape** on ODIN or MORPHIC was to use *convolutions*. In fact, using a succession of convolution and pooling layers allows both for features extraction and dimension reduction (information in the input is thereby not lost). Neuromorphic chips dedicated to convolution exist (e.g. [141], [142], [143], [144]) and are preferably used for Conv layers because cross-bar architecture are *suboptimal* to implement convolutions as a massive amount of the synapses are not used (synaptic weights set to zero) and many neurons are required due to the multiple channels. It is also possible to simulate neuromorphic convolutions on Field Programmable Gate Array (FPGA). Even if it is not as efficient as neuromorphic chips, this hybrid solution has the advantage to be easier to implement and fully digital, which allows for developing a simulator in order to verify the processing of the whole network. **This solution was chosen here as a first step towards a possible hardware implementation.**

The commercial **low-power consumption FPGA device** SPARTAN-VII (Appendix A.3) from the XILINX company was picked up in order to set hardware constraints on memory and power consumption. A similar choice was made by Corradi et al. [133] which suggests that this device is an optimal development platform for data logging and AE routing [133]. In the first instance, the SPARTAN-VII (XC7S100) was considered. It is the largest model in this family. It has a total block RAM of 4.320Kb and is optimized for low cost, low power, and high I/O performance. It offers roughly 50% power reduction compared to previous SPARTAN families, while at the same time delivers a 30% performance improvement. The SPARTAN-VII uses the MICROBLAZETM 32-bit RISC processor which supports floating-point arithmetic. For that reason and in a first approach, simulations in Chapter 5 were carried out with 32-bit floating point resolution for the Conv layers. Then, more constraining limitations are addressed in Section 5.4. Supplementary material about the SPARTAN-VII is given in the data sheets, Appendix A.3.

A **reasonable architecture** that could be implemented was a cascade of two Conv layers, each followed by a subsampling layer (i.e. average pooling layer). To take the spike polarities in consideration, the input was seen as a 2-channels input. Only the FC layers were supposed to be implemented on MORPHIC (only one of the four cores was used). The full network is shown in Figure 4.1.

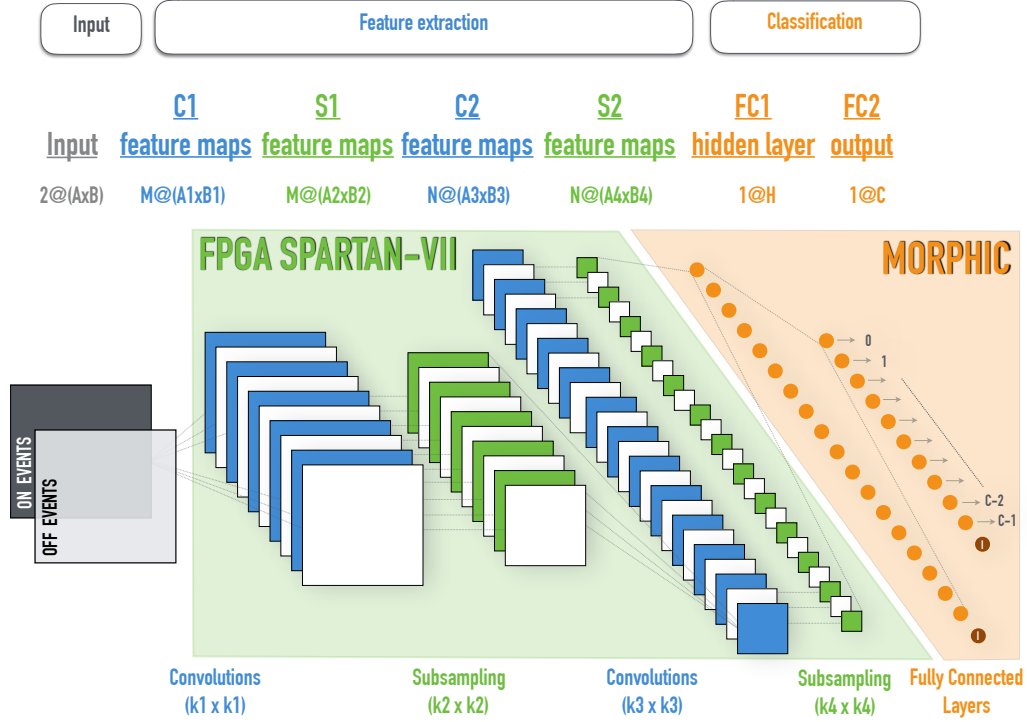


FIGURE 4.1 – **General SNN architecture** used throughout this work. A and B are set by the shape of the dataset. M and N are the number of channels for the convolution layers whereas H is the number of neurons in the FC hidden layer. C is set by the number of output classes and H must be set to fit on MORPHIC accordingly to Equation (4.1). A_i and B_i depend on the kernel sizes k_i chosen for Conv and subsampling layers. All connections are not shown.

To make sure that the output of the second subsampling layer could be used as input on MORPHIC, it was crucial that Equation (4.1) was always fulfilled. Furthermore, synaptic weights on MORPHIC are constrained to $\{0, 1\}$ whereas the weights obtained after training the tailored architecture using BinaryDense layers on KERAS are $\{-1, +1\}$: introducing a layer-wise inhibitory neuron in the FC layers allowed to compensate for that limitation as proposed by Frenkel et al. [20]. All neurons in the previous layer were connected to the input of this inhibitory neuron. It has a threshold $\theta_{thresh}^{up} = 2$ and when it spikes, all the neurons in the same layer have their membrane potential lowered. This explains the term $+2$ in the second Equation (4.1) as two inhibitory neurons are required (one for each FC layer), as indicated in Figure 4.1 by the symbol $\textcircled{1}$.

$$\begin{cases} N \times A_4 \times B_4 < \text{MORPHIC}_{fan-in} = 512 \\ H + C + 2 \leq \text{MORPHIC}_{core-capacity} = 512 \end{cases} \quad (4.1)$$

4.1.2 Training method

Regarding the **learning method**, the two main possibilities discussed in Section 3.2 are either *training the SNN in the spiking domain* or using a *mapping approach*. Both are still ongoing research fields but the latter guarantee reasonable performances at the end of the process, if good classification accuracy is obtained with a tailored conventional network and if mapping is well performed. In addition, N-CARS samples have quite short duration (i.e. 100[ms]) and it seemed possible to reconstruct images from this dataset in order to use a rate-based spiking network. Moreover, PROPHESSEE had already investigated learning in the spiking domain (with both FC and Conv architectures) which led to 75% and 95.5% classification accuracy respectively. Consequently, the **CNN-to-SNN mapping approach was chosen for that work**, which thereby focus on **offline training**. To the best of our knowledge, it was the **first time** this method was applied on **N-CARS**.



Summary

Architecture: SNN given in Figure 4.1

Training method: CNN-to-SNN mapping (offline training, KERAS)

The **challenge** was thereby not only to find out how to *extract* meaningful content from a real-world spiking dataset in order to *train* the weights with backpropagation on KERAS but also to *make sure* that these weights gave the network the ability to perform *inference* accurately *at a spiking level*.

4.2 Methodology overview: a three-step strategy

As N-CARS is a real-world spiking dataset, it has its own dynamics and its content might be quite difficult to handle to an undiscerning eye. Consequently, we proposed a **three-step strategy** in order to gain more insights on how SNNs are generally working and how to extract useful content to finally understand how to classify correctly new samples.

1. First, a synthetic spiking version of the MNIST dataset (i.e. **SSP-MNIST**) was generated and used to develop and validate the smooth functioning of the spiking simulator that produced all results in the rest of this work. In order to allow fair comparison between the continuous and the spiking domain, no noise was introduced during the spiking encoding of the SSP-MNIST dataset. For this first step, the tailored network was trained on the conventional MNIST dataset and the weights were used to perform inference in the spiking domain using the SSP-MNIST dataset.

2. Then, the **N-MNIST** dataset was used to understand what happened when a more realistic spiking dataset was used with the spiking simulator. That intermediate step from SSP-MNIST to N-CARS was crucial as it highlighted how to better account for dynamics and polarities inherent to spiking dataset.
3. Finally, bringing together tools and lessons from the two previous steps led to a stronger method that performed good classification accuracy (i.e. $> 80\%$) on **N-CARS**.

Figure 4.2 summarizes how the three datasets were used in this step-by-step strategy. Results of the mapping for each step are given in Chapter 5. Analyzing Figure 3.6 at the same time is recommended to have a better view of the general strategy. Details about the SSP-MNIST dataset are given in the next section. The FPGA implementation and MorphIC implementation boxes in Figure 4.2 are shown here to emphasize on the global hardware framework.

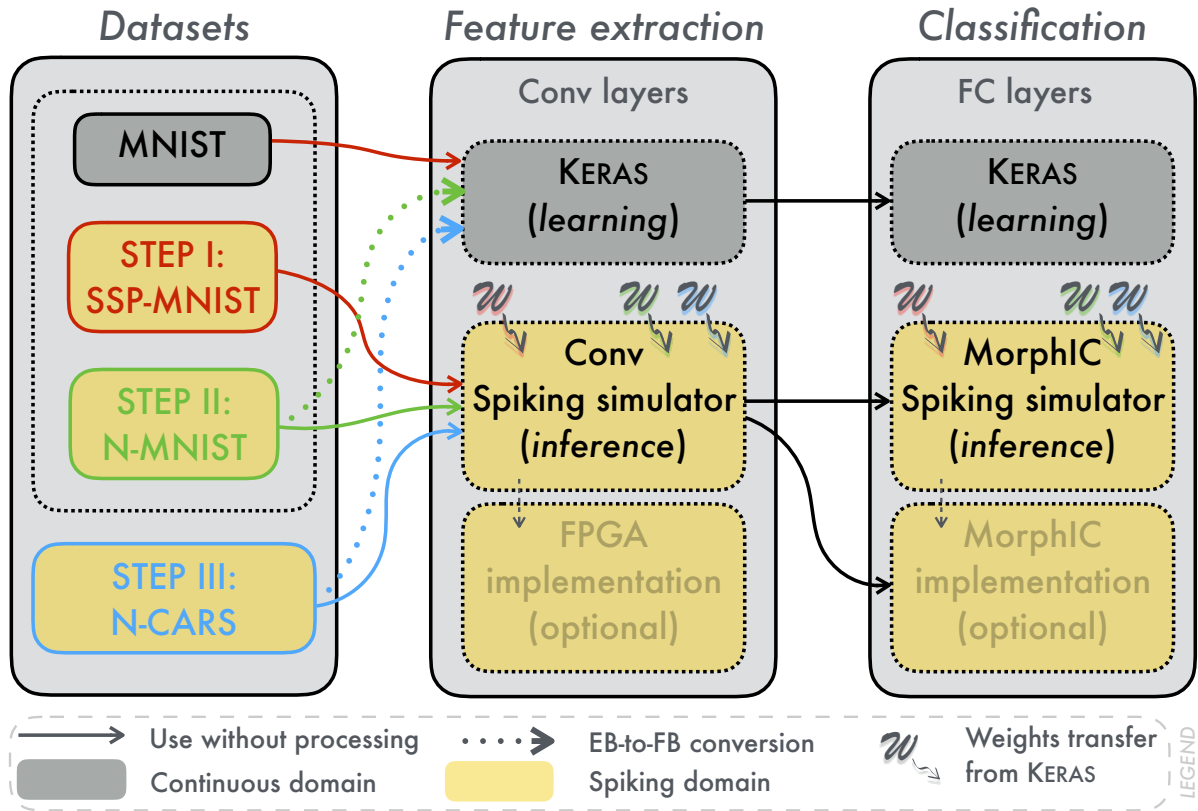


FIGURE 4.2 – *Three-step strategy overview.*

4.3 Generation of a synthetic spiking dataset: SSP-MNIST

SSP-MNIST stands for SyntheticSpiking-MNIST and was created by applying the first method presented in Section 2.5.1, i.e. converting the original MNIST dataset into an event stream through digital data processing. The reader could wonder why such a dataset has been generated whereas this method is not the best way to generate a neuromorphic dataset. There are **two mains reasons**.

The first reason is that the main purpose of SSP-MNIST was to encode *perfectly* the content of MNIST to a spiking rate-based dataset such that it was possible to validate the smooth functioning of the spiking simulator developed for this work. Thereby, the objective was *not* to outperform an existing spiking classification result on MNIST but rather to use this dataset as a *verification tool*. Usually, a Poisson process is used to generate spike trains based on the pixel illumination [145]. For the reason given here above, another process for the spike generation was chosen in order *not* to introduce external noise into the system. Creating this synthetic dataset gave thus the *control on the spike generator method used for the conversion*. For SSP-MNIST, the number of spikes generated for each pixel was equal to the illumination of that particular pixel, i.e. ranging from 0 to 255 (8 bits resolution). Spike times were chosen randomly in a given time window and for a particular time resolution as the only thing that mattered here was the mean-rate of firing over the sample duration. Only the ON polarity was used, as shown in Figures 4.3 and 4.4. The perfect spiking encoding of SSP-MNIST can be assessed by observing Figure 4.3.

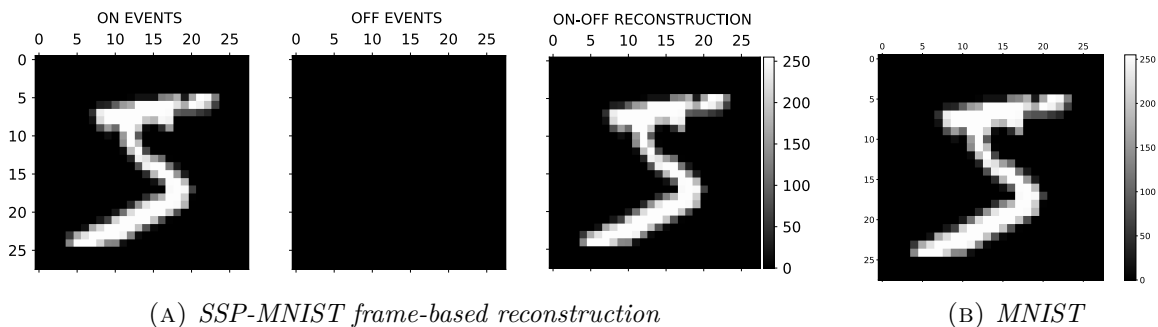


FIGURE 4.3 – (A) **SSP-MNIST frame-based reconstruction by accumulation of input spikes**, for the first sample of the training set (label = 5). Accumulation time is 100 [ms]. (B) **First sample of the original MNIST training set**, (label = 5).

The *drawback* of this generation process is that it leads to both a huge number of spikes per sample and to non-biological firing rates (e.g. $\approx 2550Hz$ for a $100ms$ time window) as can be seen in Figure 4.4. In fact, despite the fact that SSP-MNIST has only one single polarity, it has in average more than *six times* spikes per sample than N-MNIST as can be observed in Figure 4.5. As an order of magnitude, the SSP-MNIST test set file (i.e. 10^4 samples) is about $8.5GB$, which is quite heavy.

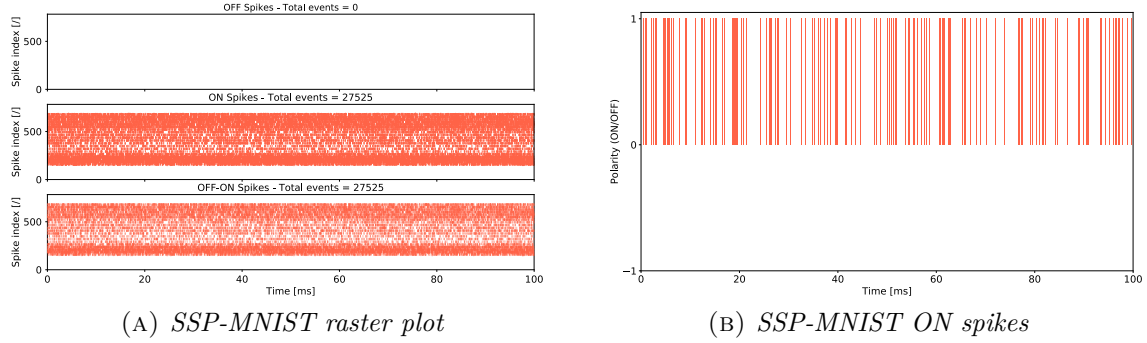


FIGURE 4.4 – (A) **SSP-MNIST input spikes raster plot**, for the first sample of the training set, i.e. label = 5. (B) **SSP-MNIST ON spikes**, for the first sample of the training set, i.e. label = 5 and pixel index = 269, i.e. $[x, y] = [17, 9]$. There are 154 ON spikes as the value of that pixel in the original dataset is 154.

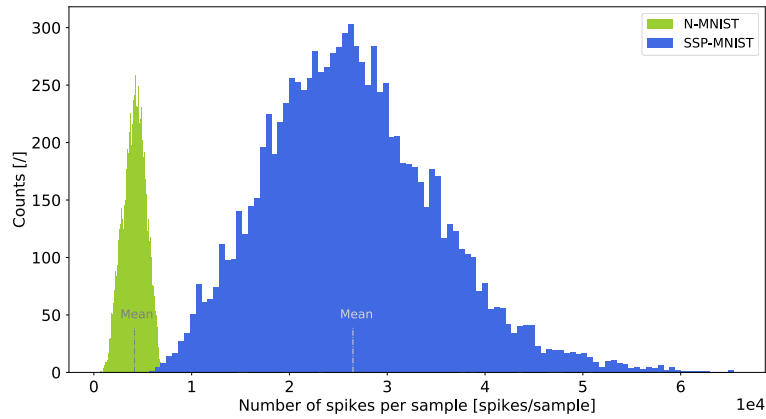


FIGURE 4.5 – **Comparison between N-MNIST and SSP-MNIST spikes per sample distribution**, for the whole test set. $Mean_{N-MNIST} = 4158$ spikes/sample, $Mean_{SSP-MNIST} = 26492$ spikes/sample.

The second reason is that SSP-MNIST was structured in the same way as the N-CARS dataset such that the work done on this dataset could be transferred easily to N-CARS. This includes data structure for interfaces, samples duration (i.e. $100ms$) and time resolution (i.e. $1\mu s$) but not the 2-channels input. Even though there is absolutely no guarantee that a powerful architecture on the SSP-MNIST dataset will reach good classification accuracy on the N-CARS dataset, a functional architecture on the first dataset can be used as a basis for the second one.

4.4 Spiking simulator

Developing this tool was a *major achievement* of this work as this spiking simulator produces all results presented in Chapter 5. It allows easy and straight mapping from KERAS once learning was carried out properly using dedicated scripts that we wrote. This simulator was developed for mapping purposes and is thereby only capable of **inference**: there is *no* learning in the spiking domain. This spiking simulator was developed based on several assumptions and structured to integrate the already existing MORPHIC Python simulator, as explained in the next sections.

4.4.1 Assumptions

- Processing units are (A)IF neurons with reset by subtraction (not reset to zero) in the Conv layers and (A)IF neurons with reset to zero in the FC layers (MORPHIC, [20])
- There is no delay between neurons, i.e. *pseudosimultaneity* or *coincidence* property [37, 51]
- All the implementation is made with digital SNN implementation (i.e. mapping 1-to-1 with the spiking simulator, no unhandled noise can corrupt computations)
- Resolution of the synaptic weights for FC layers is 1 bit (MORPHIC, [20])
- Resolution for the membrane potential/threshold for FC layers is 11 bits (MORPHIC, [20])
- Resolution of the synaptic weights for Conv layers is *limited to*: floating-point ($R_{synapses} = 32$ bits) and fixed-point ($R_{synapses} = 1 - 2 - 4 - 8 - 16 - 32$ bits)
- Resolution for the membrane potential for Conv layers is $R_{neurons} = 32$ bits
- Interfaces between the different parts of the simulator are considered perfect, i.e. no delay nor data loss
- It simulates SNNs rate-based computations
- The input is required to be of the AER-form (i.e. $\mathbf{e} = [x, y, t, p]$)

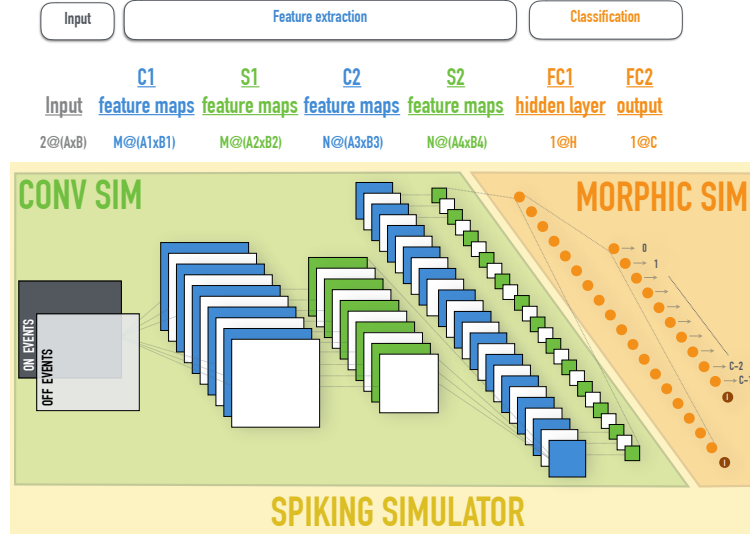


FIGURE 4.6 – *Structure of the spiking simulator. All connections are not shown.*

4.4.2 Structure

The spiking simulator is made up of **two main parts**, as sketched in Figure 4.6. The **first part** is the convolution simulator (CONV SIM) which is responsible for propagating spikes from the input (spiking dataset) to the output of the Conv layers while using (A)IF neurons and weights directly imported from KERAS. All neurons in the CONV SIM part have the same model and all neurons belonging to the layer l have the same threshold $\theta_m^{layer_l}$. The CONV SIM was designed to take advantage of the sparse and repetitive connections in convolution architectures. Indeed, when a spike is detected at the input of the network, only the neurons connected to this firing input neurons are updated in the following layer. This process is repeated until spikes reach last average pooling layer, i.e. the last layer of the Conv part. There, spikes are queued in a vector and the next input spike is processed. This produces the output of the CONV SIM part, i.e. a sequence of neuron indexes indicating which neuron has spiked.

The **second part** is the MORPHIC simulator (MORPHIC SIM) and has been originally developed by Charlotte Frenkel with a 1-to-1 mapping with the real hardware. Even if it was softly adapted for this work, great care was taken to conserve the 1-to-1 mapping property. The input is a sequence of spiking neuron (actually the output of the CONV SIM part) and the output is another sequence of spiking neurons. The *classification* is made according to the output neuron that spikes the most over a pre-defined period of time (highest firing rate). Even if initially, the MORPHIC simulator did not use explicitly timestamps, we have modified it such that it is possible to monitor the output of MORPHIC through time, which we called *inference curves*. They are used in Chapter 5.

The simulator was designed to offer an **attractive degree of adaptability**. In both part of the simulator (CONV SIM and MORPHIC SIM), it is possible to choose which neuron model is used for inference, i.e. either the basic IF neuron model or the adapted one. In addition, it is also possible to choose if both polarities have to be considered or if only the ON polarity has to be taken into account. Winner-Take-All (WTA) algorithm [146, 147] was implemented in the FC layers and can be used if needed, with a configurable intensity. Regarding the structure of the simulator, it is limited to two successive conv-pooling layers followed by two FC layers. However, the number of channels M and N as well as the kernel sizes $k_{1 \rightarrow 4}$ can be modified, depending on the architecture implemented on KERAS. Obviously, the architecture of the tailored CNN has to be the same as the SNN architecture. The weights obtained after training on KERAS are stored in a **pickle** file (**.pkl**) and directly imported in the spiking simulator. All these parameters and even more options can be easily tuned from the main Python interface of the simulator, as shown in the **COMMANDS** part of the Figure given in Appendix A.1 which gives a comprehensive overview of the spiking simulator architecture.

For simulations on conventional computers, each spike has to be sequentially propagated through the network whereas on-chip, a massive-parallel data handling occurs [68]. Thereby, depending on both the number of spikes contained in the input dataset and parameters of the SNN, *simulations* can take a huge amount of time (tens of hours...). For that reason, a **multi-cores version** of the spiking simulator was developed to speed up simulation time by a factor (roughly) equal to the number of cores N_{cores} : the dataset is split and spread among the N_{cores} cores. Results from the different cores are finally agglomerated to get the overall results (i.e. mean classification accuracy over the N_{cores} cores, confusion matrix, vector thresholds used for the simulation, average number of spikes in the output layer, ...). Moreover, after inference of each sample, all neurons are reset to their resting state in order to avoid biasing the next sample inference.

In order to be able to use the results obtained with the spiking simulator, **log files (logs)** are generated for each run in the form of **text** file (**.txt**) and **pickle** file (**.pkl**). They include the detailed results of each core and the overall results which are the one of interest.

Beside the spiking simulator, other Python scripts were developed in order to generate, convert and display neuromorphic data. All figures in this work are generated thanks to these scripts, see Appendix A.1. All the scripts and valuable files will be made available through the GITLAB ECS (more than 30GB of files).

Simulation results

This chapter gathers all the simulation results and discussions for the three different datasets (i.e. SSP-MNIST, N-MNIST and N-CARS) using a similar structure for each one (i.e. network structure, frame-based reconstructed dataset, training on KERAS, thresholds balancing, results and discussion). Considerations for hardware implementation and a comprehensive discussion are also given at the end of this chapter.

5.1 SSP-MNIST

To understand design choices made in this section, it is fundamental to remember that the major objective of this first step on SSP-MNIST was *not* to outperform state-of-the-art classification accuracy but rather to make sure that the spiking simulator was working as expected.

5.1.1 Network structure

As the rough structure of the network was sketched in Section 4.1, remaining parameters M , N , $k_{1 \rightarrow 4}$ and H had to be chosen in order to fulfill both Equation (4.1) and memory requirements. Meanwhile, obtaining sufficient performances after the training on KERAS was also an objective. Contrarily to usual machine learning problems, dealing with hardware constraints sets up limitations on the network. As long as the classification accuracy was sufficient (set arbitrarily above 98%), the structure was considered.

As SSP-MNIST is only made of ON-spikes, a single channel input was sufficient. The chosen network structure is shown in Figure 5.1. Equation (4.1) is satisfied as $25 \times 4 \times 4 = 400 < MorphIC_{fan-in}$ and $500 + 10 + 2 = 512 \leq MorphIC_{core-capacity}$. As explained in Section 3.4, constraints on activation function and bias were applied for each layer. In addition, FC layers are binary (MORPHIC).

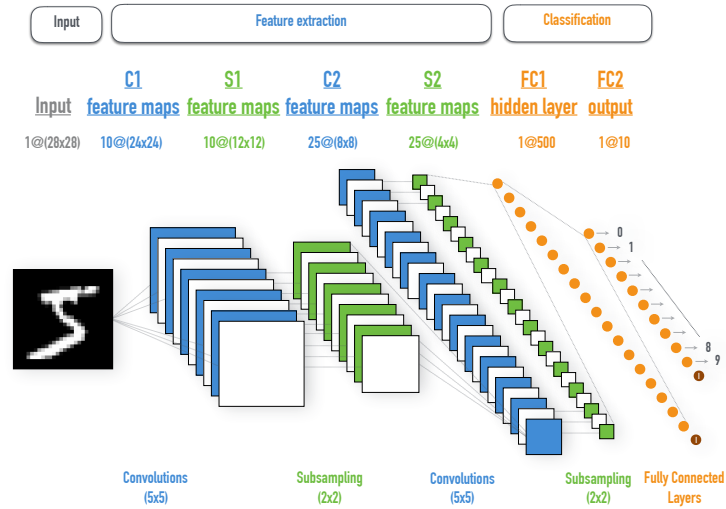


FIGURE 5.1 – *Network structure* used for SSP-MNIST. All connections are not shown.

5.1.2 Frame-based reconstructed dataset

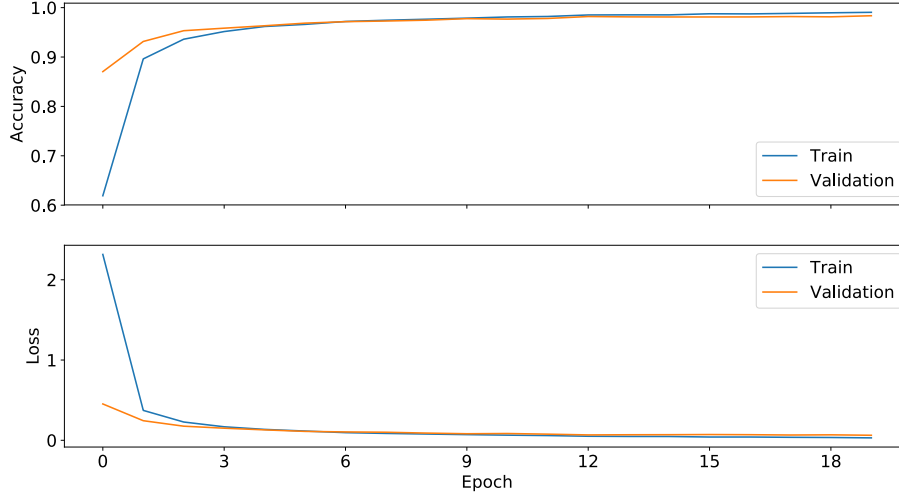
There was no need to reconstruct a dataset for learning on KERAS as SSP-MNIST was created noiselessly from the original MNIST dataset. Consequently, the frame-based reconstructed dataset corresponding to SSP-MNIST is precisely MNIST.

5.1.3 Training on Keras

First of all, no preprocessing was applied on MNIST. The only modification was to normalize pixel intensities from a range of $[0, 255]$ to a range of $[0, 1]$. Then, a validation set was created by taking apart 10 thousand samples ($\sim 16.7\%$) from the original training set. It is a rigorous approach to split clearly training-validation and test set. However, it comes at the cost of a smaller training set and thus possibly slightly poorer results. Backpropagation algorithm was applied on the remaining part of the training set whereas forward pass was carried out on the validation set (*no learning* on the validation set [148]).

By monitoring the evolution of both training accuracy and validation accuracy as shown in Figure 5.2, any overfitting can be observed. This was not the case here. Best weights were selected as weights leading to the best validation accuracy over all epochs. Eventually, the test accuracy was assessed by using these best weights for inference on the test set, leading to **98.31%**. This accuracy matters as it is the metric usually used to compare different methods and networks. Parameters used for learning are given in Table 5.1. ADAM (Adaptive Moment Estimation) [149] optimizer was used to improve accuracy and accelerate learning by computing adaptive learning rates for each parameters [92]. It is based on adaptive estimates of lower-order moments (β_1, β_2) [149].

Epochs	Batch Size	Optimizer = Adam (default values)			
		η	β_1	β_2	ϵ
20	640	$1 \cdot 10^{-3}$	0.9	0.999	$1 \cdot 10^{-8}$

TABLE 5.1 – KERAS *parameters for MNIST dataset learning.*FIGURE 5.2 – *SSP-MNIST learning curves obtained on KERAS. Test accuracy resulting of this training was 98.31%.*

5.1.4 Thresholds balancing

In order to have the best mapping possible, thresholds have to be balanced as explained in Section 3.4.3. In order to decrease drastically the degree of indetermination, all neurons belonging to the layer l were given the same threshold $\theta_m^{layer_l}$. Indeed, as a grid search algorithm was used, it was crucial to lower the number of degree of freedom to save computational resources. For this network, the problem was thus to find

$$\Theta_m = \underbrace{[1, \theta_m^{Conv1}, \theta_m^{Pool1}, \theta_m^{Conv2}, \theta_m^{Pool2}]}_{\text{Thresholds sub-vector for CONV layers}} \underbrace{[1, \theta_m^{Hidden}, \theta_m^{Output}]}_{\text{Thresholds sub-vector for FC layers}} \quad (5.1)$$

As subsampling layers implement averaging operation over a square kernel of size $k_i \times k_i$, weights are uniformly distributed with value $\frac{\theta_m^{Pool_i}}{k_i^2}$. Consequently, $\theta_m^{Pool_i}$ can be set to 1. The '1' at the beginning of each sub-vector means that input neurons are considered. The thresholds vector Θ_m in Equation 5.1 was thus reduced to

$$\Theta_m = [1, \theta_m^{Conv1}, 1, \theta_m^{Conv2}, 1][1, \theta_m^{Hidden}, \theta_m^{Output}]. \quad (5.2)$$

In order to reduce simulation time, a bit of intuition and insight was used. For instance, avoiding thresholds leading to highly saturated or dead neurons while ensuring that enough spikes can go through the convolution part of the network is an approach that usually improved performances. It can be done by using the spiking simulator in order to obtain $\theta_m^{Conv1 \rightarrow 2}$ such that error between the activation maps of the convolutions layers in the tailored network and the corresponding activation maps in the spiking network is made sufficiently small. RMSE was used as a metric to compare how close activation maps are (i.e. how great is the mapping) as can be observed on the different sub-figures in Figure 5.3. This approach was chosen to tune $\theta_m^{Conv1 \rightarrow 2}$.

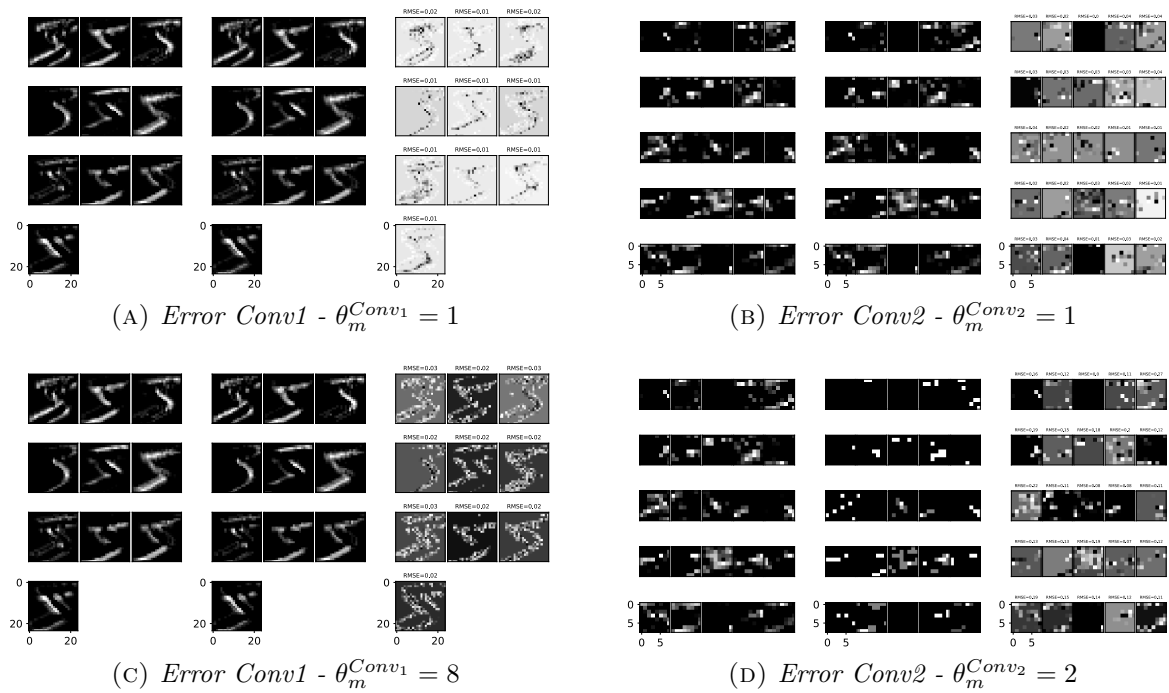


FIGURE 5.3 – **Error in Conv layers.** In each sub-figure, the first set of activation maps is the output of the Conv layer on KERAS. The second set is the activation maps of the corresponding Conv layer in the spiking simulator and the last set is the difference between the two first sets, activation map by activation map. RMSE is used as the error metric. This was done here for the first sample of the MNIST training set (label = 5).

Only θ_m^{Hidden} and θ_m^{Output} remained to be balanced. Figure 5.4 shows the behavior of the FC layers in the spiking network for two different thresholds vectors. Following the same reasoning as for Conv layers, activation maps of the spiking network were wanted to be as close as possible to the ones obtained on KERAS. A grid search algorithm was used to find the best $[1, \theta_m^{Hidden}, \theta_m^{Output}]$ sub-vector. It has been observed that the **threshold of neurons in the hidden layer** has to be sufficiently large so that neurons can exhibit their non-linear behavior. This was often also true for the threshold of the output layer, but not always. However, enough spikes have to get out of the Conv layers.

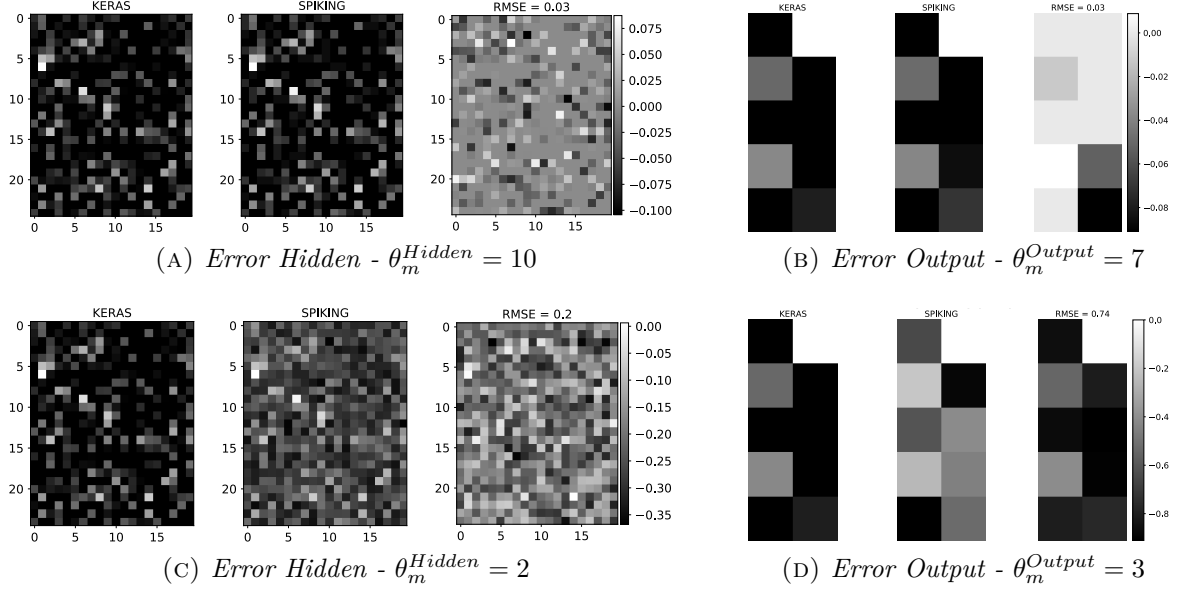


FIGURE 5.4 – **Error in hidden and output layers.** Comparison was made between activation maps obtained on KERAS and activation maps in the spiking simulator. Activation maps were normalized such that they can be compared and RMSE was used as the error metric. Sub-figures (A) and (B) illustrate a good mapping $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$ as the error is low (RMSE = 0.03 for both layers) whereas (C) and (D) give an example of a bad mapping $\Theta_m = [1, 1, 1, 1, 1][1, 2, 3]$ as activation maps are very different and error is high (RMSE = 0.2 and 0.74 respectively). This was done here for the first sample of the MNIST training set (label = 5).

5.1.5 Results

In previous sections, the focus was made on *choosing an architecture*, *training properly* the tailored network and eventually *balancing thresholds* such that the mapping is accurate. It is now time to observe how the spiking network behave from the inside and assess its performance.

An interesting result is obviously the **accuracy** obtained after the inference on the spiking test set. It was done through a majority vote using the cumulative spike count of neurons at the output layer (highest firing rate) over a pre-defined time interval [51], usually the whole sample duration. Consequently, the best scenario at the output would be to have the right neuron spiking a lot more than others neurons. In other words, the bigger the gap between the firing rate of the right neuron and the others, the better, as it means that the network is more confident about its inference.

However, the accuracy is not the only interesting result. It was observed that a variant of the basic IF neuron model could have a (strong) influence on the final performance. The

methodology was not modified, only the neuron model was adapted in the SNN. Three combinations (i.e. IF, AIF and HYBRID implementation as shown in Figure 5.5) were investigated in the following sections.

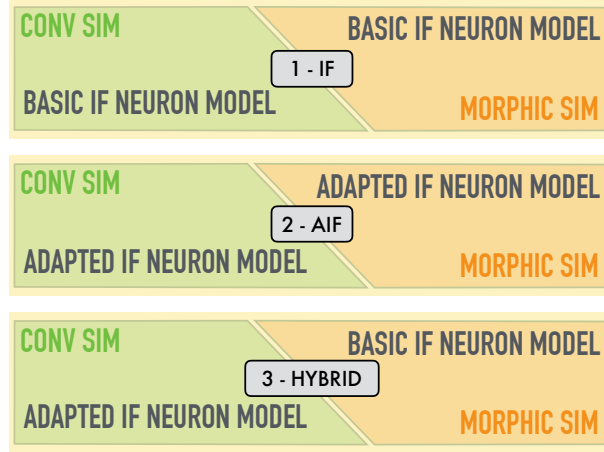


FIGURE 5.5 – *Three combinations of neuron models investigated with the spiking simulator.*

5.1.5.1 IF implementation (basic IF neuron model)

For this part, the basic IF model was considered, i.e. $\theta_m^{down} = 0$ and $V_{resting} = 0$. An accuracy surface was computed as presented in the first subplot of Figure 5.7 leading to a local optimum of 96.82% for $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$. Using this Θ_m on the full test set gave an accuracy of **96.03%**. However, even if $\theta_m^{Output} = 1$ gave the best classification accuracy, it did not allow neurons to integrate spikes properly and prevented it to express their IF behavior. In fact, all neurons have a high firing rate as can be observed in Figure

5.8. Moreover, the error between output activation maps increases drastically when $\theta_m^{Output} = 1$, as shown in Figure 5.6. Second and third best thresholds vectors were $\Theta_m = [1, 2, 1, 2, 1][1, 8, 1]$ and $\Theta_m = [1, 2, 1, 2, 1][1, 12, 1]$ leading to 96.5% and 96.37% on the accuracy surface and then 95.97% and 95.5% on the test set, respectively. Nevertheless, θ_m^{Output} was still equal to one. Using $\Theta_m = [1, 2, 1, 2, 1][1, 10, 7]$ gave 92.22% on the accuracy surface and 91.66% on the test set, which is far beyond previous results.

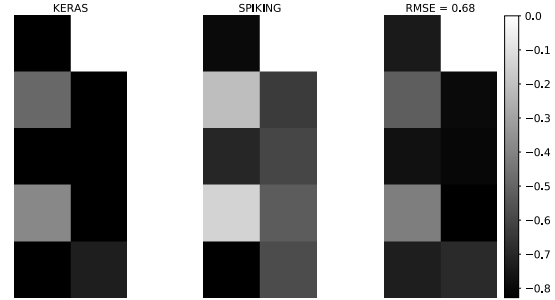


FIGURE 5.6 – *Error in output layer. Comparison between the activation map obtained on KERAS and the one obtained in the spiking simulator for the first sample of the training set (label = 5). RMSE is used as the error metric. $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$.*

Consequently, even if $\theta_m^{Output} = 1$ seems to be a bad choice in theory, it turns out to be the parameter leading to best classification accuracy.

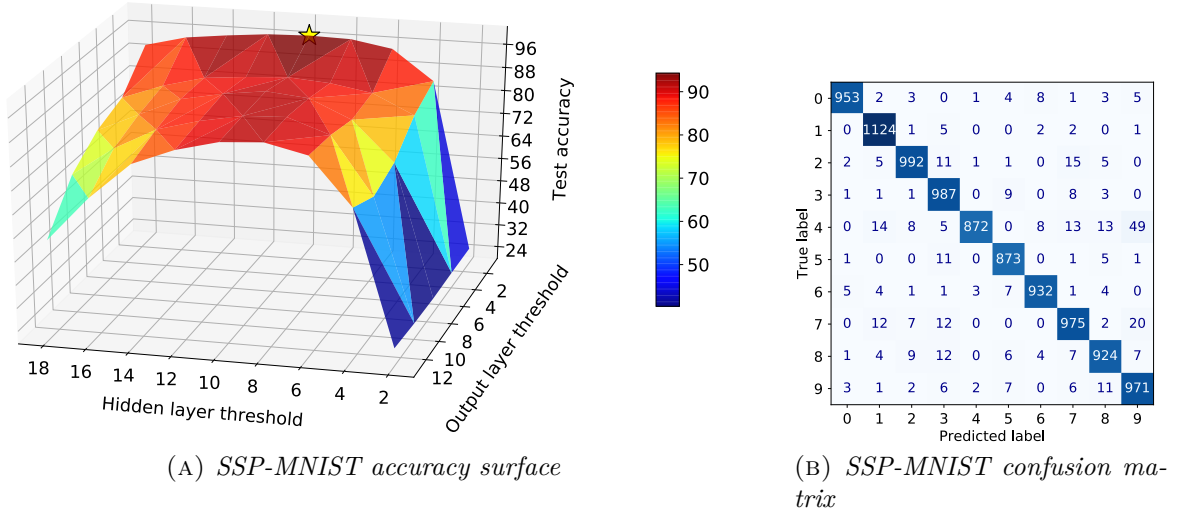


FIGURE 5.7 – *IF* implementation. (A) **SSP-MNIST accuracy surface with the basic IF neuron model**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow star corresponds to the highest accuracy on the surface (96.82%) which was obtained for $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$. (B) **SSP-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$ obtained in (A), inference was carried out on the full test set (10k samples). The resulting accuracy was 96.03%.

Inference curves given in the second subplot of Figure 5.8 are interesting to comment the performance. Despite the fact that best classification accuracy was obtained using a $\theta_m^{Output} = 1$, inference curves are really close to each other during all the inference and at the end of the inference as well (less than 1% gap regarding the total number of spikes between the first and the second highest firing rate output neuron). A small perturbation in the input or in the network could change the result of the classification, which is what we want to avoid. In addition, the firing rates of output neurons are really high (e.g. $\simeq 1500Hz$).

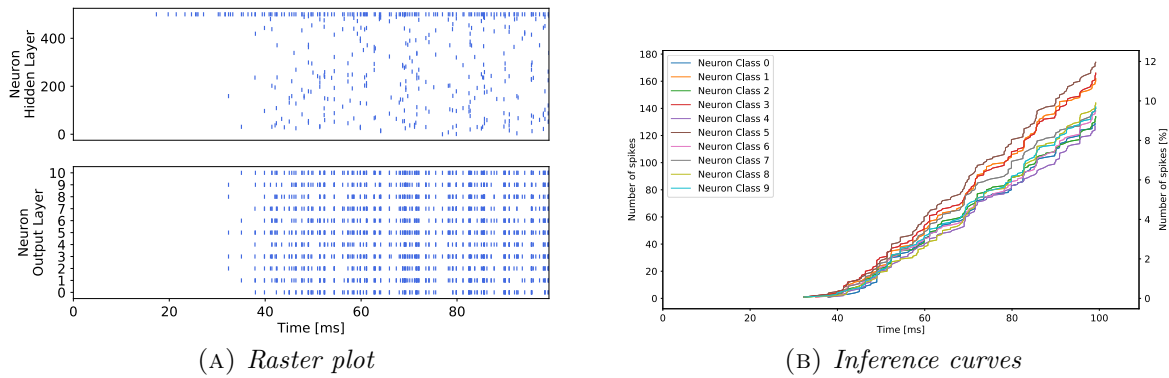


FIGURE 5.8 – IF implementation. (A) **Raster plot** for the two FC layers of the network, considering the first sample of the training set (label = 5). The sub-plot on the top is for the hidden layer while the one on the bottom is for the output layer. $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$. (B) **Inference curves**, for output neurons of the first sample of the training set (label = 5). $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$.

5.1.5.2 AIF implementation (adapted IF neuron model)

For this part, the adapted IF model was considered. A similar variant was investigated in Stomatias et al. [3]. Figure 5.9 gives the result of the grid search algorithm used to find $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$ on the training set. The resulting accuracy on the test set was **98.17%** as can be seen with the confusion matrix given in Figure 5.9. It is only 0.14% away from the accuracy obtained with the continuous network using KERAS.

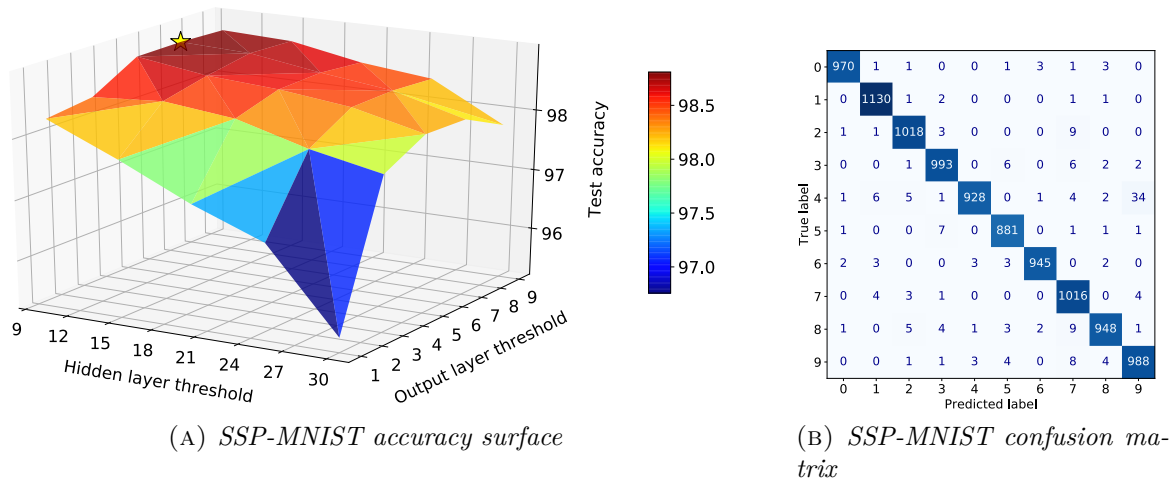


FIGURE 5.9 – AIF implementation. (A) **SSP-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow star corresponds to the highest accuracy on the surface (98.83%) which was obtained for $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$. (B) **SSP-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$ obtained in (A), inference was carried out on the full test set (10k samples). The resulting accuracy was 98.17%.

The first subplot of Figure 5.10 displays the behavior of the fifth output neuron for the two first sample of the training set. The sub-threshold membrane behavior can be observed and indeed the membrane potential goes below zero (as the AIF neuron model was considered). The second subplot of Figure 5.10 gives the inference curves for all output neurons. The (approximated) slope of each inference curve is the output mean rate of firing and in this case, spikes coming out of the fifth neuron represent around 45% of all output spikes, whereas spikes coming out of the third neuron (second highest rate) represent around 25% of all spikes. This is more distinct than in Figure 5.8. This could be explained by the fact that the output threshold is higher and thanks to $\theta_{thresh}^{down} < 0$ in the whole network which allows for a more accurate conversion of the continuous-valued neurons activations in firing rates.

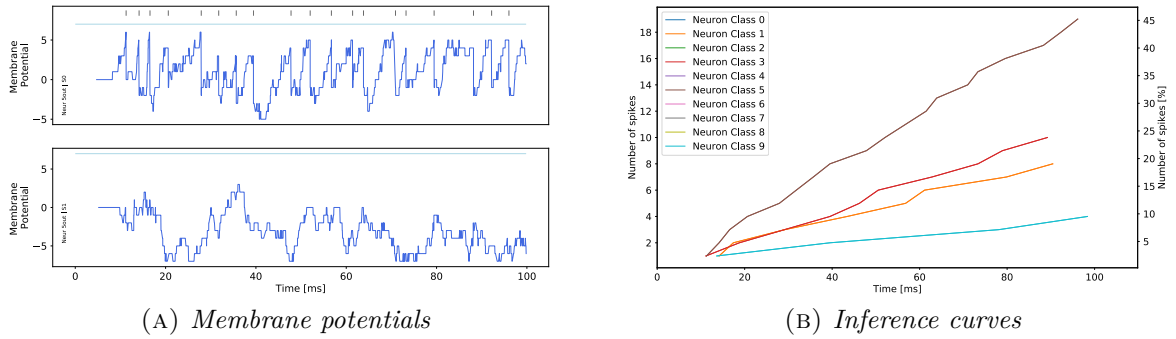


FIGURE 5.10 – AIF implementation. (A) **Evolution of the membrane potential $V(t)$** , for the fifth neuron of the output layer. The sub-plot on the top is for the first sample of the training set (label = 5) while the one on the bottom is for the second sample (label = 2). $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$. (B) **Inference curves**, for output neurons of the first sample of the training set (label = 5). $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$.

5.1.5.3 Hybrid implementation (trade-off)

The basic IF neuron model yields poorer results than the variant proposed here above (96.03% against 98.17%). However, this increase in performances when using the hybrid implementation has a major *drawback*: it cannot be mapped on MORPHIC as the membrane potential cannot go below 0 (the basic IF neuron model is implemented in MORPHIC, practical hardware constraints). To mitigate this limitation, a trade-off is possible: using the AIF neuron model in the Conv layers and the basic IF model in the FC layers, such that it could fit on MORPHIC.

The same methodology was applied, only models in the spiking network were modified. A local optimum of 98.87% was found for $\Theta_m = [1, 2, 1, 2, 1][1, 16, 1]$. Using this Θ_m on the full test set gave **98.19%** of accuracy as shown in Figure 5.11, which is surprisingly

slightly better than previous results. Indeed, poorer results were expected as the conversion in firing rates is of lower quality. However, θ_m^{Output} was still equal to one. It happened once again when the neuron model in MORPHIC was the basic IF model with $\theta_m^{down} = 0$ and $V_{resting} = 0$. Other Θ_m were also of interest as $\Theta_m = [1, 1, 1, 1, 1][1, 10, 1]$, $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$ and $\Theta_m = [1, 1, 1, 1, 1][1, 14, 13]$ leading to 98.7%, 98% and 98.72% on the accuracy surface and then 97.99%, 97.44% and 97.79% on the test set, respectively. Consequently, using the same Θ_m as in Section 5.1.5.2 led to a drop of $0.73 + 0.14\%$ in the classification accuracy whereas using the accuracy surface with the constraint $\theta_m^{Output} \neq 1$ led to a drop of $0.38 + 0.14\%$. In both cases, hardware constraints are fulfilled.

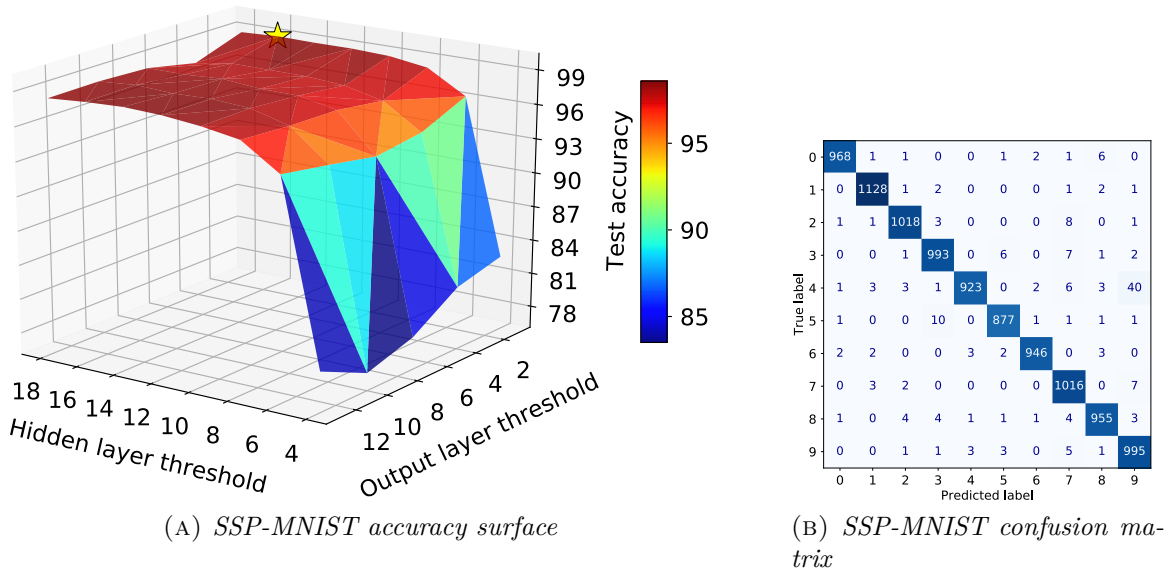


FIGURE 5.11 – Hybrid implementation. (A) **SSP-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow star corresponds to the highest accuracy on the surface (98.87%) which was obtained for $\Theta_m = [1, 1, 1, 1, 1][1, 16, 1]$. (B) **SSP-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 1, 1, 1, 1][1, 16, 1]$ obtained in (A), inference was carried out on the full test set (10k samples). The resulting accuracy was 98.19%.

5.1.5.4 Winner-Take-All (WTA) implementation

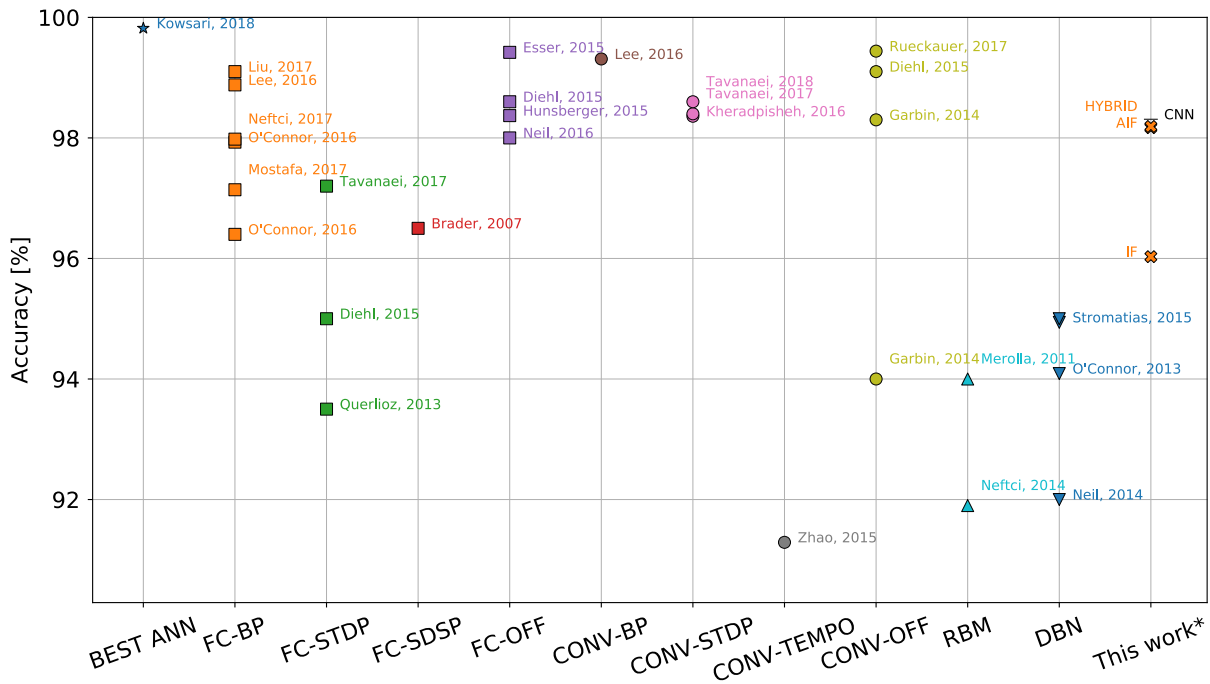
To enhance the separation between the inference curves at the output layer, we implemented a Winner-Take-All (WTA) structure on top of the hybrid implementation. WTA was only implemented within the output layer. Each time a neuron spiked, the membrane potential of all other output neurons (except the inhibitory neuron) was lowered by a factor $\phi_{WTA} \times \theta_m^{Output}$ where ϕ_{WTA} was usually between 0 and 1. Further WTA simulations were not carried out because it did not lead to valuable improvements in classification accuracy.

5.1.6 Discussion

The **main objective** of this section was to confirm that the spiking simulator is working well, which is the case at the light of the given results. In fact, the difference between the inference error before and after mapping can be made as small as 0.14%. Moreover, the behavior of each spiking layer was compared with the continuous ones to ensure that the rate-based spiking network mimics well the tailored network. Consequently, this spiking simulator can be used with confidence.

An **important result** is that the ratio thresholds/weights has a crucial role on the general behavior of the spiking network and on the final performances, as suggested in Section 3.4.3. Performances are stable for similar Θ_m , but they can drop drastically for other Θ_m as shown in Figure 5.7. It is thereby capital to find the best Θ_m , or at least a relatively good one. In addition, the **influence of the neuron model** was investigated and performances improvements were shown on this dataset, i.e. +2.14%. A trade-off (i.e. hybrid implementation) was proposed such that the network could be implemented in practice while respecting hardware constraints. This has to be carried out on other datasets in order to confirm that it is actually an improvement.

Finally, even if the objective was not to surpass state-of-the-art accuracy, this approach led to **decent performances** when compared to reported best accuracies of different approaches, as shown in Figure 5.12. However, SSP-MNIST is not exactly the same spiking dataset as the one used in those approaches. In fact, they usually used a Poisson spike generator. Nevertheless, they usually did not have any constraints on the resolution of thresholds and synapses or on the number of neurons, leading to difficulties for fair comparison. Moreover, the performance obtained for this dataset might not be the best performances achievable with that network. In fact, there is no guarantee that the operating point is a global optimum. The accuracy surfaces in Figures 5.7 and 5.9 were computed for a pre-defined convolution thresholds sub-vector so it might lead to a local optimum. Although the error can be made really small both in the Conv and FC parts, an important question arises: is it possible to reach perfect mapping, i.e. no error at all between activation maps? In other words, does it exist a thresholds vector $\Theta_{m-optimum}$ leading to global optimum such that the spiking network behaves exactly as the continuous one ? Using a proper optimization formalism (threshold balancing algorithm) with a cost function should solve that issue or at least, help sketching an answer.



*: Spiking dataset has not been generated with Poisson spike train generator. Network is subject to hardware constraints.

FIGURE 5.12 – **MNIST reported accuracies** for spiking networks. *BEST ANN* = best accuracy reported for ANN network, *FC* = Fully Connected, *BP* = Backpropagation, *OFF* = Offline training (conversion or hardware), *CONV* = Convolution, *TEMPO* = Tempotron, *RBM* = Restricted Boltzmann Machines, *DBN* = Deep Belief Network. Data compiled from [71] and [4, 47, 92, 123, 150]. The classification accuracy for the three combinations (IF, AIF and HYBRID implementation) proposed in this section are given. The continuous-valued CNN classification accuracy is also shown in order to compare the classification accuracy before and after mapping.

5.2 N-MNIST

5.2.1 Network structure

As N-MNIST is *not* a framed-based dataset and because a mapping approach was chosen, it was needed to get a relevant dataset (as close as possible to N-MNIST) in order to train the tailored network. Thereby, there were mainly **two ways to proceed**: either *re-using weights obtained from the MNIST training* (as N-MNIST was produced directly from this dataset) or *training a new network* with a reconstructed frame-based dataset. Both ideas are investigated and discussed in this chapter.

Regarding the first simulation (i.e. re-using weights obtained from the MNIST training), there was no need to reconstruct a dataset to train a tailored CNN as weights were used without any modification from Section 5.1. Nevertheless, it has a significant *drawback*: it requires to have exactly the same architecture as the one used for SSP-MNIST. **That is the reason why N-MNIST was cropped to a 28x28 shape instead of 34x34.** Consequently, the architecture was the same as the one used for SSP-MNIST, as shown in Figure 5.13. The only difference was the number of input channels: only the ON-channel was used for this first simulation (as there is only weights available for the ON channel because SSP-MNIST has a single channel input).

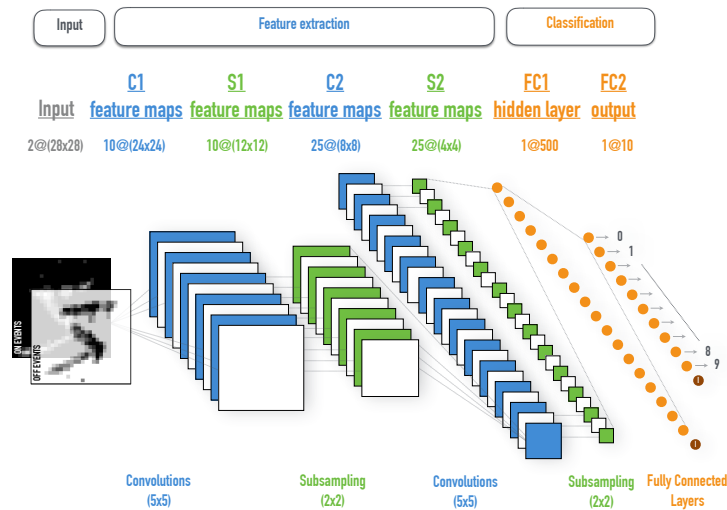


FIGURE 5.13 – *Network structure* used for N-MNIST. All connections are not shown.

In order to be able to compare performances improvements due to training with a reconstructed dataset, the same network architecture was used for the second simulation, except for the input that have to handle 2 channels, as shown in Figure 5.13. Once again, it appears clearly why the original N-MNIST dataset was cropped.

5.2.2 Frame-based reconstructed dataset

The **reconstruction method** followed the approach described in Section 3.4.2 and was applied to the cropped version of N-MNIST. Figure 5.14 exhibits how the five first samples of the N-MNIST test set (spiking) were converted to several corresponding frame-based images (non-spiking) by **applying the reconstruction approach over multiple accumulation windows**. The purpose of this *temporal multi-scale reconstruction* was to **capture more dynamics** of the spiking dataset. How to choose the duration dt and the position of the different accumulation windows within the whole sample duration was an important question that we attempted to address in this section.

Despite it was not the result of any optimization, **six accumulation windows were chosen** (e.g. 20, 50, 80, 120, 175 and 250 ms) to capture several dynamics behaviors of this dataset. For each sample, the **positions of accumulation windows were randomly chosen** within the available time for that sample. It was motivated by the willingness to develop a **method that generalizes well**. This particular process was repeated for all the training and test samples to finally get the frame-based reconstructed dataset. Consequently, the training set of this reconstructed N-MNIST dataset includes now $6 \times 60k = 360k$ samples and the test set includes $6 \times 10k = 60k$ samples. It could be seen as data augmentation but the objective was more to capture a part of the dynamics inherent to this dataset rather than increasing the number of samples in the dataset.

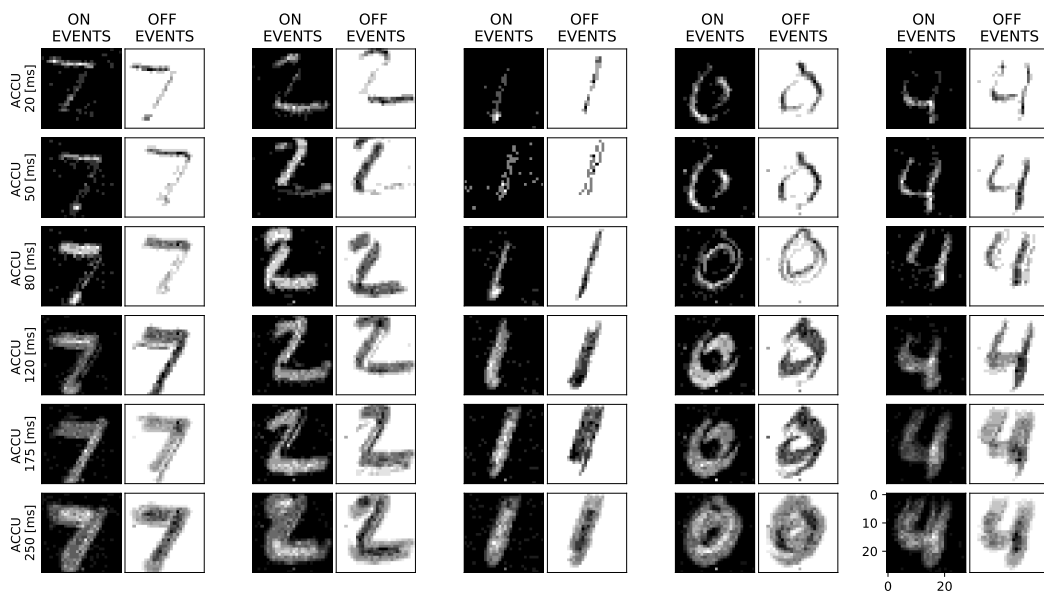


FIGURE 5.14 – *N-MNIST frame-based reconstructed dataset*. The samples shown here are the five first samples of the N-MNIST test set (one per column). Each row corresponds to a different accumulation time (ACCU) as indicated on the left axis. Each image was normalized as explained in Section 3.4.2.

5.2.3 Training on Keras

Unlike MNIST, some preprocessing was applied on N-MNIST as explained in Section 2.5.3. The only modification was to crop the original N-MNIST dataset from a 34x34 input shape to a 28x28. Then, a frame-based version of the dataset was created in order to train the network on KERAS. As for MNIST, a validation set was created by taking apart 60 thousand samples ($\sim 16.7\%$) from the reconstructed training set. The reconstructed training set was shuffled before the validation split in order to prevent the validation set to be composed only with samples reconstructed with large accumulation windows, which would not be representative. In fact, the validation set on KERAS is created with the last samples of the dataset [148]. Backpropagation algorithm was applied on the remaining part of the training set whereas forward pass was carried out on the validation set (*no learning* on the validation set [148]). As the input is split in two different channels (ON/OFF polarities), two independent banks of filters were used for kernels in the first Conv layer.

By monitoring the evolution of both training accuracy and validation accuracy as shown in Figure 5.15, any overfitting can be observed. This was not the case here. Best weights were selected as weights leading to the best validation accuracy over all epochs. Eventually, the test accuracy was assessed by using these best weights for inference on the test set. The resulting classification accuracy was **96.92%**. Parameters used for learning were the same as the ones used for MNIST, Section 5.1.3. The main difference with this learning method is that weights depends on the input dataset which is *defined by the chosen reconstruction method*, Sections 3.4.2 and 5.2.2.

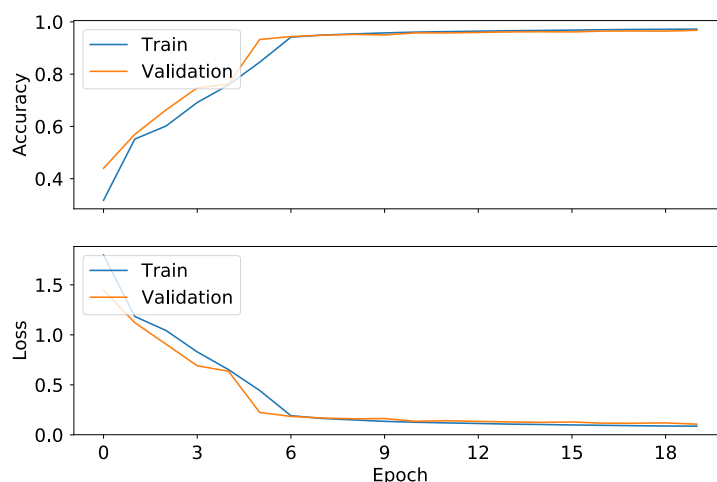


FIGURE 5.15 – *N-MNIST learning curves* obtained on KERAS. Test accuracy resulting of this training was 96.92%.

5.2.4 Thresholds balancing

Similarly to the threshold balancing approach for SSP-MNIST, the objective is to find Θ_m leading to the best classification accuracy. Applying the same simplifications as before to Equation (5.1), Θ_m is reduced to the form given in Equation (5.2).

However, the **approach to find $\theta_m^{Conv1 \rightarrow 2}$** was **different** as there is spatial dynamics in the input dataset. In fact, handwritten digits are moving around which was not the case with the static SSP-MNIST dataset. When comparing activation maps in the spiking simulator and continuous activation maps as shown in Figure 5.16 with an experienced eye, *some resemblances can still be observed*: spiking activation maps look often like **blurry** version of the continuous ones but it cannot really be used to find accurate $\theta_m^{Conv1 \rightarrow 2}$. One solution to get net activation maps could be to accumulate spikes on shorter period of time within spiking layers but still, as convolution kernels apply on input with very different spatial dynamics (static versus mobile), the output behavior will be different. $\theta_m^{Conv1 \rightarrow 2}$ were set such that a reasonable number of spikes was propagated through Conv layers. It was carried out by inspection using the spiking simulator. Once again, a proper *threshold balancing algorithm would be much appreciated* and more formal.

In deeper layers however, **invariant spatial features** were progressively extracted and it was easier to compare activation maps for the FC layers, as shown in subplots (E) and (F) in Figure 5.16. The **effect of threshold balancing** can be observed by comparing these two activation maps with the ones resulting of a bad threshold balancing as shown in Figure 5.17. The RMSE increased from 0.08[/] to 0.26[/] for the hidden layer and from 0.13[/] to 0.77[/] for the output layer. All Figures generated in this section rely on simulations based on the hybrid implementation but the behavior was similar for the others models. **Another way to observe the influence of the threshold balancing** is using the *inference curves* at the output of the last FC layer of MORPHIC, as shown in Figures 5.18 and 5.19. It can be observed that for a bad threshold balancing, the right output neuron had (never) the highest number of spikes or at least, not at the end of the inference. However, that is what matters as the firing rate of the neuron is approximated by the number of spikes over the simulation time. In the case of N-MNIST, an interesting phenomenon appears: the three saccades can be observed during the inference. In fact, the firing rate of the neuron can be (really) different from one saccade to the other and it is significantly larger during saccades as valuable information is going through the network. The *best scenario* would be that the right neuron maintains its inference curve above all the others, as far as possible from them, ensuring a right classification even before the end of the sample duration. It happened that the right neuron had its inference curve

below the others at the beginning of the inference and then the firing rate of that neuron increased rapidly during a saccade, which boosted its inference curve leading eventually to the right classification. This happen in the second subplot of Figure 5.18 and 5.19.

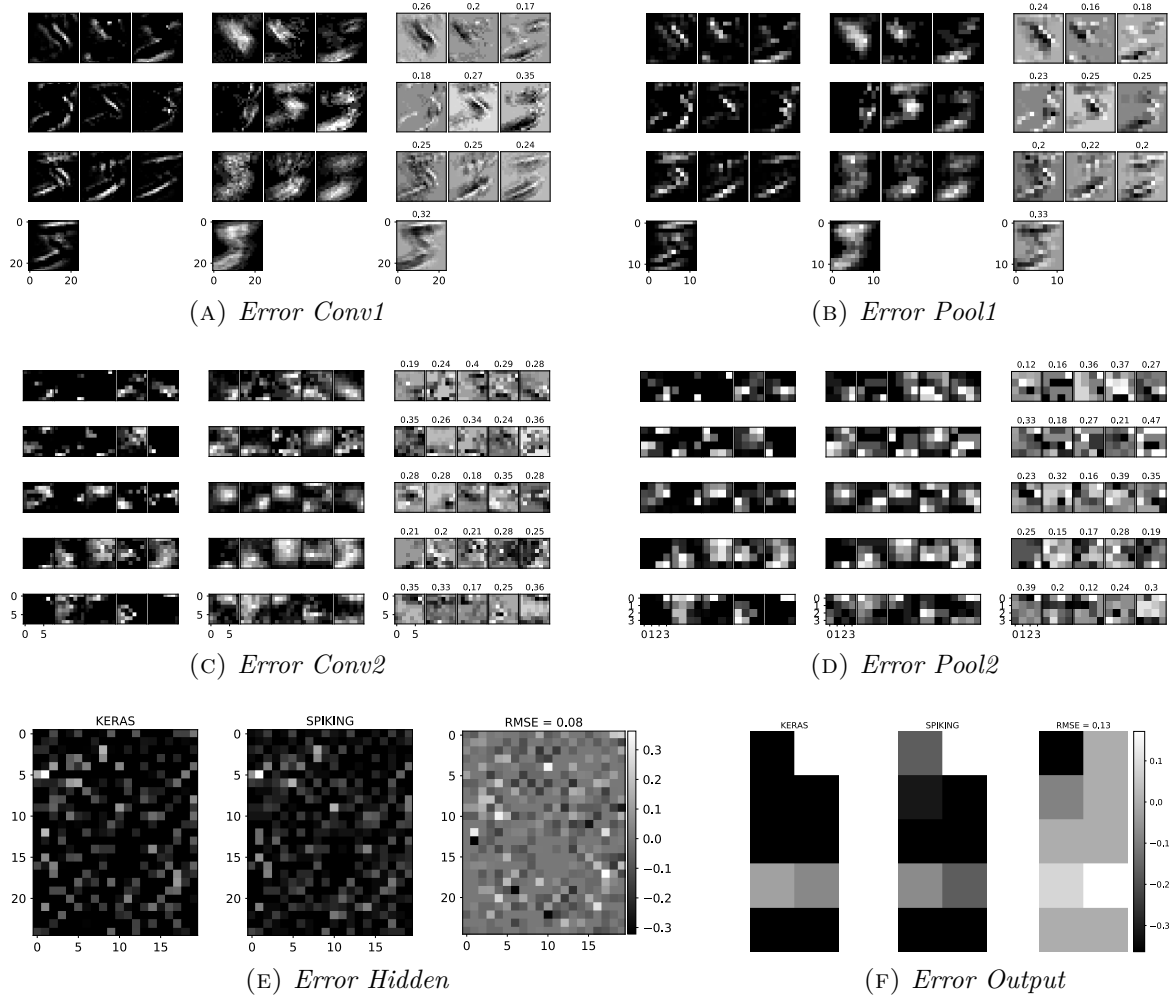


FIGURE 5.16 – **Error in Conv and FC layers.** In each sub-figure, the first set of activation maps is the output of the layer on KERAS. The second set is the activation maps of the corresponding layer in the spiking simulator and the last set is the difference between the two first sets, activation map by activation map. RMSE is used as the error metric. This was done here for the first sample of the MNIST training set (label = 5). $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 10, 10]$.

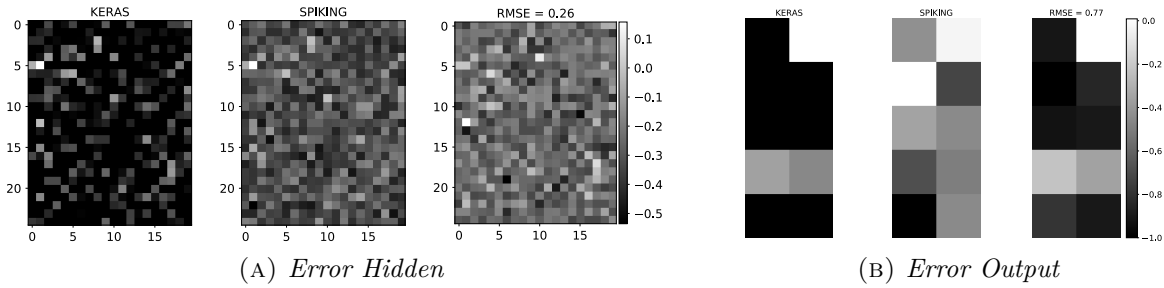


FIGURE 5.17 – **Error in FC layers due to bad threshold balancing.** RMSE is used as the error metric. This was done here for the first sample of the MNIST training set (label = 5). $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 2, 4]$.

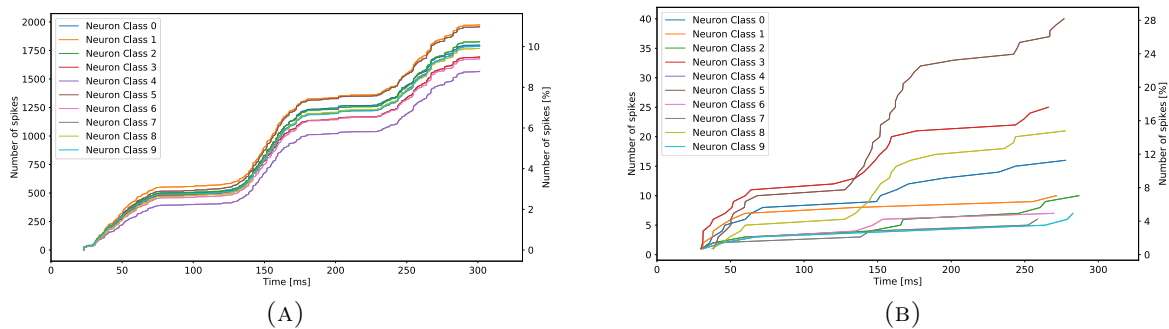


FIGURE 5.18 – **Comparison between inference curves for different threshold balancing** for the **first** sample of the training set (label = 5). (A) Bad threshold balancing, $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 2, 4]$. (B) Good threshold balancing, $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 4]$.

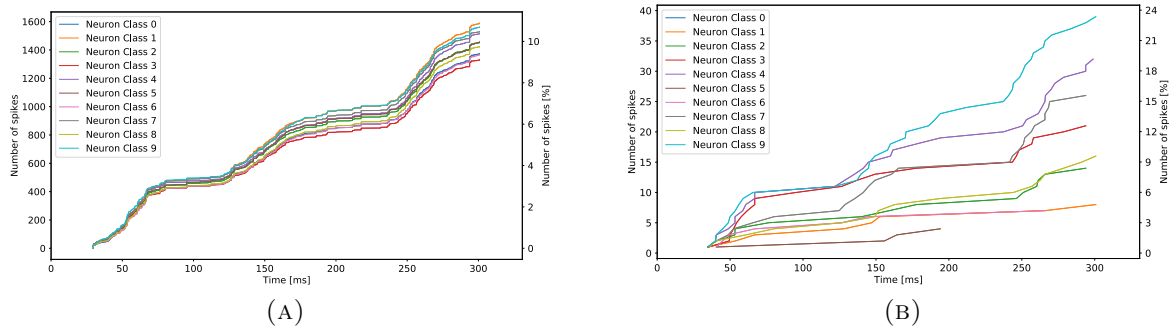


FIGURE 5.19 – **Comparison between inference curves for different threshold balancing** for the **fourth** sample of the training set (label = 9). (A) Bad threshold balancing, $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 2, 4]$. (B) Good threshold balancing, $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 4]$.

5.2.5 Results

This section presents results obtained on N-MNIST. It is organized similarly as the SSP-MNIST results, Section 5.1.5 except for the next section that focuses on re-using SSP-MNIST weights.

5.2.5.1 Inference using the MNIST weights and the hybrid implementation (trade-off)

For this first simulation (i.e. re-using weights obtained from the MNIST training), the architecture of the network that performed inference on N-MNIST was exactly the same as the one used for SSP-MNIST. A single input channel (ON channel) was considered and weights were imported without any modification from the MNIST training, Section 5.1.3. Only the hybrid implementation was considered. The **aim of this simulation** was to understand if weights that had not been trained with noise and dynamics could still lead to reasonable performances.

As can be observed in the first subplot of Figure 5.20, the classification accuracy for 600 samples of the training set was about 50% for several $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 16 \rightarrow 19, 2 \rightarrow 13]$. Although 50% is obviously *not* a good classification accuracy for N-MNIST (chance level is at 10% classification accuracy), it proved that the spiking network was able to recognize samples (some better than others e.g. labels = 1, 2, 3, 5, 6 and 7 as given in the second subplot of Figure 5.20) even if the weights used for the inference had not been trained using N-MNIST. Clearly, N-MNIST information is close to MNIST even if the encoding is different [87] which could certainly explain the potential of MNIST weights to classify N-MNIST samples. However, this is far from sufficient: polarity, noise and scene dynamics have to be considered. This is investigated in the next sections.

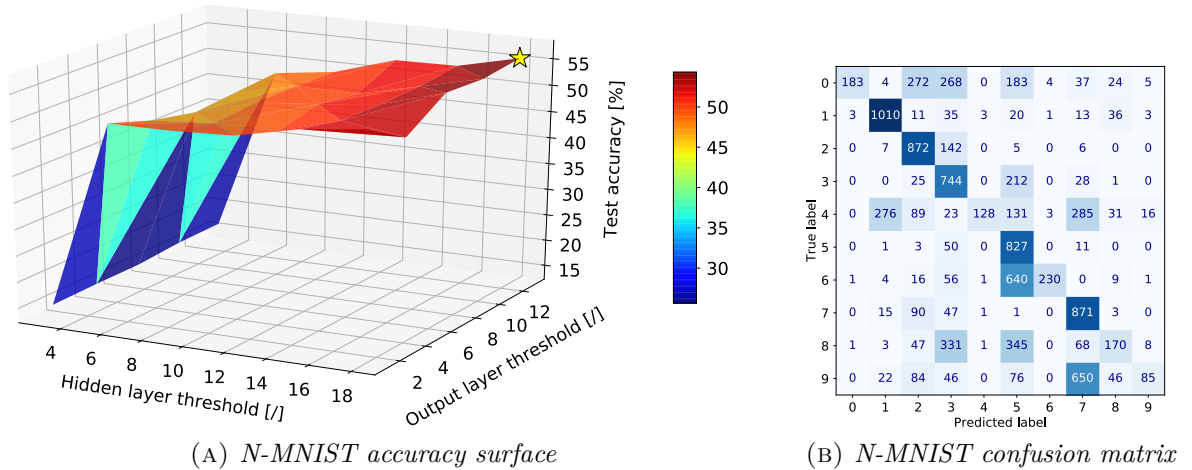


FIGURE 5.20 – Inference using the MNIST weights and the hybrid implementation (trade-off). (A) **N-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 600 samples from the training set for each combination. The yellow star corresponds to the highest accuracy 55.5% on the surface which was obtained for $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 18, 13]$. (B) **N-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 18, 13]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 51.2%.

5.2.5.2 IF implementation (basic IF neuron model)

Figure 5.21 shows how $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 1]$ was chosen and the resulting classification accuracy of **81.25%**. This is clearly better than 50% obtained with SSP-MNIST weights. However, it is about fifteen percent less than the accuracy obtained with the tailored CNN on KERAS, i.e. 96.92%. As low output thresholds gave good results on SSP-MNIST, inference on the full test set was carried out using $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 1]$. It achieved 86.53% classification accuracy which is about five percent more than what was obtained with $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 6]$. Yet, the mapping is not satisfying.

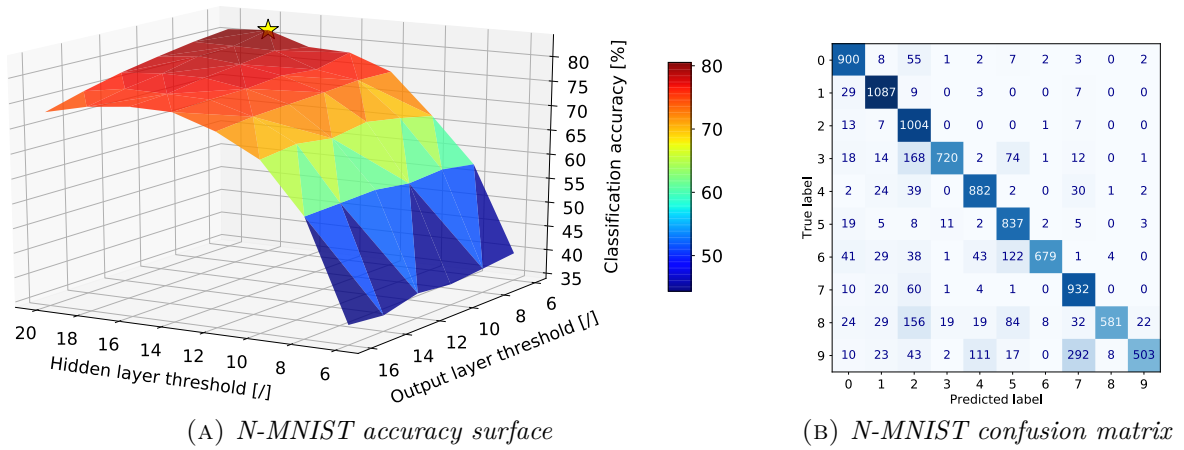


FIGURE 5.21 – *IF implementation. (A) N-MNIST accuracy surface, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow stars correspond to the highest accuracy 81.20% on the surface which was obtained for $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 6]$ and $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 7]$. (B) N-MNIST confusion matrix, for the best thresholds vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 6]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 81.25%.*

5.2.5.3 AIF implementation (adapted IF neuron model)

Using the adapted IF neuron model in all layers boosted the quality of the mapping as presented in Figure 5.22. The classification accuracy obtained after inference in the spiking domain on the full test set was **96.92%**, which is the **same score as its continuous-valued counterpart**. In order to observe if the accuracy was similar when using different Θ_m located in a zone of the grid leading also to good classification accuracy, inference on the full test set was performed with several Θ_m as shown in Figure 5.23. The change in performance is within 1% which suggests stability around the chosen Θ_m , i.e. a certain form of robustness to parameter variation.

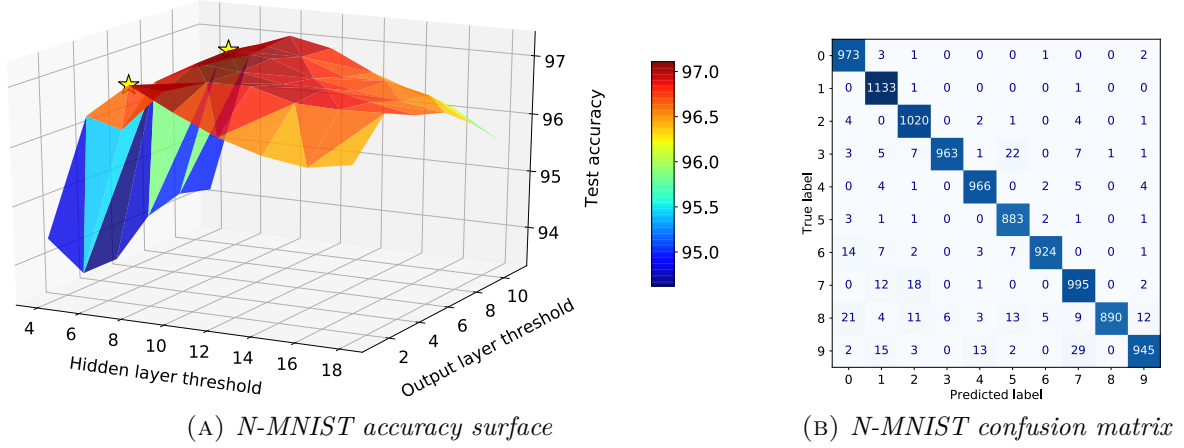


FIGURE 5.22 – AIF implementation. (A) ***N-MNIST accuracy surface***, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow stars correspond to the highest accuracy 97.15% on the surface which was obtained for $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 1]$ and $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 7]$. (B) ***N-MNIST confusion matrix***, for the best thresholds vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 1]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 96.92%.

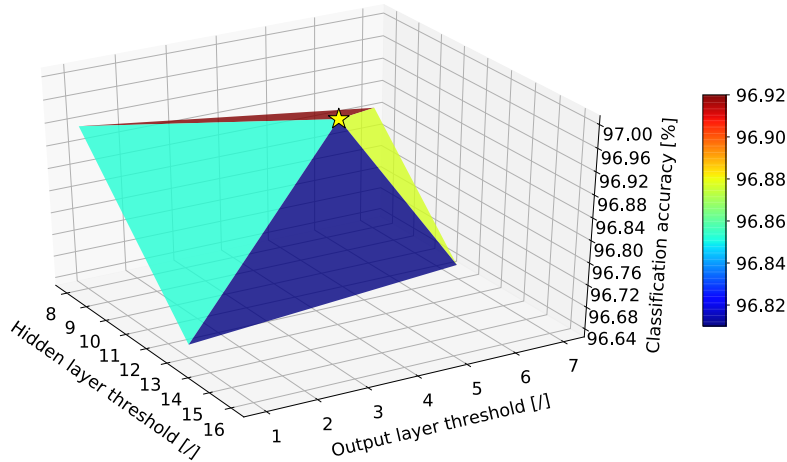


FIGURE 5.23 – AIF implementation. ***Classification accuracy*** on the full test set for similar Θ_m . Brief analysis of the robustness to parameter variation.

5.2.5.4 Hybrid implementation (trade-off)

The hybrid implementation led to almost negligible loss of performance (i.e. 96.83%) but ensures a possible hardware implementation as explained in Section 5.1.5.3. The behavior observed for SSP-MNIST was confirmed with N-MNIST.

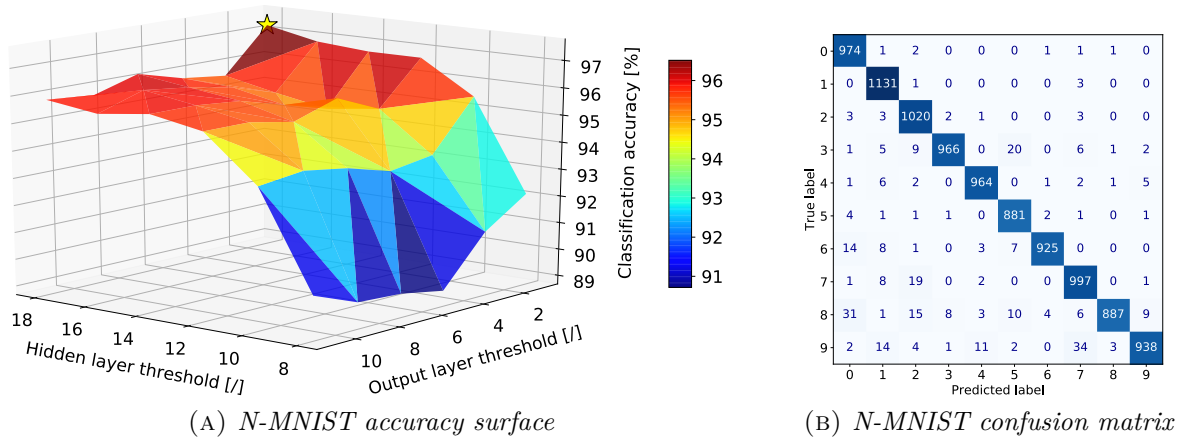


FIGURE 5.24 – *Hybrid implementation. (A) **N-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 2000 samples from the training set for each combination. The yellow star corresponds to the highest accuracy 97.20% on the surface which was obtained for $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 1]$. (B) **N-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 1]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 96.83%.*

5.2.5.5 Accuracy-Latency trade-off in SNNs

Another interesting phenomenon appears when tuning thresholds. Generally, **high thresholds help to improve accuracy at the cost of a higher latency**, because neurons integrate more spikes before reporting the detection of a feature. Indeed, the higher the threshold, the more time needed for a neuron to fire. However, high input rates are usually needed such that enough spikes can be propagated through the network. On the contrary, **if short latencies are important**, low thresholds lead to response after only a few input spikes [4, 11]. In addition, **the number of spikes generated within the network is related to the input firing rate and to the thresholds values**: it increases with higher input rates whereas it decreases with increasing thresholds (weights are considered static). These relations can be observed in Figure 5.25 for the output layer.

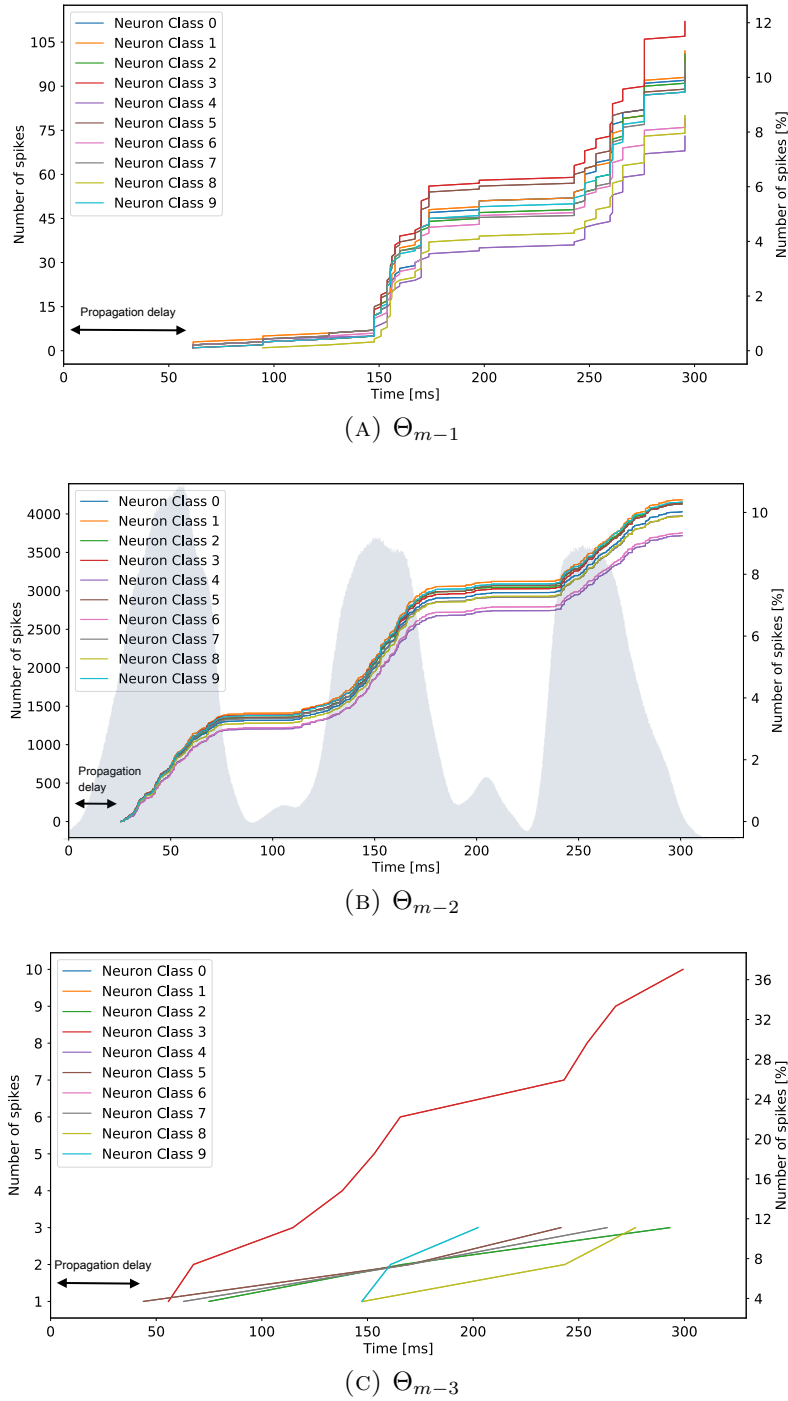


FIGURE 5.25 – **Accuracy-latency trade-off highlighted by inference curves for the hybrid implementation.** It was carried out for three different Θ_m in order to highlight the latency introduced due to higher thresholds. $\Theta_{m-1} = [1, 1, 1, 1, 1][1, 6, 1]$, $\Theta_{m-2} = [1, 0.5, 1, 0.5, 1][1, 6, 1]$, $\Theta_{m-3} = [1, 0.5, 1, 0.5, 1][1, 28, 7]$. The histogram in the background of the subplot (B) can also be observed in Figure 2.19 and allows to observe the three saccades and the corresponding inference. The histogram is given without scale, just to outline the distribution of spikes through time in order to make the link between the spikes distribution and the inference curves.

If latency is low, it is possible to **classify samples after only few input spikes**, meaning that a *consistent output* is already available even if computations are not completely done. Classifying samples after only a few spikes requires that inference curves are available and easy to dissociate since the first spikes. Accuracy is however usually *lowered* as thresholding the number of input spikes is similar to a quantization of the input (loss of information). The influence of thresholding the input number of spikes on the classification accuracy is given in Figure 5.26. It also shows the influence on the simulation time needed to run the inference on the multi-cores spiking simulator ($N_{cores} = 25$). This confirms that SNNs have the opportunity to **trade off classification error rate against the number of spikes in the network** [11], i.e. the number of operations. This is a really *attractive property*, especially for *low-power applications*. For instance, considering only 37% (i.e. 3000 spikes) of the maximum number of spikes (i.e. 8122 spikes) led to a drop of only 1.05% in the classification accuracy whereas simulation runs about 34% faster. A *similar analysis* could be done when truncating the input with respect to time (i.e. only considering a fraction of the sample duration) which would be interesting from a real-time processing point of view.

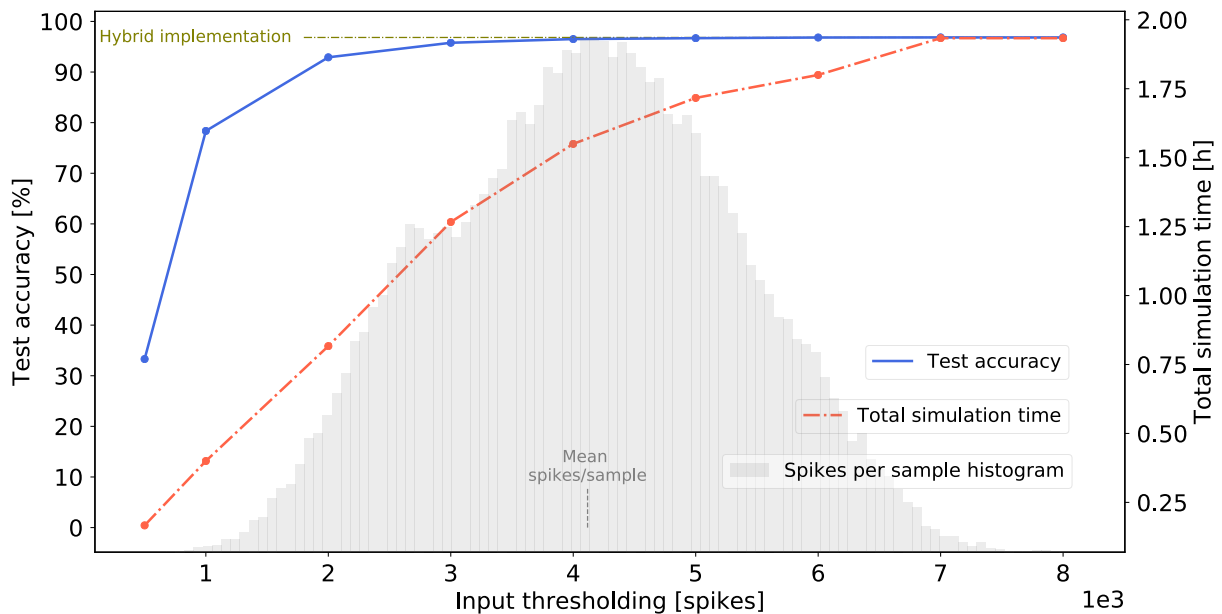


FIGURE 5.26 – **Influence of input thresholding on the classification accuracy.** This was done with the hybrid implementation on the full test set (10k samples). The gray histogram in the background represents the spikes per sample distribution obtained in Figure 2.18. Simulations are run with $N_{cores} = 25$. Mean = 4117 spikes/sample.

5.2.6 Discussion

A critical development achieved with N-MNIST was the **modification of the spiking simulator in order to account for the polarity**. This intermediate step between SSP-MNIST and N-CARS has brought more information about how to account for the ON/OFF polarity inherent to neuromorphic datasets. Two independent channels can be used at the input, i.e. one for each polarity, and the first convolution layer process both simultaneously with two independent filters/kernels banks. This means that two channels images have to be reconstructed as well to train the tailored network. It is similar to RGB images (tri-channels) from a structural point of view.

Another achievement was to understand how to embed a part of the dynamics present in the event-based dataset to its frame-based counterpart. Obviously, some information is lost when converting the dataset. However, analyzing the inference curves at the output of the network suggest that the **spiking network is able to detect features on a very small period of time as well as generalizing invariant features on a longer period of time**. Using convolutions allows to discard isolated noisy pixels as they do not have a high contribution in the convolution. Activation maps in the Conv layers have emphasized what is happening within the network due to movements in the input: they look like a *blurry version* of what is expected over a short period of time. This could be explained by the fact that if a pattern is spotted in the input at a particular time, i.e. a group of neurons fire in a convolution map, this is likely that an adjacent group of neurons will fire just after if the pattern has moved a bit in the input, which is exactly what happens with N-MNIST due to the continuous and repeated saccades as shown in the spatio-temporal representation given in Figure 5.27. This scenario is similar to cases where RNNs [120, 46, 2] are used in conventional machine learning.

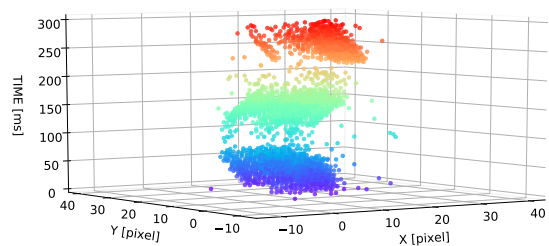
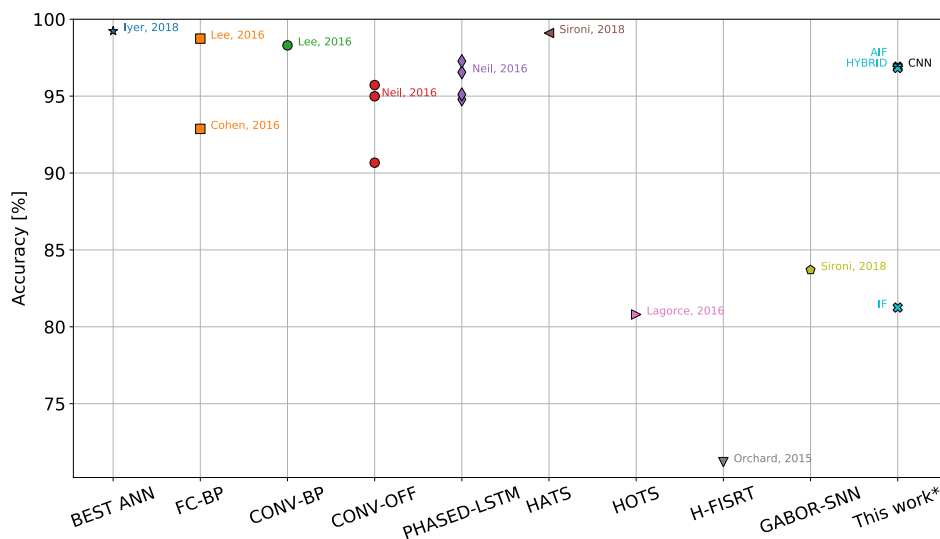


FIGURE 5.27 – *Space-time representation of N-MNIST. 12th training sample, label = 3.*

In addition, **inference curves** provide an interesting tool in order to study the **influence of threshold balancing on classification accuracy performance**. There is no doubt that this balancing step is critical in the mapping approach: using a bad threshold/weights ratio leads *de facto* to poor classification accuracy. However, even if it has been stated that the higher threshold, the better the capacity for the neuron to differentiate between inputs, simulation results came up with a surprising result: the threshold of the output

layer was often 1 when the basic IF neuron model was used in MORPHIC, meaning that the neuron spiked as soon as it received a spike (as the weights are binary on MORPHIC). This behavior leads to an important number of spikes at the output, which is not interesting from an energy consumption point of view. The number of spikes at the output can be consequently reduced (at least one order of magnitude) by choosing a Θ_m with higher output thresholds while affecting moderately the classification accuracy (usually within 5%): this is an **interesting trade-off as it could allow to save power in practice**. Moreover, the ability of SNN to **trade-off classification error rate against the number of spikes in the network** is also a very appealing property for low-power applications.

Finally, the **achieved classification accuracy** confirms that this method is working on the chosen architecture and with the above-mentioned constraints. It seems that our method performs even better than some other convolution mapping approaches that use bigger networks (three alternating layers of convolution with eight channels and 5×5 kernels followed by max-pooling, which were then followed by a FC layer of 256 neurons and finally a FC layer with 10 output neurons) [120] and roughly as good as Phased-LSTM networks [120], in terms of accuracy. However, it is still **difficult to explain** why performance are so little affected by using the basic IF model in the FC layers whereas it affects substantially the performance when it is used in the Conv layers.



*: Cropped version (28x28) of the original N-MNIST dataset (34x34). Network is subject to hardware constraints.

FIGURE 5.28 – ***N-MNIST reported accuracies for spiking networks.*** *BEST ANN = best accuracy reported for ANN network, FC = Fully Connected, BP = Backpropagation, OFF = Offline training (conversion or hardware), CONV = Convolution.* Data compiled in [71] and [31, 87, 92, 120]. The classification accuracy of the IF, AIF and HYBRID implementations are given alongside the one of the continuous-valued CNN.

5.3 N-CARS

5.3.1 Network structure

As for the previous datasets, the rough structure of the network was sketched in Section 4.1. Similarly, parameters M , N , $k_{1 \rightarrow 4}$ and H had to be chosen in order to fulfill both Equation (4.1) and memory requirements.

The **input shape is bigger than for previous datasets** (2000 pixels vs. 784 pixels, i.e. +155%). Therefore, using the same structure as before would result in violating Equation (4.1) as $25 \times 7 \times 9 = 1575 > MorphIC_{fan-in}$. In order to decrease the number of neurons in the last pooling layer, the number of convolution-pooling layers could be increased but this solution was rejected for **two reasons**: the spiking simulator can only handle 2 successive layers of convolutions-pooling and the focus is made on reasonably shallow network (not too deep). **Thereby, only the size of the kernels and the number of channels can be modified**. Using $M = 16$, $N = 40$, and $k_3 = 4$ instead of 2 gives $40 \times 4 \times 3 = 480 < MorphIC_{fan-in}$. The resulting network structure is shown in Figure 5.29. The hardware constraints are the same as previously.

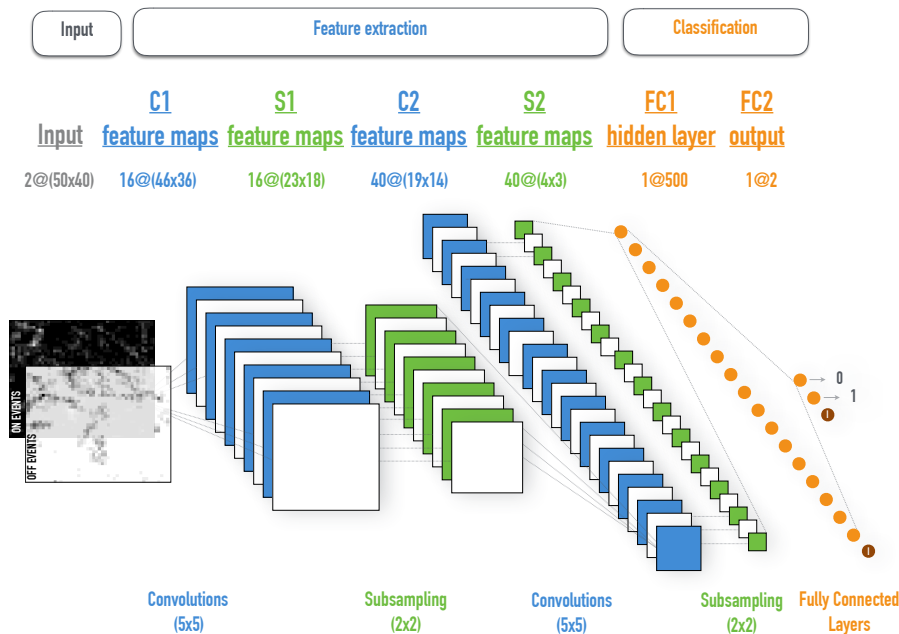


FIGURE 5.29 – *Network structure* used for N-CARS. All connections are not shown.

5.3.2 Frame-based reconstructed dataset

The approach for the reconstruction of the dataset was the same as for N-MNIST, except for the number and duration of the accumulation windows. For N-NCARS, three accumulation periods (e.g. 10, 50, 100 *ms*) were chosen by inspection of the dataset. Consequently, the training set of this reconstructed N-CARS dataset gathers $3 \times 14650 = 43950$ samples and the test set includes $3 \times 8607 = 25821$ samples. The reconstruction of the five first samples of the training set are given in Figure 5.30.

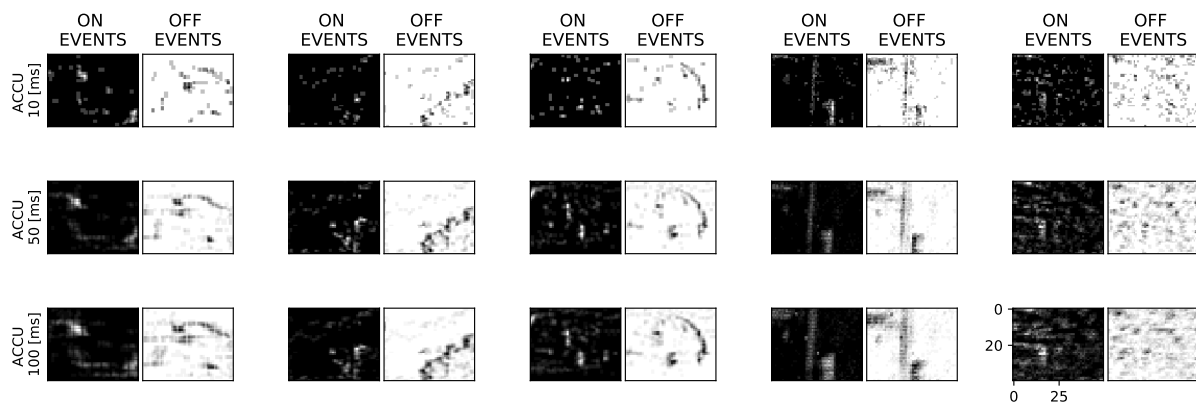
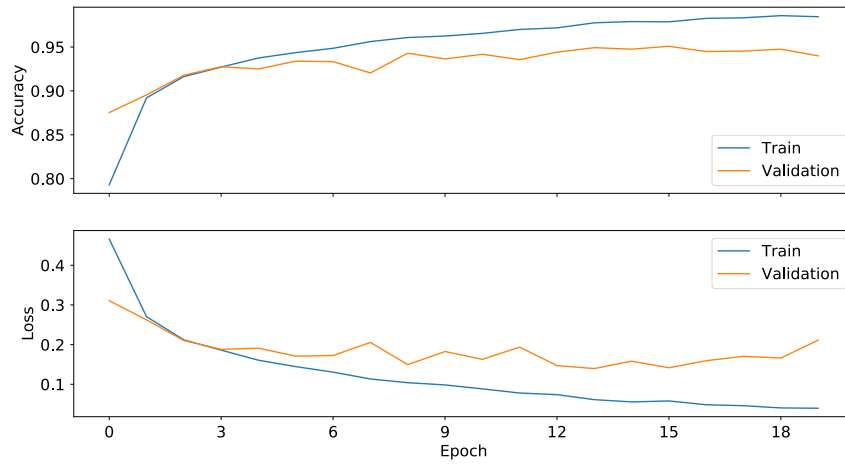
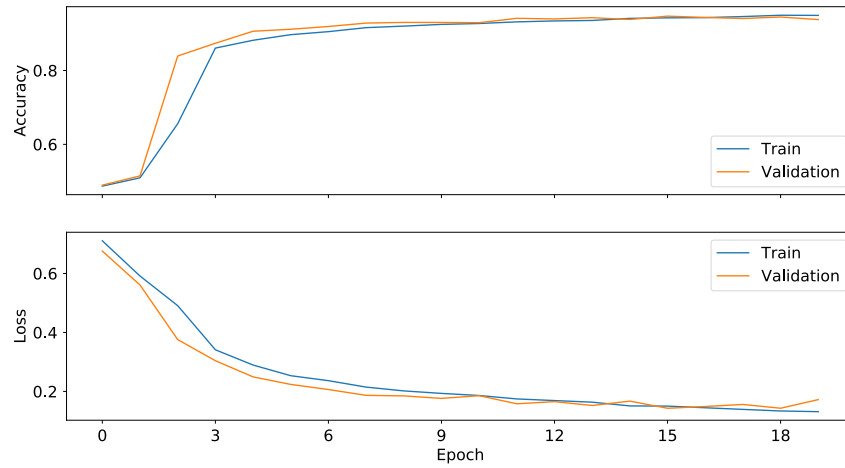


FIGURE 5.30 – *N-CARS frame-based reconstructed dataset*. The samples shown here are the five first samples of the N-CARS test set (one per column). Each row corresponds to a different accumulation time (ACCU) as indicated.

5.3.3 Training on Keras

The same approach as previously was applied regarding the offline training of the weights. As for the two previous datasets, a validation set was created by taking apart 8790 training samples ($\sim 20\%$) from the reconstructed training set (chosen randomly). The reconstructed training set was also shuffled before the validation split. However in this case, **overfitting** was observed as shown in Figure 5.31 and **must be avoided**. It was achieved by using **dropout** regularization technique [4, 49, 63] in the two FC layers in order to improve the robustness of the network [4]. Usually, the dropout method is used with a parameter of 0.5, meaning that half of the connections in the layer are randomly turned off at each training step [4, 63]. However, it has been demonstrated that dropout slows down the learning process [63]. Consequently, we chose a value of 0.3, which is a trade-off solution i.e. 30% of the units are randomly dropped at each iteration. The effectiveness of the dropout technique was clearly demonstrated in Figure 5.32. This never happened for the two previous datasets. It could be explained by the larger network architecture and the smaller dataset.

FIGURE 5.31 – *N-CARS learning curves: presence of overfitting.*FIGURE 5.32 – *N-CARS learning curves, without overfitting. Dropout (0.3) was applied. Test accuracy resulting of this training was 87.2%.*

Best weights were selected as weights leading to the best validation accuracy over all epochs, as for the two other datasets. Eventually, the test accuracy was assessed by using these best weights for inference on the test set, leading to a classification accuracy of **87.2%**. Parameters used for learning are given in Table 5.2.

Epochs	Batch Size	Regularization	Optimizer = Adam			
			η	β_1	β_2	ϵ
20	100	Dropout (0.3)	$1 \cdot 10^{-3}$	0.9	0.999	$1 \cdot 10^{-8}$

TABLE 5.2 – Keras parameters for N-CARS reconstructed dataset learning

5.3.4 Thresholds balancing

The approach was the same as previously even though the limited resolution of the dataset made the comparison between the continuous network and its spiking counterpart less obvious. First Conv-Avg layers underwent the same issue as for N-MNIST. In the last Conv-Avg layers, it was difficult to find convincing similarities. It was a bit easier in the hidden layers, as shown in subplots (C) and (D) in Figure 5.33.

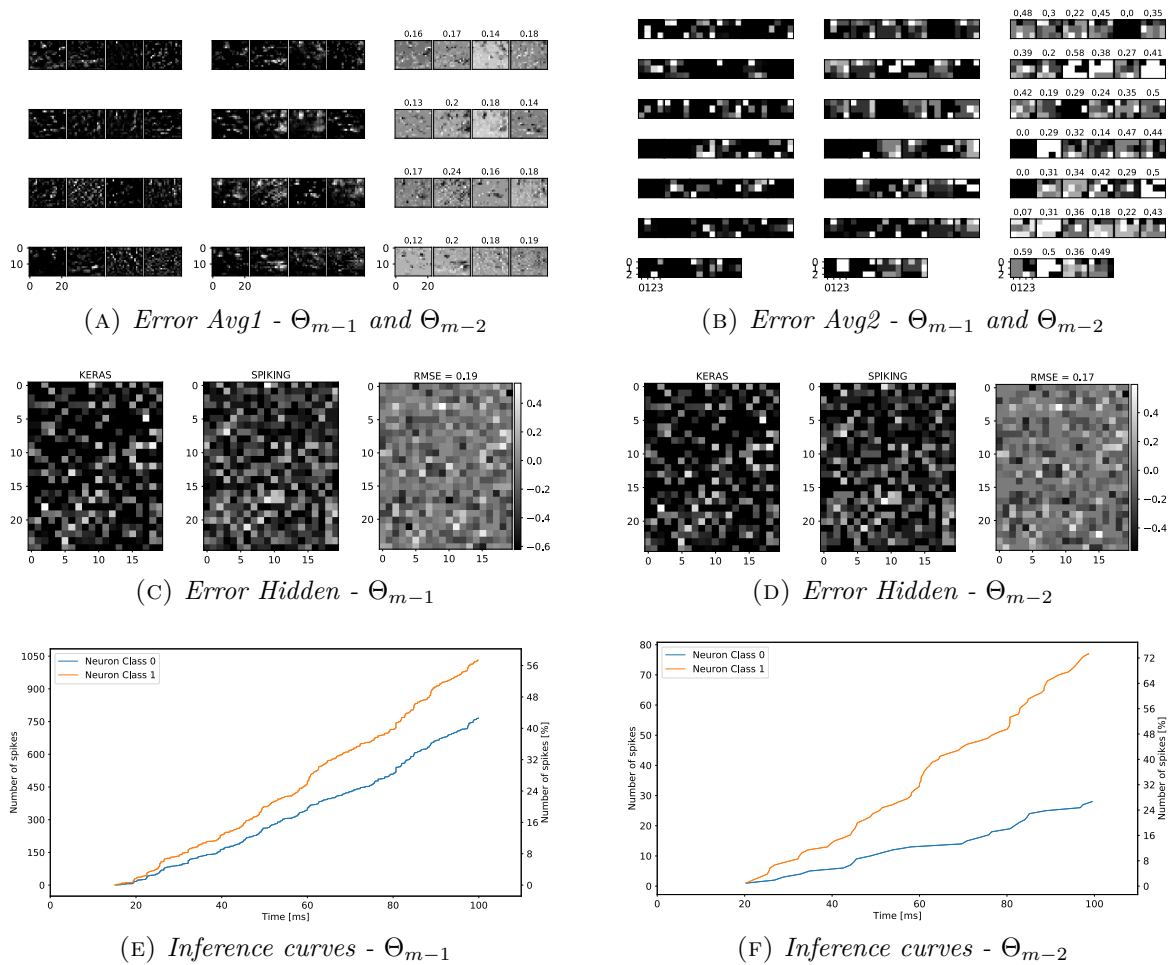


FIGURE 5.33 – **Error in Conv and FC layers.** In each sub-figure, the first set of activation maps is the output of the layer on KERAS. The second set is the activation maps of the corresponding layer in the spiking simulator and the last set is the difference between the two first sets, activation map by activation map. RMSE is used as the error metric. This was done here for the third sample of the N-CARS training set (label = 1). Only the hybrid implementation was considered with $\Theta_{m-1} = [1, 0.25, 1, 0.5, 1][1, 8, 1]$ and $\Theta_{m-2} = [1, 0.25, 1, 0.5, 1][1, 10, 5]$.

5.3.5 Results

This section presents results obtained on N-CARS. It is organized similarly as the SSP-MNIST results, Section 5.1.5. As this datasets offers a binary classification challenge, the chance level corresponds to 50% classification accuracy whereas it was at 10% for the two previous handwritten datasets.

5.3.5.1 IF implementation (basic IF neuron model)

This simulation was the **first step in the N-CARS inference**. The same methodology as for the other datasets was carried out. The threshold vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 20, 4]$ was the best Θ_m over the grid search presented in the first subplot of Figure 5.34, leading to **68.71%** classification accuracy. Despite it is above chance level, it is far from the performance of the continuous-valued tailored network. In addition, there are more mistakes at classifying cars as backgrounds than the contrary, as shown in the confusion matrix given in the subplot (B) of Figure 5.34. The basic IF neuron model led to similar results (i.e. bad mapping) in previous simulations. It is worth noting that for the first time, the thresholds vector Θ_m leading to the best classification accuracy satisfy $\theta_m^{Output} \neq 1$ whereas MORPHIC uses the basic IF neuron model.

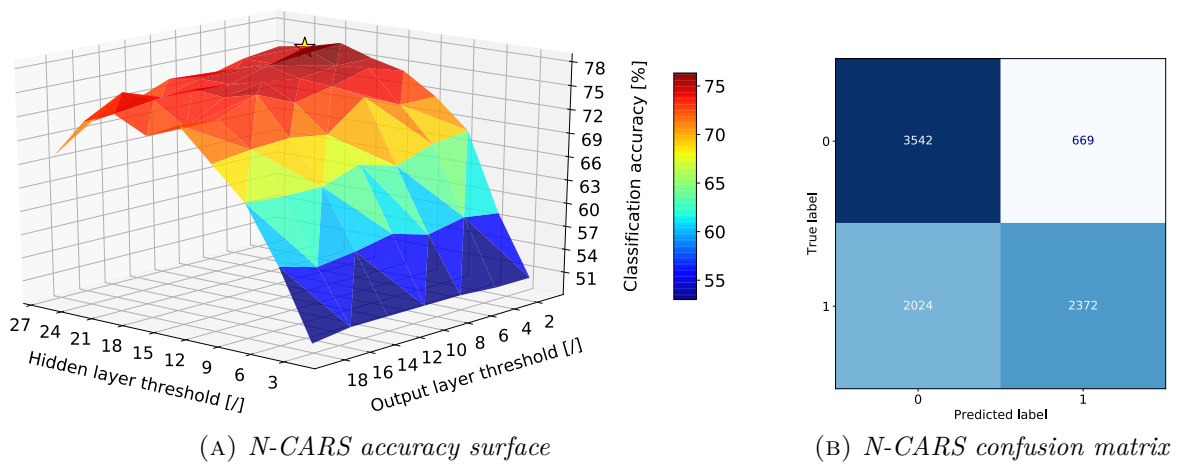


FIGURE 5.34 – *IF implementation. (A) N-CARS accuracy surface, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 300 samples from the training set for each combination. The yellow star corresponds to the highest accuracy 77% on the surface which was obtained for $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 20, 4]$. (B) N-CARS confusion matrix, for the best thresholds vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 20, 4]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 68.71%.*

5.3.5.2 AIF implementation (adapted IF neuron model)

As observed for the handwritten datasets, using the adapted IF neuron model led to a sharp increase in mapping quality and performance, as shown in Figure 5.35. Choosing $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 5, 19]$ for carrying spiking inference on the full test set gave **82.25%** classification accuracy. **For the first time, the gap between the SNN and its continuous-valued counterpart cannot be filled.**

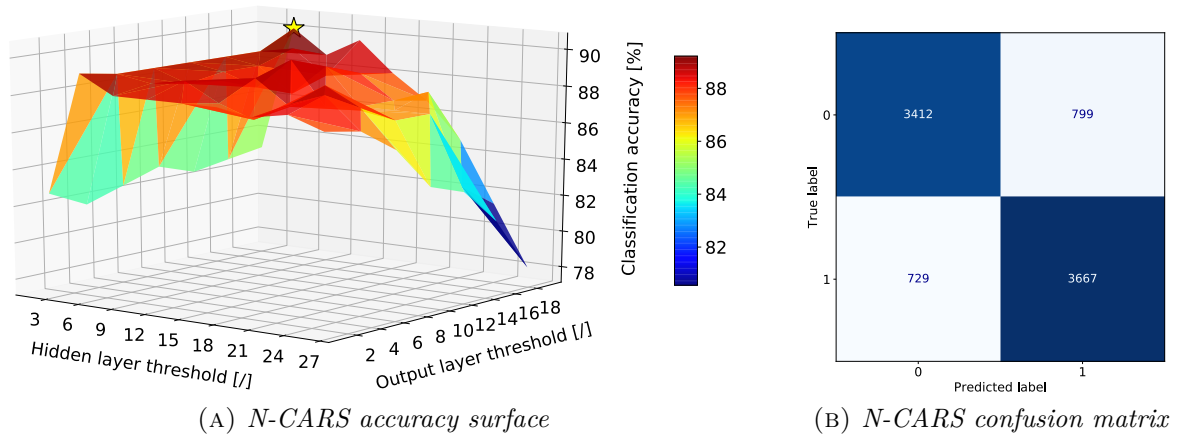


FIGURE 5.35 – AIF implementation. (A) **N-CARS accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 300 samples from the training set for each combination. The yellow star corresponds to the highest accuracy 92% on the surface which was obtained for $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 5, 19]$. (B) **N-CARS confusion matrix**, for the best thresholds vector $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 5, 19]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 82.25%.

5.3.5.3 Hybrid implementation (trade-off)

The hybrid implementation led again to almost negligible loss of performance (i.e. **81.97%**, Figure 5.36) but ensures a possible hardware implementation.

The **impact of thresholding** the input number of spikes was also investigated and the result is given in Figure 5.37. Truncating the input to 3000 spikes (i.e. 30% of the maximum number of spikes) can be considered without causing a significant drop (i.e. 1.5%) in the classification accuracy. This allows to decrease the simulation time of about 60% and also the number of spikes in the network which is related to energy consumption. Furthermore, this demonstrates again that **classification can be performed using a restricted number of input spikes** and within a shorter time interval. The **accuracy-latency trade-off** is again well illustrated in Figure 5.37.

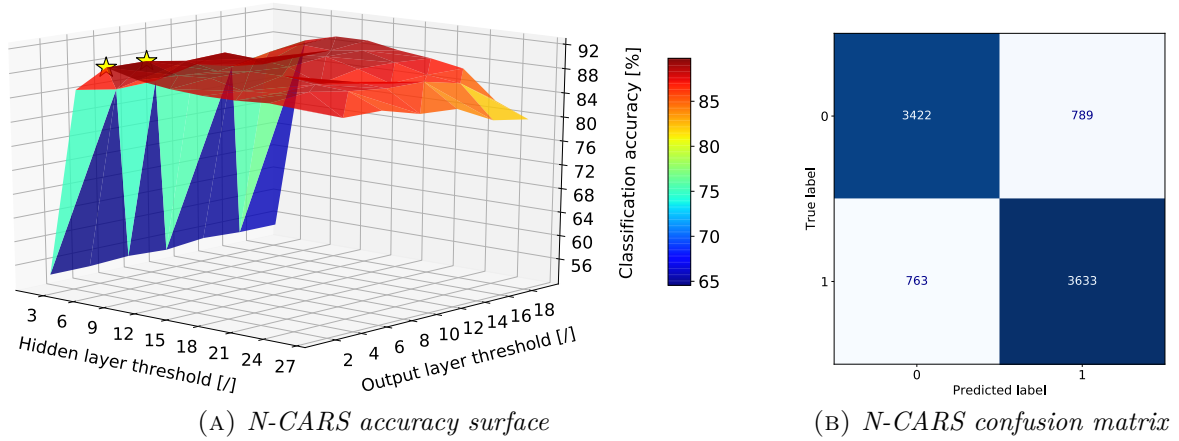


FIGURE 5.36 – *Hybrid implementation. (A) N-CARS accuracy surface*, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 300 samples from the training set for each combination. The yellow stars correspond to the highest accuracies 90.33% on the surface which was obtained for $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 8, 1]$ and $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 8, 4]$. (B) *N-CARS confusion matrix*, for the best thresholds vector $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 8, 1]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 81.97%.

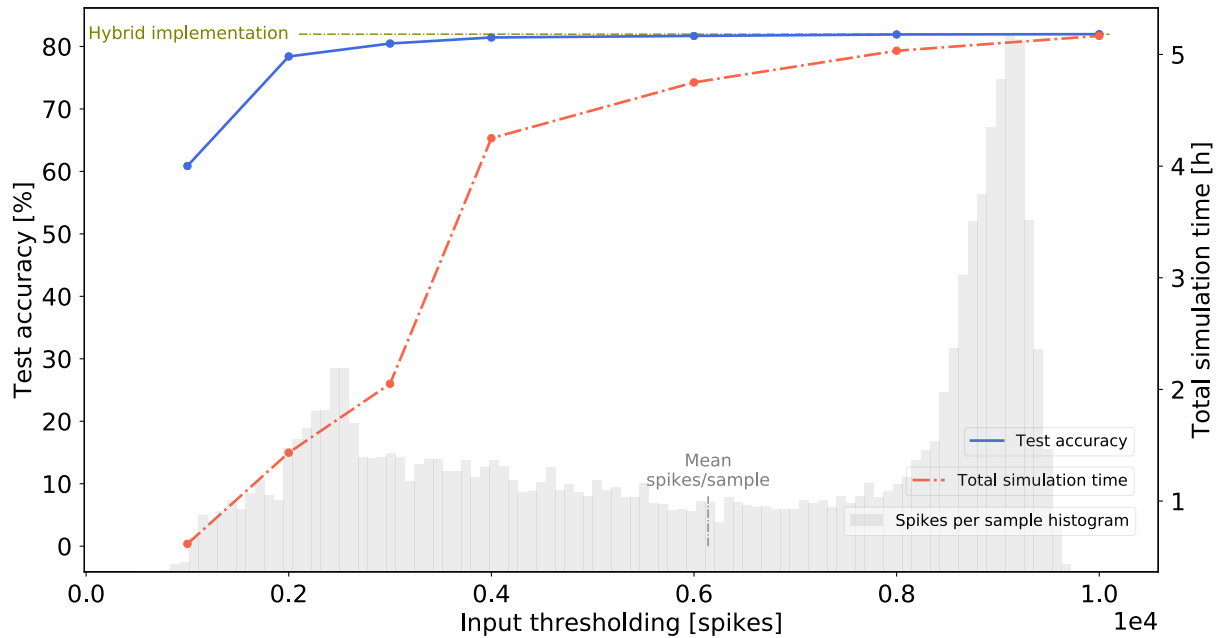


FIGURE 5.37 – *Influence of input thresholding on the classification accuracy.* This was done with the hybrid implementation on the full test set (14650k samples). The gray histogram in the background represents the spikes per sample distribution obtained in Figure 2.22. Simulations are run with $N_{cores} = 25$. Mean = 6139 spikes/sample.

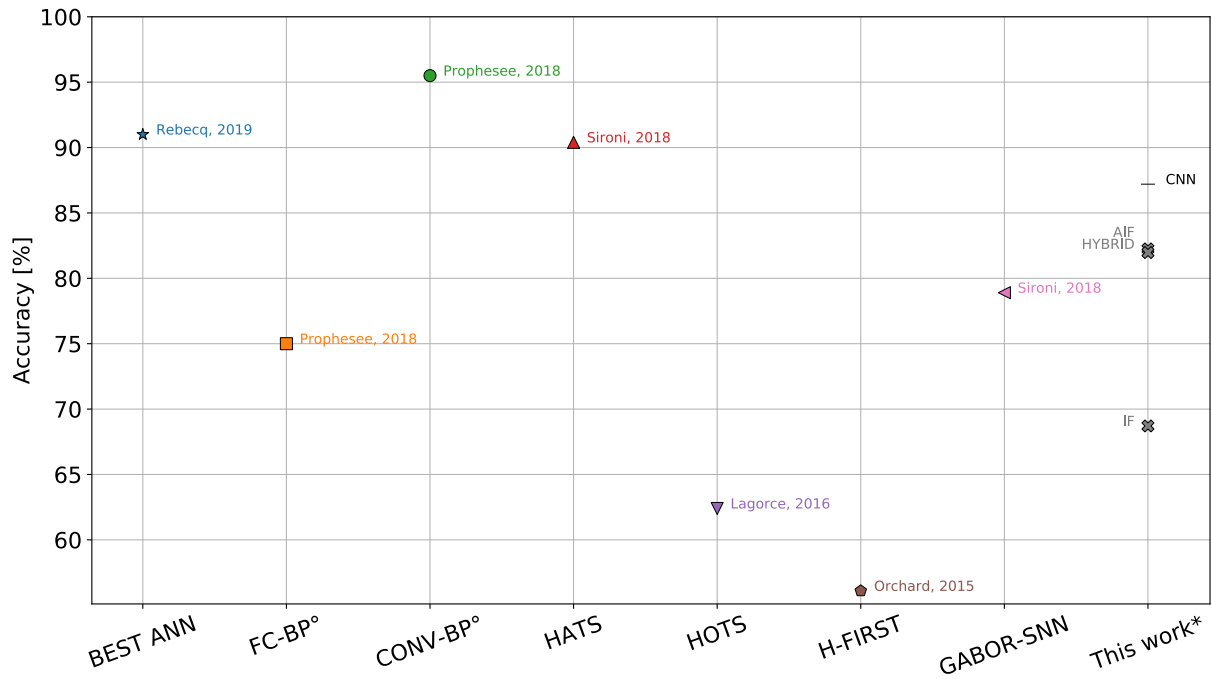
5.3.6 Discussion

Applying our method on N-CARS was the **last step of the process** proposed in this thesis. The same approach as for the two previous datasets was carried out. However, it was the first for which **overfitting** occurred clearly during the training of the tailored network. We think it could be explained by the fact that the network was expanded whereas the number of training samples is smaller, leading to overfitting. Using the **dropout** technique as regularization solved this issue.

Even if the resulting classification accuracy was satisfying (i.e. 81.97%), it was the first time that such a big **performance gap** was present at the end of the mapping process, i.e. about 5% mismatch. We suppose that it is related to the more complex nature of N-CARS which has a real-world dynamics. That was not the case for SSP-MNIST or N-MNIST. Consequently, dynamics beyond ideal mean-rate modelling cannot be embedded for training the continuous tailored network, possibly leading to mis-classification when back to the spiking domain. Another reason could be that the tailored network "boosts" its performance using the images reconstructed with big accumulation windows which are more difficult to detect in the spiking domain as it needs a longer integration time.

Regarding the **structure of the network**, improvements could be made. Indeed, as there are fewer output neurons compare to MNIST dataset, H could be set to $512 - 2 - 2 = 508$ instead of 500 (i.e. +1.6% neurons in the hidden layer). However, it would probably not have a significant impact on the final classification accuracy.

Finally, we have demonstrated that **our method was working on N-CARS**, leading to a classification accuracy of 81.97%. This was the first time that this kind of approach was used on N-CARS, to the best of our knowledge. This result can be compared to state-of-the-art results on N-CARS in order to have a global view of current performance on this dataset, even though the method presented in this work is constrained by hardware limitations and uses the UCLouvain version of the dataset. Our mapping approach gave **better results** (up to 82%) than the FC network proposed by PROPHESSEE and trained with spiking backpropagation (about 75%). However, their network architecture (4000-100-2) was really smaller than the one presented in this work. Our method also performs slightly better than the Gabor-SNN approach (78.9%) presented in [31] which however also uses a smaller network. However, the ConvNet proposed by PROPHESSEE and trained with spiking backpropagation reaches 95.5% classification accuracy which is **far better than our method** whereas they use a bigger network, i.e. two Conv layers (with 20 and 50 channels respectively) followed by two FC layers with 500 and 2 neurons, respectively.



*: UClouvain version of the original N-CARS dataset. Network is subject to hardware constraints.

°: Results not published but kindly shared by a member of PROPHESEE.

FIGURE 5.38 – **N-CARS reported accuracies** for spiking networks. *BEST ANN* = best accuracy reported for ANN network, *FC* = Fully Connected, *BP* = Backpropagation, *GABOR-SNN* = Gabor [151, 152, 153, 154] filters for Conv layers followed by SNN training with BP. Data compiled in [30, 31, 39]. The classification accuracy for the three combinations (*IF*, *AIF* and *HYBRID* implementation) proposed in this section are given. The continuous-valued CNN classification accuracy is also shown in order to compare the classification accuracy before and after mapping.

5.4 Hardware implementation: careful considerations

This section aims at presenting considerations regarding the potential hardware implementation of the proposed solution. This focus on results obtained with the hybrid implementation, i.e. adapted IF neuron model in the Conv layers and basic IF neuron model in MORPHIC.

5.4.1 Resources distribution

In a ConvNet architecture, the majority of neurons is usually located in the Conv layers as multiple channels are needed to extract several features. However, most of the synaptic weights are located in the FC part of the network. Indeed, this can be observed in Figure 5.39 for both N-MNIST and N-CARS network: more than 95% of neurons are located in the Conv layers whereas roughly 95% of synaptic weights are in the FC layers.

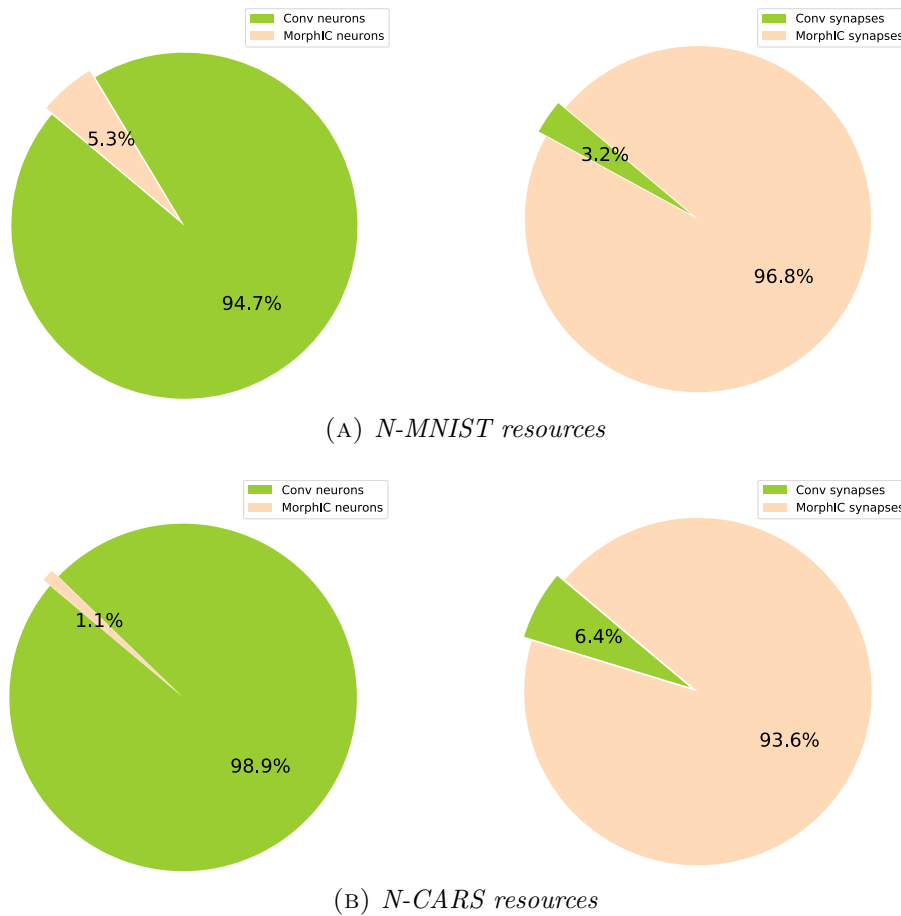


FIGURE 5.39 – **Hardware resources** needed by neural networks proposed in this work for (A) N-MNIST and (B) N-CARS.

5.4.2 Influence of the weights quantization on the classification accuracy

In order to reduce the memory footprint of the proposed networks, we investigated the influence of the weights quantization on the classification accuracy. As the resolution of neurons and synapses cannot be modified on MORPHIC, the analysis was carried out *only* on the Conv part of the network. **Synaptic weights were quantized** to different *fixed-point* resolution (e.g. 1, 2, 4, 8, 16 and 32 bits) and the inference was then performed without any intermediate threshold balancing. This was done for both N-MNIST and N-CARS test set with respective networks and thresholds vector Θ_m found in previous sections. Figure 5.40 presents the results of this investigation for the hybrid implementation. For N-MNIST network, it can be observed that classification accuracy dropped dramatically below 4 bits of resolution and a similar behavior was observed with N-CARS (below 8 bits). Above these boundaries, classification accuracy was almost the same (within 1%). Hence, this suggests that **considerable memory savings** (i.e. at least four times less storage needed per synapse) can be achieved [17] by **quantizing the weights in Conv layers almost without any classification accuracy loss**.

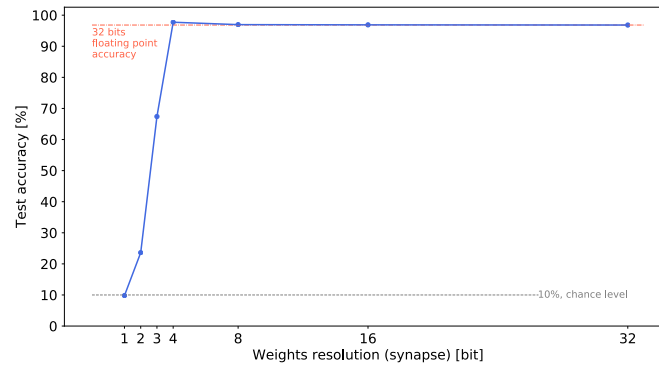
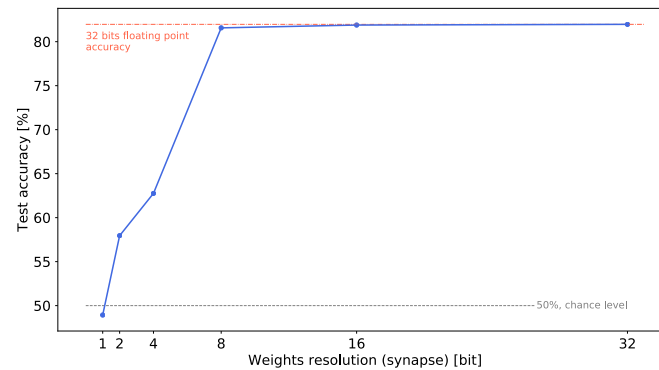
(A) *N-MNIST accuracy*(B) *N-CARS accuracy*

FIGURE 5.40 – ***Influence of the quantization of the Conv weights on the classification accuracy for (A) N-MNIST and (B) N-CARS.***

The **distribution of quantized weights** can be compared to the original distribution before quantization as shown in Figure 5.41. Key characteristics are conserved. By visual inspection of Figure 5.42, it is possible to assess that kernels looks similar, there is just *less granularity* after quantization. More Figures are given in Appendix A.2.6. More in-depth analysis can be found in [155].

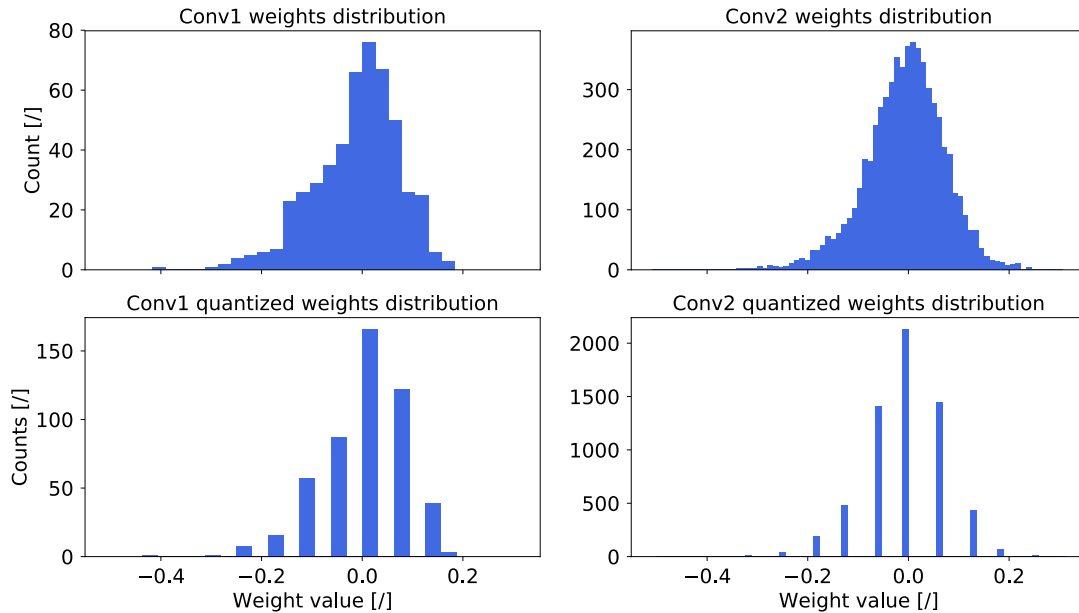


FIGURE 5.41 – *Comparison between original and 4-bits quantized weights distributions for the N-MNIST network.*

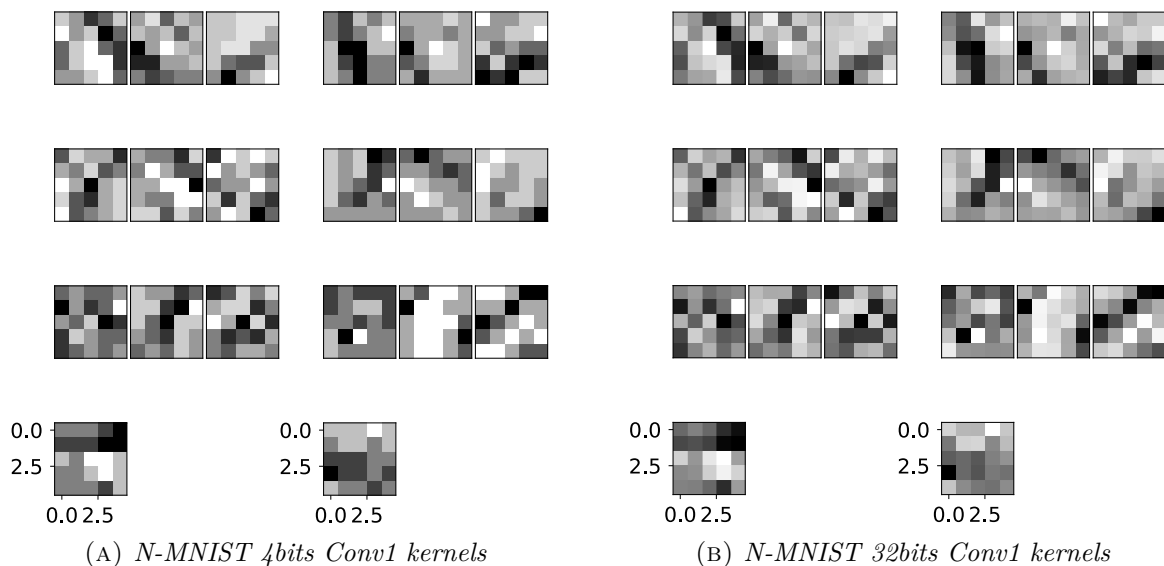


FIGURE 5.42 – *Influence of the quantization on the Conv1 weights of the N-MNIST network for (A) 4 bits resolution and (B) 32 bits resolution. OFF/ON channels are shown for each kernel.*

5.4.3 Memory and power considerations

In order to **choose the best FPGA device** for this application, the power consumption had to be minimized as the focus is made on **ultra-low-power** systems. However, the more sophisticated the SPARTAN-VII model, the higher the power consumption, Appendix A.3. Nevertheless, the chosen model had to provide enough memory such that all weights and neuron states could be stored. Figure 5.43 shows how much memory is needed for both N-MNIST and N-CARS networks, depending on the resolution of neurons and synapses. As the number of neurons is much higher than the number of synapses in Conv layers, it was expected that the **resolution of neurons has more impact on the total memory needed** for the Conv part. This was confirmed in Figure 5.43. The storage capacity (RAM) of several SPARTAN-VII models is displayed in gray dashed lines on the same Figure and allows to chose the most suitable device. In Section 4.1.1, we chose the SPARTAN-VII (XC7S100) as a first step but this is the largest model of its family, leading to a *suboptimal choice*. The SPARTAN-VII (XC7S50) seemed to be a *better candidate* because:

- it provides enough memory (i.e. 2700KB) for a wide range of neurons and synapses resolution.
- it has a low static power as shown in Figure 5.44.
- it can be used in low-power mode (-1L speed grade devices) in order to decrease both static (70%, Appendix A.3) and dynamic (50%, Appendix A.3) power
- it is designed for low-power applications

The SPARTAN-VII (XC7S25) could have been chosen but it gives less security margin. As it is very likely that practical limitations related to FPGA programming and architecture will require more memory or functionalities, it is common practice to **provide a bit of security margin**.

Another quantity of interest would be a **coarse approximation of the operating frequency** at which the FPGA should work in order to ensure real-time processing of such datasets. Indeed, the dynamic power is proportional to the clock frequency (i.e. $P_{dynamic} = CV^2 f_{switching} \alpha$, where C is the parasitic capacitance, V is the supply voltage, $f_{switching}$ is the switching frequency and α the switching rate A.3) which means that the higher the operating frequency, the higher the power consumption. However, the more operations in the network, the higher the operating frequency to ensure real-time processing.

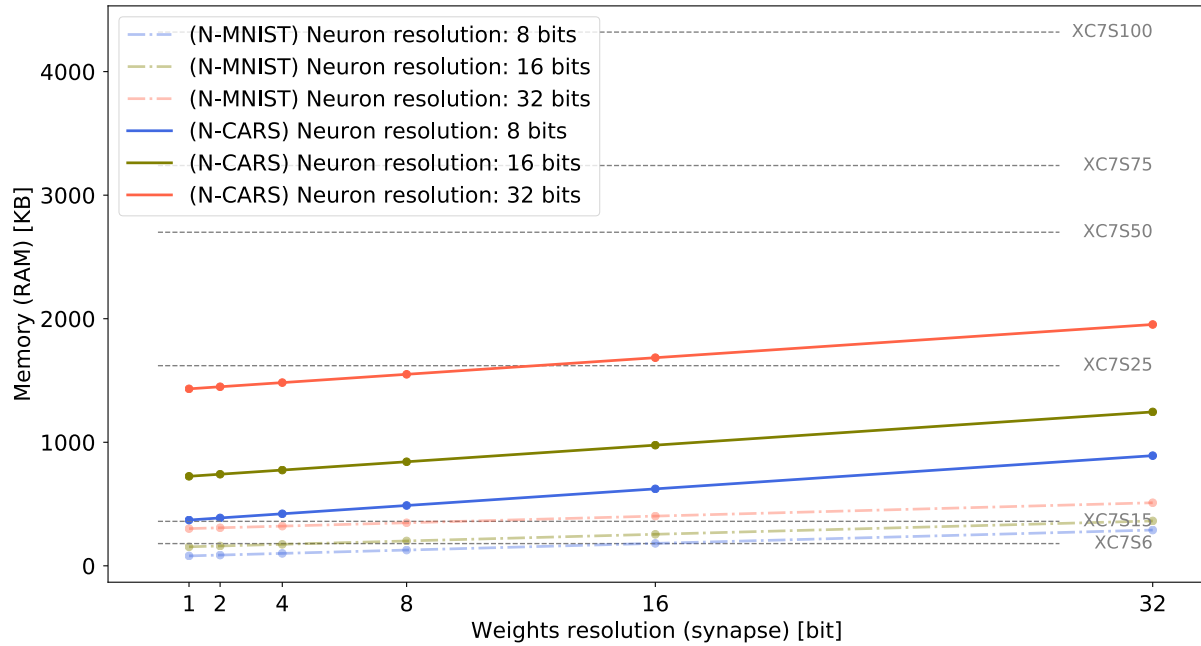


FIGURE 5.43 – *Estimation of the evolution of the memory needed with respect to the resolution of the neurons and synapses in the proposed networks. The analysis was carried out for N-MNIST and N-CARS networks. The gray dashed line represents the memory (RAM) available on different models of the SPARTAN-VII device. These results are given as a coarse estimation of the necessary storage (lower bound): practical limitations related to FPGA programming should be considered to have a better estimation. More details about the estimation are in given in Appendix A.2.4.*

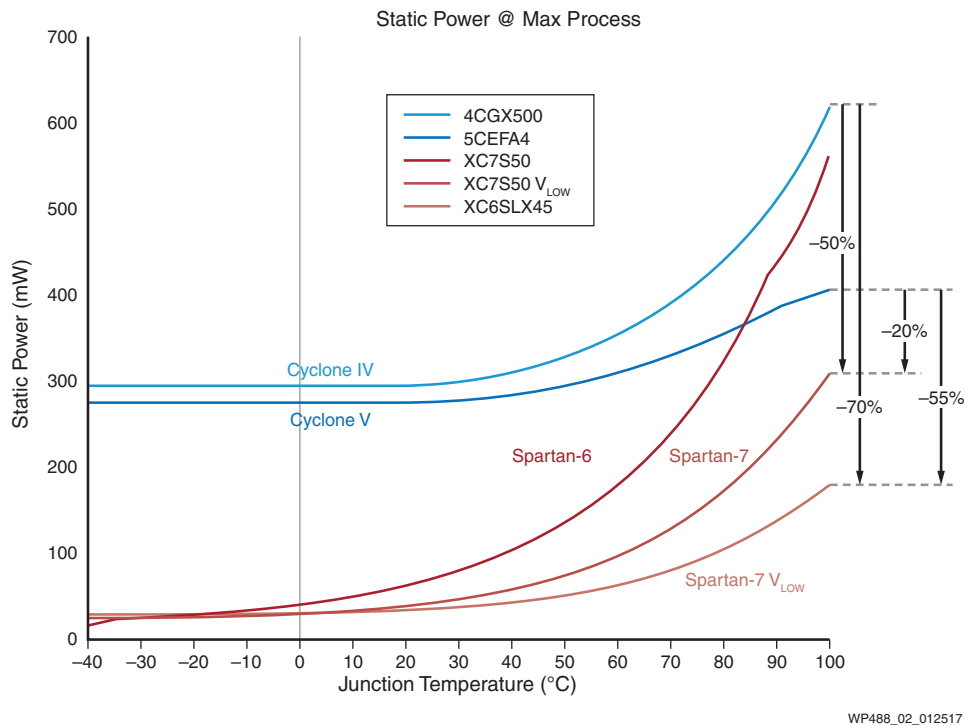


FIGURE 5.44 – *Evolution of the static power for different FPGA devices. Image from SPARTAN-VII data sheets, Appendix A.3.*

5.5 Comprehensive discussion

The **first lesson to point out** is that for all three datasets, the mapping was not of good quality using the basic IF neuron model. In fact, the grid search algorithm led each time to relatively bad Θ_m , preventing the SNN from performing efficiently. This issue has highlighted the impact of the chosen neuron model on the classification accuracy. Indeed, using the AIF neuron model has considerably increased the mapping quality and thus the classification accuracy as highlighted in Figure 5.45. Moreover, it has been observed that using this AIF model only in the Conv layers was sufficient to enhance the mapping quality, which allows a possible mapping of FC layers on MORPHIC.

The **second important lesson** is that the threshold balancing step is critical in the mapping method. Choosing a bad Θ_m for the SNN leads *de facto* to bad classification accuracy. Finding good Θ_m has been done by *computationally intensive grid search* over a space of Θ_m . Using a proper threshold balancing algorithm would definitely be a better approach. Nevertheless, the grid search algorithm gave us a valuable result: even if an optimal Θ_m is expected to exist, sub-optimal Θ_m are relatively easy to find using this method and lead to a decent mapping quality. Classification accuracy decreases but often *do not collapse* for Θ_m close to the sub-optimal Θ_m , suggesting that the proposed strategy is robust to parameter variation.

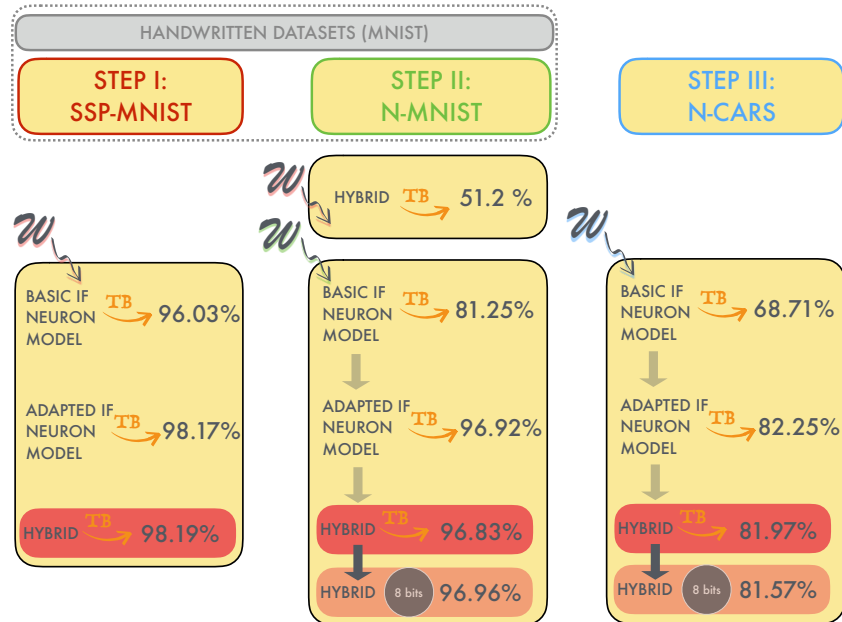


FIGURE 5.45 – *Summary of the classification accuracy performance for each step of the proposed strategy. Key improvements are highlighted. Symbols details are given in Figure 3.6.*

The **analysis regarding the weights quantization** has demonstrated that the classification accuracy could be maintained within 1% while performing quantization of the weights belonging to the Conv layers. This is true for both N-MNIST and N-CARS networks when the Conv weights are quantized with a resolution of *8bits fixed-point*. This improvement allows for *savings in footprint memory and power consumption*. In addition, the ability of SNNs to **trade off classification error rate against the number of operations in the network** [11] is another *attractive property* for *low-power applications* that has been demonstrated in this work, for both N-MNIST and N-CARS.

Then, we want to give **objective considerations** regarding one of the main challenges on which machine learning focuses: getting the best possible results on one type of metric: **accuracy** [1]. However, this single metric cannot summarize the range of different realities and performance characterizing a given method, especially when hardware implementation is considered. The overwhelming majority of research on machine learning focuses only on the accuracy and often forgets about other important metrics such as the number of operations needed to make the system work, the energy needed to train and use the network or the memory footprint and the real-time constraints that limit implementation on autonomous and portable devices [1, 155]. To give an idea of how much resources are necessary for such networks [1], the LeNet-5 [61] Deep Neural Network (DNN) performs 5.74 million floating-point arithmetic operations (MFLOPS) in order to classify a 28x28 image and the *VGG – 19* [156] architecture requires 43 times more computation than LeNet-5, which is too expensive to be mapped on dedicated hardware.

Finally, whereas the goal of conventional approaches is to maximize accuracy, the **neuromorphic approach** aims at developing real-time and ultra low-power systems for decentralized sensing and processing (e.g. wireless sensor nodes). **Spike encoding** is not just an alternate mechanism for representing static information [87], even though it is used in this way throughout this work (rate-based SNN, Poisson spike train with a rate proportional the intensity of a pixel, etc...). Biological brains have developed computing mechanisms that are specialized at representing and processing multi sensory information from a dynamic world. **Spike encoding has undoubtedly a key role in this biological efficiency and better algorithms are needed in order to leverage spike sparsity.**

Conclusion

This thesis **focuses** on how to *train* SNNs in order to *classify* accurately samples from real-world neuromorphic datasets. The spiking dataset N-CARS was used as a benchmark. At the same time, neuromorphic processor(s) developed at UCLouvain came as a support hardware framework, which defined *chip related constraints*.

For this work, we chose to use a **CNN architecture** (two conv-pooling layers followed by two FC layers) and a **mapping approach** as *offline training method*. This was motivated by practical constraints and by the fact that this approach had never been investigated on N-CARS, to the best of our knowledge. In order to develop the necessary tools and to acquire experience in the neuromorphic field, a **three-step strategy** was proposed:

- First, the development of a *spiking simulator* using a *synthetic spiking dataset* generated especially for this work, i.e. SSP-MNIST.
- Second, the use of a neuromorphic well-known version of the handwritten MNIST dataset (i.e. N-MNIST) to *extend* the spiking simulator and to gain more insight on how to reach satisfactory performance on spike-based data with SNNs.
- Finally, gather lessons from the two first steps in order to tackle the *main challenge*, i.e. perform an accurate inference (i.e. $> 80\%$) in the spiking domain using the N-CARS dataset.

Then, *practical considerations* were investigated as a first step towards a hardware implementation using a FPGA SPARTAN-VII (XC7S50) and the UCLouvain neuromorphic chip MORPHIC.

This approach led to decent classification accuracies on the different datasets, i.e. 98.19% on SSP-MNIST, 96.83% on N-MNIST and 81.97% on N-CARS. However, classification accuracy is *not* the only metric that matters and it is important to have a **critical opinion on the approach proposed in this work**. **On the one hand**, this approach is relatively *easy* to implement and it leads to *reasonable* results at the end of the process. In addition, it is a good method to assess the importance of the *temporal information*

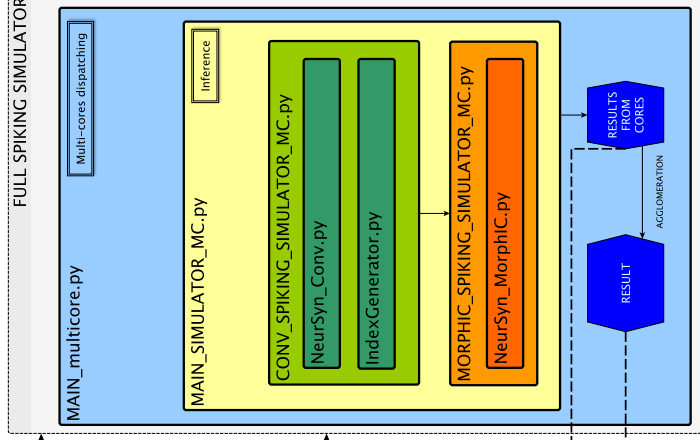
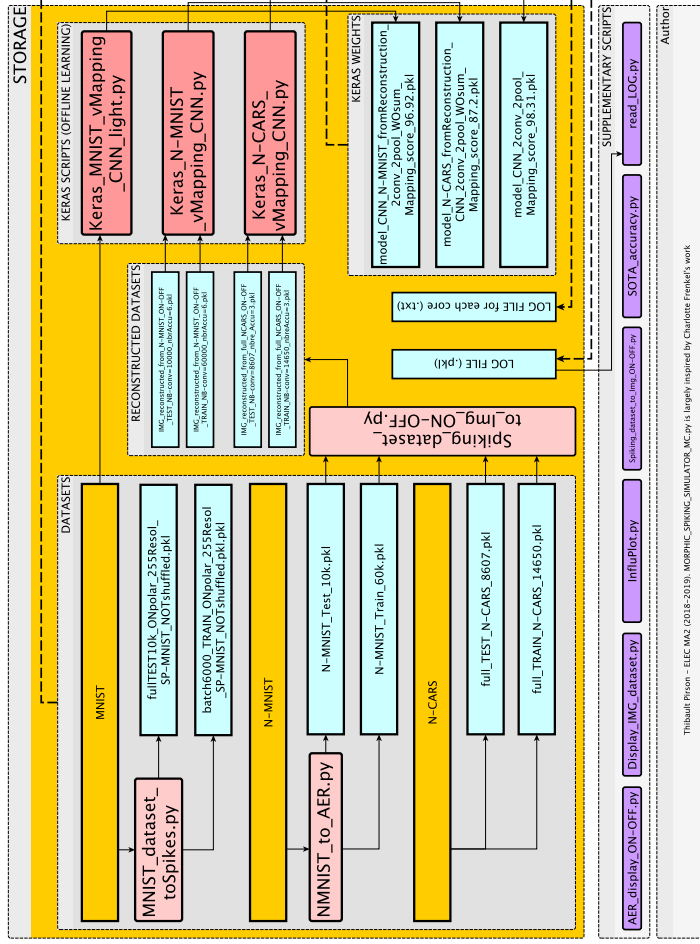
contained in a given dataset. Indeed, as this method uses a rate-based CNN-to-SNN mapping, it suggests that leveraging precise timing information in these specific datasets is not required to reach satisfactory classification accuracy. **On the other hand**, the grid search approach is really *computationally intensive* and does *not* guarantee to obtain an optimum. Moreover, the proposed architecture only uses a single core of MORPHIC and needs a dedicated platform for the convolution part, which results in a *sub-optimal* use of currently available hardware resources. A dedicated accelerator would be more appropriate. Furthermore the mapping method is an *hybrid approach*: it has a foot in both worlds, i.e. the continuous and spiking domains. Accordingly, training is *not* energy efficient and far from bio-inspiration: we only use neuromorphic devices as a support to implement algorithms that were developed for conventional machine learning applications. In order to obtain an end-to-end neuromorphic approach, spike-based processing should be used for both training and inference [32].

Clearly, it is tempting to reuse the *background* developed in machine learning and computer vision over the last decades to **shape the basis of neuromorphic vision** [29]. However, neuromorphic vision relies on very different data encoding and processing primitives and it is calling for its *own* architectures, algorithms and metrics. This is why biologically inspired learning rules were presented in this work: even if they do not offer sufficient performance for now, they are a first step towards the development of more biologically inspired algorithms. Apart from that, it has been observed very frequently that the only focus of much machine learning researches was to outperform state-of-the-art *accuracy* on benchmark datasets. Using only this single metric is not very informative as it does not account for the energy cost, the complexity of the method or the real systems constraints [1]. We are deeply convinced that it would be more representative to compare the different methods using a *cost function* that captures the aspects related to power, performance and area of inference devices.

To **conclude**, neuromorphic engineering has to focus on getting closer to *biological capacities* in order to pave the way for new technologies but also in order to have a better understanding of the working principles of biological systems. It is about *real-time interaction* between the environment and the algorithm supported by physical implementation [18], especially for portable and autonomous devices. This includes not only reaction to stimuli but also *cognition*, which is a wide topic falling out of the scope of this thesis. How can biological systems transform the perceptions of the world into a form adequate for the manifestation of cognition [17] and how can such systems infer unknown information from incomplete sensory data [17] are deeply interesting queries. Neuromorphic engineering could lead to efficient *alternatives* over current methods regarding these challenges.

Appendix

A.1 Architecture of the Python scripts



COMMANDS	
thresh_MIC	Membrane thresholds for MIC layers
rank_threshold	Membrane thresholds for Conv layers
rank_thresholding	Number of input spikes to consider
test_len	1 = Enable rank_threshold
nbre_cores	Number of samples for inference
activate_polarity	Number of cores to run the inference
ThreshNeg	1 = uni-core version / > 1 = multi-cores
	0 = enable 2 channels polarity at the input
	1 = only the ON polarity is considered
	[A,B] : A = 1 to use the AIF neuron model in the Conv layers, else IF model
quantizer	B = 1, same but for FC layers
	[A, B] : A = 1 to enable Conv weights quantization, B = number of bits
WTa_MIC	Intensity of the WTA coefficient
device	'MAC' or 'UCL', device that runs the inference
save	1 = enable saving the .pkl log file with results
print_file	1 = enable saving intermediate .txt log file for each core and disable informations to be printed in the console
display_graph	1 = enable plots displaying, works only with the uni-core version
sample_selected	Sample selected for detailed plots
nbre_to_display	Number of samples to show the plots for
num_neur	Neurons architecture in the Conv layers
kernel_chan	Channels architecture in the Conv layers
kernel_size	Kernel size in the Conv layers
nbre_output_class	Number of output neurons
name_pkl_weights_to_load	Name of the file containing the Keras weights
name_pkl_dataset_to_load	Name of the file containing the spiking dataset
ds_tag	'N-CARS', 'N-MNIST', 'MNIST' depending on the dataset that is considered
offset	***** Offset for the chosen samples in the dataset *****

A.2 Supplementary material

A.2.1 DVS working principle

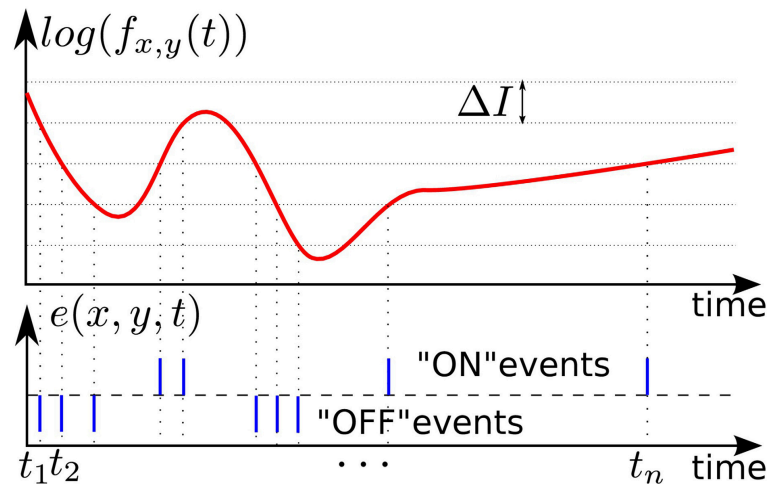


FIGURE A.1 – Illustration of the event-based encoding for visual signals. The evolution of $\log(f_{x,y}(t))$, i.e. the log-illuminance through, time measured at a pixel $[x,y]$ is shown. The resulting two-polarities asynchronous spikes generated by the DVS can be observed when $\log(f_{x,y}(t))$ crosses a predefined threshold ΔI , i.e. the contrast sensitivity. Image from [36].

A.2.2 N-MNIST rescaling

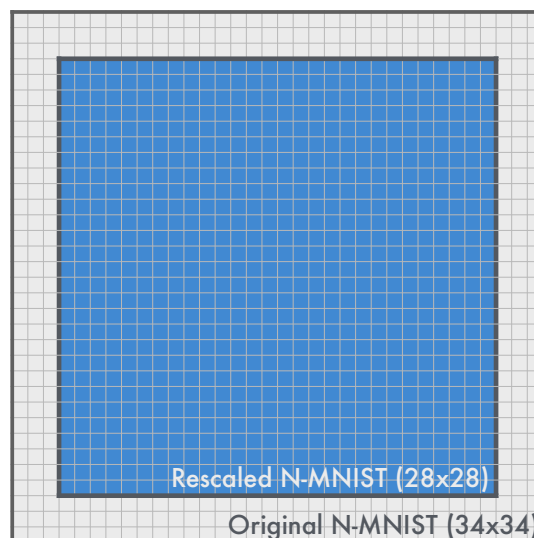


FIGURE A.2 – By visual inspection of the original N-MNIST dataset, events seemed to happen more frequently in the bottom part of the image. However, the cropping was carried out with respect to the center of the image.

A.2.3 Block chain

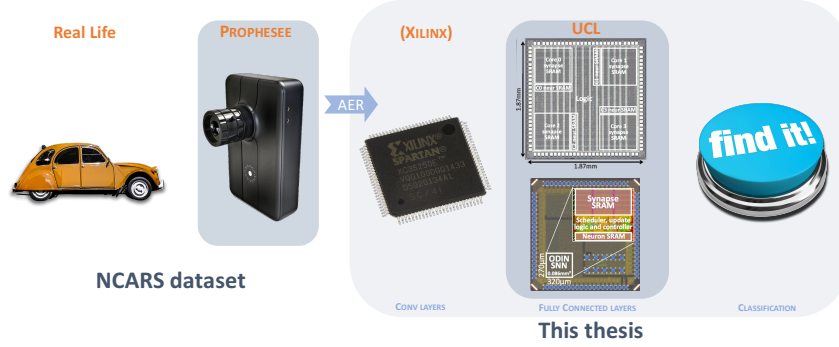


FIGURE A.3 – Schematic block chain proposed in this thesis (given for illustration purposes).

A.2.4 Memory and power considerations

In order to obtain the Figure 5.43, the estimation given in Equation (A.2.4) was used:

$$\begin{cases} N_{synapses} = k_1 \times k_1 \times M \times 2 + 1 + k_3 \times k_3 \times M \times N + 1 \\ N_{neurons} = M \times A_1 \times B_1 + M \times A_2 \times B_2 + N \times A_3 \times B_3 + N \times A_4 \times B_4 \\ \boxed{RAM_{estimation} = N_{synapses} \times R_{synapses} + N_{neurons} \times R_{neurons}} \end{cases}$$

where $N_{synapses}$ is the total number of synaptic weights needed for the Conv part of the network, $R_{synapses}$ is the resolution in *bits* for the synaptic weights, $N_{neurons}$ is the total number of neurons in the Conv part, $R_{neurons}$ is the resolution in *bits* for the neurons, $RAM_{estimation}$ is the estimation in *bits* of total memory needed for the considered network and the other parameters have already been presented.

A.2.5 Error metrics

$$\begin{cases} MSE = \frac{\sum_g (\psi_g - \phi_g)^2}{G} \\ RMSE = \sqrt{MSE} \\ PSNR = 10 \cdot \log_{10}(\frac{d^2}{MSE}) \end{cases}$$

where g is the pixel index, ψ_g is the normalized illumination of the pixel g in the image ψ , ϕ_g is the normalized illumination of the pixel g in the image ϕ , G is the total number of pixels in the image and d is the dynamics of the signal (e.g. 255 for a 8 bits resolution image).

A.2.6 Quantization

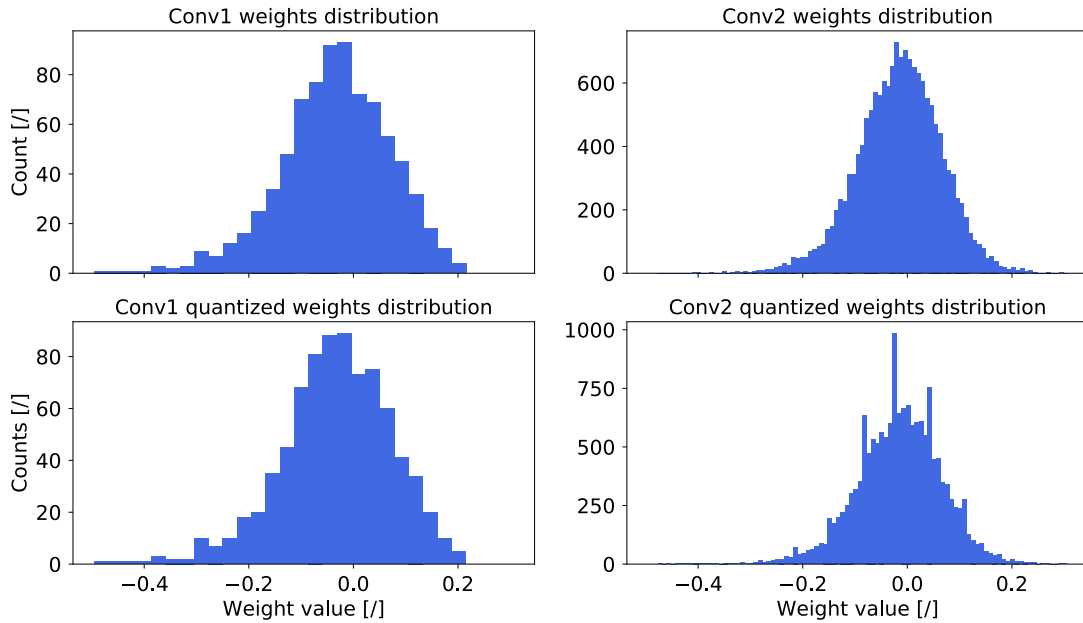


FIGURE A.4 – *Comparison between original and 8-bits quantized weights distributions for the N-CARS network.*

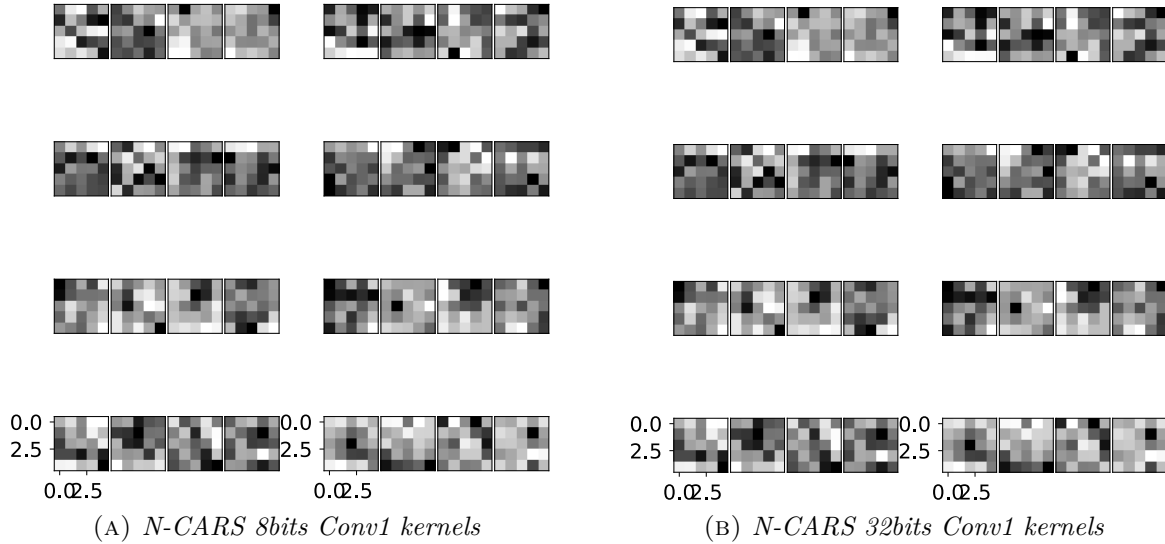


FIGURE A.5 – *Influence of the quantization on the Conv1 weights of the N-CARS network for (A) 8 bits resolution and (B) 32 bits resolution. OFF/ON channels are shown for each kernel.*

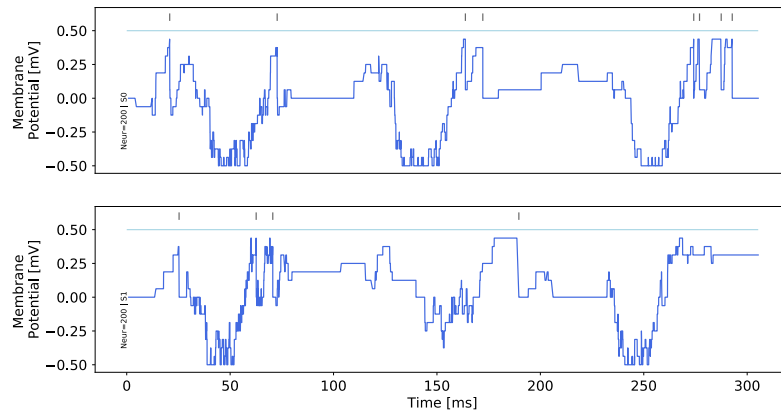
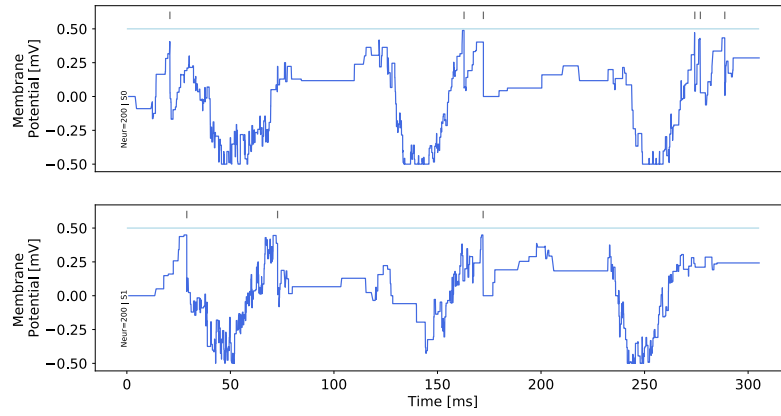
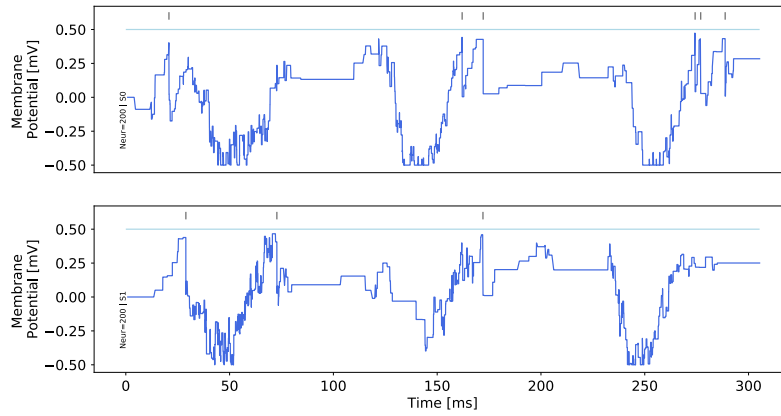
(A) *N*-MNIST 4bits weights resolution(B) *N*-MNIST 8bits weights resolution(C) *N*-MNIST 32bits weights resolution

FIGURE A.6 – ***Influence of the quantization*** on the membrane potential of a *N*-MNIST Conv neuron for (A) 4 bits resolution (B) 8 bits resolution and (C) 32 bits resolution. This is done for the two first samples of the training set.

A.3 Spartan-VII: data sheets



7 Series FPGAs Data Sheet: Overview

DS180 (v2.6) February 27, 2018

Product Specification

General Description

Xilinx® 7 series FPGAs comprise four FPGA families that address the complete range of system requirements, ranging from low cost, small form factor, cost-sensitive, high-volume applications to ultra high-end connectivity bandwidth, logic capacity, and signal processing capability for the most demanding high-performance applications. The 7 series FPGAs include:

- Spartan®-7 Family: Optimized for low cost, lowest power, and high I/O performance. Available in low-cost, very small form-factor packaging for smallest PCB footprint.
- Artix®-7 Family: Optimized for low power applications requiring serial transceivers and high DSP and logic throughput. Provides the lowest total bill of materials cost for high-throughput, cost-sensitive applications.
- Kintex®-7 Family: Optimized for best price-performance with a 2X improvement compared to previous generation, enabling a new class of FPGAs.
- Virtex®-7 Family: Optimized for highest system performance and capacity with a 2X improvement in system performance. Highest capability devices enabled by stacked silicon interconnect (SSI) technology.

Built on a state-of-the-art, high-performance, low-power (HPL), 28 nm, high-k metal gate (HKMG) process technology, 7 series FPGAs enable an unparalleled increase in system performance with 2.9 Tb/s of I/O bandwidth, 2 million logic cell capacity, and 5.3 TMAC/s DSP, while consuming 50% less power than previous generation devices to offer a fully programmable alternative to ASSPs and ASICs.

Summary of 7 Series FPGA Features

- Advanced high-performance FPGA logic based on real 6-input look-up table (LUT) technology configurable as distributed memory.
- 36 Kb dual-port block RAM with built-in FIFO logic for on-chip data buffering.
- High-performance SelectIO™ technology with support for DDR3 interfaces up to 1,866 Mb/s.
- High-speed serial connectivity with built-in multi-gigabit transceivers from 600 Mb/s to max. rates of 6.6 Gb/s up to 28.05 Gb/s, offering a special low-power mode, optimized for chip-to-chip interfaces.
- A user configurable analog interface (XADC), incorporating dual 12-bit 1MSPS analog-to-digital converters with on-chip thermal and supply sensors.
- DSP slices with 25 x 18 multiplier, 48-bit accumulator, and pre-adder for high-performance filtering, including optimized symmetric coefficient filtering.
- Powerful clock management tiles (CMT), combining phase-locked loop (PLL) and mixed-mode clock manager (MMCM) blocks for high precision and low jitter.
- Quickly deploy embedded processing with MicroBlaze™ processor.
- Integrated block for PCI Express® (PCIe), for up to x8 Gen3 Endpoint and Root Port designs.
- Wide variety of configuration options, including support for commodity memories, 256-bit AES encryption with HMAC/SHA-256 authentication, and built-in SEU detection and correction.
- Low-cost, wire-bond, bare-die flip-chip, and high signal integrity flip-chip packaging offering easy migration between family members in the same package. All packages available in Pb-free and selected packages in Pb option.
- Designed for high performance and lowest power with 28 nm, HKMG, HPL process, 1.0V core voltage process technology and 0.9V core voltage option for even lower power.

Table 1: 7 Series Families Comparison

Max. Capability	Spartan-7	Artix-7	Kintex-7	Virtex-7
Logic Cells	102K	215K	478K	1,955K
Block RAM ⁽¹⁾	4.2 Mb	13 Mb	34 Mb	68 Mb
DSP Slices	160	740	1,920	3,600
DSP Performance ⁽²⁾	176 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
MicroBlaze CPU ⁽³⁾	260 DMIPs	303 DMIPs	438 DMIPs	441 DMIPs
Transceivers	—	16	32	96
Transceiver Speed	—	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Serial Bandwidth	—	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	—	x4 Gen2	x8 Gen2	x8 Gen3
Memory Interface	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O Pins	400	500	500	1,200
I/O Voltage	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V
Package Options	Low-Cost, Wire-Bond	Low-Cost, Wire-Bond, Bare-Die Flip-Chip	Bare-Die Flip-Chip and High-Performance Flip-Chip	Highest Performance Flip-Chip

Notes:

- Additional memory available in the form of distributed RAM.
- Peak DSP performance numbers are based on symmetrical filter implementation.
- Peak MicroBlaze CPU performance numbers based on microcontroller preset.

© Copyright 2010–2018 Xilinx, Inc., Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. PCI Express is a trademark of PCI-SIG and used under license. All other trademarks are the property of their respective owners.

Spartan-7 FPGA Feature Summary

Table 2: Spartan-7 FPGA Feature Summary by Device

Device	Logic Cells	CLB		DSP Slices ⁽²⁾	Block RAM Blocks ⁽³⁾			CMTs ⁽⁴⁾	PCIe	GT	XADC Blocks	Total I/O Banks ⁽⁵⁾	Max User I/O
		Slices ⁽¹⁾	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XC7S6	6,000	938	70	10	10	5	180	2	0	0	0	2	100
XC7S15	12,800	2,000	150	20	20	10	360	2	0	0	0	2	100
XC7S25	23,360	3,650	313	80	90	45	1,620	3	0	0	1	3	150
XC7S50	52,160	8,150	600	120	150	75	2,700	5	0	0	1	5	250
XC7S75	76,800	12,000	832	140	180	90	3,240	8	0	0	1	8	400
XC7S100	102,400	16,000	1,100	160	240	120	4,320	8	0	0	1	8	400

Notes:

- Each 7 series FPGA slice contains four LUTs and eight flip-flops; only some slices can use their LUTs as distributed RAM or SRLs.
- Each DSP slice contains a pre-adder, a 25 x 18 multiplier, an adder, and an accumulator.
- Block RAMs are fundamentally 36 Kb in size; each block can also be used as two independent 18 Kb blocks.
- Each CMT contains one MMCM and one PLL.
- Does not include configuration Bank 0.

Table 3: Spartan-7 FPGA Device-Package Combinations and Maximum I/Os

Package	CPGA196	CSGA225	CSGA324	FTGB196	FGGA484	FGGA676
Size (mm)	8 x 8	13 x 13	15 x 15	15 x 15	23 x 23	27 x 27
Ball Pitch (mm)	0.5	0.8	0.8	1.0	1.0	1.0
Device	HR I/O ⁽¹⁾	HR I/O ⁽¹⁾	HR I/O ⁽¹⁾	HR I/O ⁽¹⁾	HR I/O ⁽¹⁾	HR I/O ⁽¹⁾
XC7S6	100	100		100		
XC7S15	100	100		100		
XC7S25		150	150	100		
XC7S50			210	100	250	
XC7S75					338	400
XC7S100					338	400

Notes:

- HR = High-range I/O with support for I/O voltage from 1.2V to 3.3V.

Introduction

The challenge to deliver the highest performance with the lowest power has taken center stage in the evaluation of FPGAs because power conservation impacts both technical constraints and system cost. Product acceptability, reliability, and profitability depend on both power efficiency and performance. Xilinx realized the importance of power reduction and made power management a top priority in 28nm FPGAs. Spartan-7 FPGAs have far superior performance and exhibit a real power consumption advantage to achieve higher performance per watt over previous generation FPGAs. Xilinx focused on total power reduction using these key strategies:

- An innovative and FPGA-optimized 28nm high-performance, low power (HPL) process and a unique power-binning strategy to achieve up to 70% lower static power.
- A superior 7 series architectural innovation to extend the power advantage by reducing dynamic power by 50% and I/O power by 30%.
- Reliable, credible, and more established power estimation and analysis software from the next-generation Vivado® Design Suite.

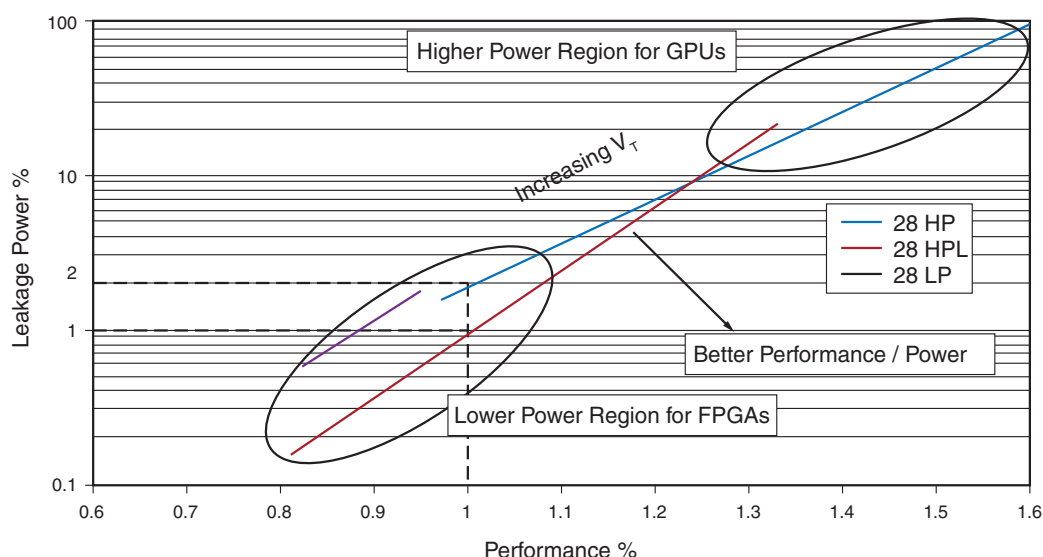
These strategies enabled Xilinx to deliver more than 50% lower total power over previous-generation devices to achieve the highest performance-per-watt goals demanded by cost-sensitive application designs.

The power saving technologies in Spartan-7 FPGAs offer significant benefits in reducing thermal component and power supply costs while increasing overall system reliability. When a transistor consumes more power, the device junction temperature increases exponentially. Therefore, expensive thermal solutions (larger and advanced heat-sinks, effective thermal interface materials, higher airflow from fans, and a larger chassis) can be required—at additional expense. Higher power also requires expensive board solutions, e.g., voltage regulators, complex BOM solutions, and increased PCB area.

Lowest Static Power: Process and Binning Strategies

A key enabler for power reduction is Xilinx's choice to implement its Spartan-7 FPGAs on the mature TSMC 28nm HPL process—tailored for FPGAs and developed through Xilinx and TSMC collaboration. This process employs high-K metal gate technology (HKMG), which dramatically lowers leakage in transistors and enables significant intrinsic performance improvements over 40nm and traditional Polysilicon Oxynitride (PolySiON) technology. The HPL process is scalable and well suited for the high-performance space while enabling significant power reduction. TSMC also offers the 28LP and the 28HP processes at the same node. The 28HP process technology is targeted for high-performance applications where static power is not a concern. It is not suited for FPGAs, where static power is significant compared to ASIC, MPU, and GPU class products. If the HP process is tuned to reduce the leakage power to make it suitable for FPGAs, the performance is significantly reduced. The resulting process is more expensive and more complex than the HPL process—but with no significant benefit in performance. On the other hand, the 28LP process is based on the legacy PolySiON process targeted for low-performance applications. If the LP process is tuned to meet typical FPGA performance requirements, the core voltage needs to be increased to 1.1V. This affects both static and dynamic power significantly, resulting in higher total power than the HPL process.

At low leakage points, the 28HPL process offers a better performance-per-watt metric than 28HP and 28LP processes, and extends to a lower leakage region. The ability to extend into a lower leakage region enabled Xilinx to create the low-power Spartan-7 family without switching to a completely different process technology. See Figure 1.



WP488_01_012017

Figure 1: 28nm HPL Process Advantage

Besides offering a slew of intrinsic advantages in terms of power, the HPL process has ample headroom to enable power binning and voltage scaling techniques that are not available for FPGAs implemented in other processes. Voltage headroom is calculated by $V_{CC} - V_T$, where V_{CC} is the core voltage and V_T is the threshold voltage, enabling Xilinx to offer two different operating modes in Spartan-7 FPGAs:

- **High-Performance Mode:** The core voltage is 1.0V and offers comparable performance to 28HP at lower static power.
- **Low-Power Mode:** The core voltage is 0.95V and offers up to 70% lower static power than 28HP. Spartan-7 FPGAs offer the -1LI speed grade to run at 0.95V with full -1 performance, along with 10% savings on dynamic power and up to 40% savings on static power. See Figure 1.

Leakage power is closely related to threshold voltage (V_T). To obtain the lowest possible leakage, Xilinx began with only the regular V_T (lowest leakage) transistors to design the logic, then moved to smaller V_T (leakier) transistors only as necessary to meet the block's performance target. In this optimal transistor mix strategy, the number of leaky transistors was reduced considerably. On average, optimized blocks have a 25–90% static power reduction without impacting performance.

The data shown in Figure 2 compares static power when implemented on a Xilinx 28HPL device with around 50K logic cells versus a Spartan-6 device with a similar logic cell count. Note that the Spartan-7 7S50 FPGA is 15% larger than the Spartan-6 SLX45T in terms of logic cells. Nevertheless, at max temperature, Spartan-7 FPGA power is 40% lower at nominal voltage and ~70% lower at

V_{LOW} (0.95V), demonstrating a significant static power improvement over previous-generation devices. Figure 2 also shows the competitor's 28LP device and previous-generation device with similar 50K equivalent logic cells. Note that the Spartan-7 7S50 FPGA has 4% more logic cells than competitor's 5CEFA4. However, the Spartan-7 device's static power is 20% lower than the 28LP device at nominal voltage and 55% lower power at V_{LOW} (lower core voltage)—a significant power reduction. Similarly, with respect to the competitor's previous generation devices, Spartan-7 FPGAs offer around 50% lower static power at nominal voltage and 70% lower static power at V_{LOW} . The difference in static power at a fixed junction temperature (T_j) is important not only from the point of view of total power, but also from the perspective of operating temperature (thermal) headroom over competing devices.

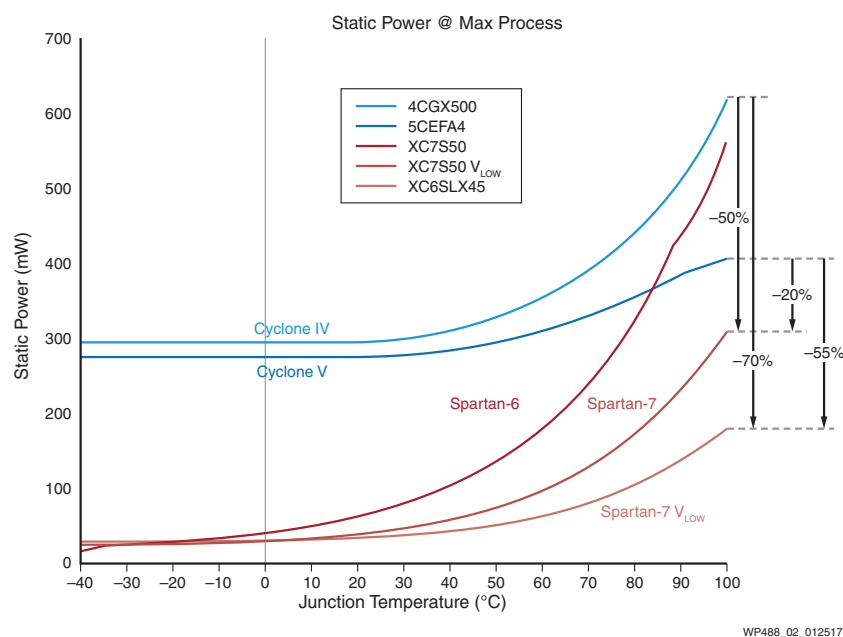


Figure 2: Static Power vs. Junction Temperature Analysis

Power Gating of Unused Blocks

For many device generations, Xilinx has provided users the ability to employ power gating to shut off unused clock managers (PLL/MMCM) and I/Os in their design. In Spartan-7 FPGAs, the power gating feature is extended to block RAMs as well. This is a significant advancement because almost 30% of total device leakage is block RAM leakage. Xilinx improved block RAM leakage by power gating uninstantiated or unused block RAMs, reducing overall device static power.

Table 4: V_{IN} Maximum Allowed AC Voltage Overshoot and Undershoot for HR I/O Banks⁽¹⁾⁽²⁾

AC Voltage Overshoot	% of UI at –40°C to 125°C	AC Voltage Undershoot	% of UI at –40°C to 125°C
$V_{CCO} + 0.55$	100	–0.40	100
		–0.45	61.7
		–0.50	25.8
		–0.55	11.0
$V_{CCO} + 0.60$	46.6	–0.60	4.77
$V_{CCO} + 0.65$	21.2	–0.65	2.10
$V_{CCO} + 0.70$	9.75	–0.70	0.94
$V_{CCO} + 0.75$	4.55	–0.75	0.43
$V_{CCO} + 0.80$	2.15	–0.80	0.20
$V_{CCO} + 0.85$	1.02	–0.85	0.09
$V_{CCO} + 0.90$	0.49	–0.90	0.04
$V_{CCO} + 0.95$	0.24	–0.95	0.02

Notes:

1. A total of 200 mA per bank should not be exceeded.
2. The peak voltage of the overshoot or undershoot, and the duration above $V_{CCO} + 0.20V$ or below $GND - 0.20V$, must not exceed the values in this table.

Table 5: Typical Quiescent Supply Current⁽¹⁾⁽²⁾⁽³⁾

Symbol	Description	Device	Speed Grade						Units
			1.0V					0.95V	
			–2C	–2I	–1C	–1I	–1Q	–1II	
I_{CCINTQ}	Quiescent V_{CCINT} supply current.	XC7S6	36	36	36	36	36	32	mA
		XC7S15	36	36	36	36	36	32	mA
		XC7S25	48	48	48	48	48	43	mA
		XC7S50	95	95	95	95	95	59	mA
		XC7S75	148	148	148	148	148	134	mA
		XC7S100	148	148	148	148	148	134	mA
		XA7S6	N/A	36	N/A	36	36	N/A	mA
		XA7S15	N/A	36	N/A	36	36	N/A	mA
		XA7S25	N/A	48	N/A	48	48	N/A	mA
		XA7S50	N/A	95	N/A	95	95	N/A	mA
		XA7S75	N/A	148	N/A	148	148	N/A	mA
		XA7S100	N/A	148	N/A	148	148	N/A	mA

Spartan-7 FPGAs

I/O Optimization at the Lowest Cost and Highest Performance-per-Watt (1.0V, 0.95V)									
	Part Number	XC7S6	XC7S15	XC7S25	XC7S50	XC7S75	XC7S100		
Logic Resources	Logic Cells	6,000	12,800	23,360	52,160	76,800	102,400		
	Slices	938	2,000	3,650	8,150	12,000	16,000		
	CLB Flip-Flops	7,500	16,000	29,200	65,200	96,000	128,000		
Memory Resources	Max. Distributed RAM (Kb)	70	150	313	600	832	1,100		
	Block RAM/FIFO w/ ECC (36 Kb each)	5	10	45	75	90	120		
	Total Block RAM (Kb)	180	360	1,620	2,700	3,240	4,320		
Clock Resources	Clock Mgmt Tiles (1 MMCM + 1 PLL)	2	2	3	5	8	8		
I/O Resources	Max. Single-Ended I/O Pins	100	100	150	250	400	400		
	Max. Differential I/O Pairs	48	48	72	120	192	192		
	DSP Slices	10	20	80	120	140	160		
Embedded Hard IP Resources	Analog Mixed Signal (AMS) / XADC	0	0	1	1	1	1		
	Configuration AES / HMAC Blocks	0	0	1	1	1	1		
	Commercial Temp (C)	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2		
Speed Grades	Industrial Temp (I)	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L		
	Expanded Temp (Q)	-1	-1	-1	-1	-1	-1		
	Package ⁽¹⁾	Body Area (mm)	Ball Pitch (mm)	Available User I/O: 3.3V SelectIO™ HR I/O					
	CPGA196	8x8	0.5	100	100				
	CSGA225	13x13	0.8	100	100	150			
	CSGA324	15x15	0.8			150	210		
	FTGB196	15x15	1.0	100	100	100	100		
	FGGA484	23x23	1.0			250	338		
	FGGA676	27x27	1.0				400		

1. Packages with the same last letter and number sequence, e.g., A484, are footprint compatible with all other Spartan-7 devices with the same sequence. The footprint compatible devices within this family are outlined.

List of Figures

2.1	Identification of redundant and interesting data in an image obtained with a conventional camera (frame-based sensing). The sky, trees and grass are unnecessary oversampled whereas the swing of the club and the movement of the player/ball are undersampled. Image adapted from [40].	6
2.2	Fully-connected network. Illustration of a three layers FC network with I input neurons, H hidden neurons and C output neurons.	8
2.3	Processing unit (artificial neuron). Similarities between an artificial and a biological neuron. Image inspired by [37, 57, 58].	8
2.4	Convolution network. Example of a basic two layers convolution-pooling network followed by two FC layers.	9
2.5	Illustration of the convolution operation	9
2.6	von Neumann architecture scheme. The red disc highlights where the VNB occurs. Image inspired from [65].	11
2.7	Comparison of power and space efficiency between digital computers and biological/silicon neurons. Image adapted from [9].	12
2.8	General working principle of a DVS camera. (A) Simplified pixel schematic (B) ON/OFF spikes generation principle. Image adapted from [6, 27].	13
2.9	Data obtained from a DVS sensor. Different visualization techniques are possible: histogram of events in 2D-space over a time window which forms a contrast image based on ON/OFF events, or a gray scale reconstruction showing the relative event time, or even a space-time representation to view the spatio-temporal structure. Image adapted from [6, 27].	14
2.10	Impact of the contrast sensitivity of a DVS sensor on the quality of a reconstructed scene. The accumulation is carried out by accumulating events over a period of 30ms. Contrast sensitivity is set to 1.5% in (A) and 10% in (B). Image adapted from [19].	14

2.11	Schematic comparison of sampling rate performance between frame-based and event-based imagers. Image adapted from [78].	15
2.12	Schematic representation of the AER communication protocol. The numbers in the address encoder/decoder are the neurons indexes. Image inspired by [8, 27].	16
2.13	ODIN. (A) Chip microphotograph with zoom on floorplan view. (B) Specifications and measurements. Image adapted from [7].	17
2.14	MorphIC specifications. (A) Chip microphotograph showing the floorplan view and synapse SRAM macros of each core. (B) Specifications and measurements. Image adapted from [20].	17
2.15	Frame-based to event-based dataset conversion principle. Illustration for the first method.	19
2.16	(A) N-MNIST input spikes raster plot , for the first sample of the training set (label = 5). (B) N-MNIST ON/OFF spikes , for the first sample of the training set (label = 5) and pixel index = 290, i.e. $[x, y] = [10, 10]$	21
2.17	N-MNIST frame-based reconstruction based on accumulation of input spikes and ON-OFF reconstruction. This is done for the first sample of the training set (label = 5) with an accumulation window of (A) 300ms and (B) 50ms.	22
2.18	N-MNIST spikes per sample distribution , for the whole training set. Mean = 4117 spikes/sample.	22
2.19	N-MNIST time distribution of spikes , for 6000 randomly chosen samples of the training set.	22
2.20	(A) Samples from the original N-CARS dataset , reconstructed from accumulating spikes over a time interval. Image adapted from [31]. (B) Space-time representation : events received from the neuromorphic sensor are plotted through time. Image adapted from [31].	23
2.21	(A) N-CARS input spikes raster plot , for the 54 th sample of the training set (label = 1). (B) N-CARS ON/OFF spikes , for the 54 th sample of the training set (label = 1) and pixel index = 470, i.e. $[x, y] = [20, 9]$	24
2.22	N-CARS spikes per sample distribution , for the whole training set (UCL version). Mean = 6139 spikes/sample.	24
2.23	N-CARS time distribution of spikes , for the whole training set (UCL version). The distribution is uniform.	24

2.24	N-CARS frame-based reconstruction based on accumulation of input spikes and ON-OFF reconstruction. This is done for the 54 th sample of the training set (label = 1 = car) with an accumulation window of (A) 100ms and (B) 30ms.	25
3.1	(A) Action potential measurement and synapse illustration . On the left part, an AP recorded <i>in vivo</i> from a cultured rat neocortical pyramidal cell. On the right part, a schematic representation of a synapse. Adapted from [95]. (B) Biological neuron illustration and firing measurements . The dendritic tree is a branching topology specific to neurons, allowing to receive inputs from multiples other neurons through synaptic connections as shown in (A). Adapted from [95].	28
3.2	Rate-based and population coding , illustration.	30
3.3	LIF single-cell electronic schematics . Image adapted from [86, 112].	32
3.4	Asymmetric STDP learning window , characterized in hippocampal cultures. Image adapted from [125].	36
3.5	Digital SDSP rule , proposed by <i>Brader et al.</i> [132]. Image adapted from [7].	37
3.6	Methodology to convert a CNN to a SNN	39
3.7	(A) Illustration to explain the gray-level reconstruction principle of images by accumulation of spikes (only ON spikes are considered here). (B) Result of a reconstruction gray-level image using the first sample of the N-MNIST training set with an accumulation window $dt = 50ms$ (only ON spikes are considered).	42
3.8	Example of gray-level (contrast) reconstruction for the 19 th sample of the N-CARS training set, label = 1 (car). Accumulation window is 100ms, i.e. the whole sample duration. The ON-OFF corresponding reconstruction is shown on the right. The spike trains for four consecutive pixels (e.g. $[x, y] = [29 \rightarrow 32, 18]$) are represented on the left.	42

4.1	General SNN architecture used throughout this work. A and B are set by the shape of the dataset. M and N are the number of channels for the convolution layers whereas H is the number of neurons in the FC hidden layer. C is set by the number of output classes and H must be set to fit on MORPHIC accordingly to Equation (4.1). A_i and B_i depend on the kernel sizes k_i chosen for Conv and subsampling layers. All connections are not shown.	47
4.2	Three-step strategy overview.	49
4.3	(A) SSP-MNIST frame-based reconstruction by accumulation of input spikes , for the first sample of the training set (label = 5). Accumulation time is 100 [ms]. (B) First sample of the original MNIST training set , (label = 5).	50
4.4	(A) SSP-MNIST input spikes raster plot , for the first sample of the training set, i.e. label = 5. (B) SSP-MNIST ON spikes , for the first sample of the training set, i.e. label = 5 and pixel index = 269, i.e. $[x, y] = [17, 9]$. There are 154 ON spikes as the value of that pixel in the original dataset is 154.	51
4.5	Comparison between N-MNIST and SSP-MNIST spikes per sample distribution , for the whole test set. $\text{Mean}_{N-MNIST} = 4158$ spikes/sample, $\text{Mean}_{SSP-MNIST} = 26492$ spikes/sample.	51
4.6	Structure of the spiking simulator. All connections are not shown. . .	53
5.1	Network structure used for SSP-MNIST. All connections are not shown.	56
5.2	SSP-MNIST learning curves obtained on KERAS. Test accuracy resulting of this training was 98.31%.	57
5.3	Error in Conv layers. In each sub-figure, the first set of activation maps is the output of the Conv layer on KERAS. The second set is the activation maps of the corresponding Conv layer in the spiking simulator and the last set is the difference between the two first sets, activation map by activation map. RMSE is used as the error metric. This was done here for the first sample of the MNIST training set (label = 5).	58

5.4	Error in hidden and output layers. Comparison was made between activation maps obtained on KERAS and activation maps in the spiking simulator. Activation maps were normalized such that they can be compared and RMSE was used as the error metric. Sub-figures (A) and (B) illustrate a good mapping $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$ as the error is low (RMSE = 0.03 for both layers) whereas (C) and (D) give an example of a bad mapping $\Theta_m = [1, 1, 1, 1, 1][1, 2, 3]$ as activation maps are very different and error is high (RMSE = 0.2 and 0.74 respectively). This was done here for the first sample of the MNIST training set (label = 5).	59
5.5	Three combinations of neuron models investigated with the spiking simulator.	60
5.6	Error in output layer. Comparison between the activation map obtained on KERAS and the one obtained in the spiking simulator for the first sample of the training set (label = 5). RMSE is used as the error metric. $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$	60
5.7	IF implementation. (A) SSP-MNIST accuracy surface with the basic IF neuron model , over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow star corresponds to the highest accuracy on the surface (96.82%) which was obtained for $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$. (B) SSP-MNIST confusion matrix , for the best thresholds vector $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$ obtained in (A), inference was carried out on the full test set (10k samples). The resulting accuracy was 96.03%.	61
5.8	IF implementation. (A) Raster plot for the two FC layers of the network, considering the first sample of the training set (label = 5). The sub-plot on the top is for the hidden layer while the one on the bottom is for the output layer. $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$. (B) Inference curves , for output neurons of the first sample of the training set (label = 5). $\Theta_m = [1, 2, 1, 2, 1][1, 10, 1]$	62
5.9	AIF implementation. (A) SSP-MNIST accuracy surface , over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow star corresponds to the highest accuracy on the surface (98.83%) which was obtained for $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$. (B) SSP-MNIST confusion matrix , for the best thresholds vector $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$ obtained in (A), inference was carried out on the full test set (10k samples). The resulting accuracy was 98.17%.	62

- 5.10 AIF implementation. (A) **Evolution of the membrane potential $V(t)$** , for the fifth neuron of the output layer. The sub-plot on the top is for the first sample of the training set (label = 5) while the one on the bottom is for the second sample (label = 2). $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$. (B) **Inference curves**, for output neurons of the first sample of the training set (label = 5). $\Theta_m = [1, 1, 1, 1, 1][1, 10, 7]$ 63
- 5.11 Hybrid implementation. (A) **SSP-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow star corresponds to the highest accuracy on the surface (98.87%) which was obtained for $\Theta_m = [1, 1, 1, 1, 1][1, 16, 1]$. (B) **SSP-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 1, 1, 1, 1][1, 16, 1]$ obtained in (A), inference was carried out on the full test set (10k samples). The resulting accuracy was 98.19%. 64
- 5.12 **MNIST reported accuracies** for spiking networks. BEST ANN = best accuracy reported for ANN network, FC = Fully Connected, BP = Back-propagation, OFF = Offline training (conversion or hardware), CONV = Convolution, TEMPO = Tempotron, RBM = Restricted Boltzmann Machines, DBN = Deep Belief Network. Data compiled from [71] and [4, 47, 92, 123, 150]. The classification accuracy for the three combinations (IF, AIF and HYBRID implementation) proposed in this section are given. The continuous-valued CNN classification accuracy is also shown in order to compare the classification accuracy before and after mapping. 66
- 5.13 **Network structure** used for N-MNIST. All connections are not shown. . . 67
- 5.14 **N-MNIST frame-based reconstructed dataset**. The samples shown here are the five first samples of the N-MNIST test set (one per column). Each row corresponds to a different accumulation time (ACCU) as indicated on the left axis. Each image was normalized as explained in Section 3.4.2. 68
- 5.15 **N-MNIST learning curves** obtained on KERAS. Test accuracy resulting of this training was 96.92%. 69

- 5.16 **Error in Conv and FC layers.** In each sub-figure, the first set of activation maps is the output of the layer on KERAS. The second set is the activation maps of the corresponding layer in the spiking simulator and the last set is the difference between the two first sets, activation map by activation map. RMSE is used as the error metric. This was done here for the first sample of the MNIST training set (label = 5). $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 10, 10]$. 71
- 5.17 **Error in FC layers due to bad threshold balancing.** RMSE is used as the error metric. This was done here for the first sample of the MNIST training set (label = 5). $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 2, 4]$ 72
- 5.18 **Comparison between inference curves for different threshold balancing** for the **first** sample of the training set (label = 5). (A) Bad threshold balancing, $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 2, 4]$. (B) Good threshold balancing, $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 4]$ 72
- 5.19 **Comparison between inference curves for different threshold balancing** for the **fourth** sample of the training set (label = 9). (A) Bad threshold balancing, $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 2, 4]$. (B) Good threshold balancing, $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 4]$ 72
- 5.20 Inference using the MNIST weights and the hybrid implementation (trade-off). (A) **N-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 600 samples from the training set for each combination. The yellow star corresponds to the highest accuracy 55.5% on the surface which was obtained for $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 18, 13]$. (B) **N-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 18, 13]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 51.2%. 73
- 5.21 IF implementation. (A) **N-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow stars correspond to the highest accuracy 81.20% on the surface which was obtained for $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 6]$ and $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 7]$. (B) **N-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 6]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 81.25%. 74

- 5.22 AIF implementation. (A) **N-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 6000 samples from the training set for each combination. The yellow stars correspond to the highest accuracy 97.15% on the surface which was obtained for $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 1]$ and $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 7]$. (B) **N-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 8, 1]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 96.92%. 75
- 5.23 AIF implementation. **Classification accuracy** on the full test set for similar Θ_m . Brief analysis of the robustness to parameter variation. 75
- 5.24 Hybrid implementation. (A) **N-MNIST accuracy surface**, over a grid of different combinations of hidden-output thresholds. Inference was carried out on 2000 samples from the training set for each combination. The yellow star corresponds to the highest accuracy 97.20% on the surface which was obtained for $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 1]$. (B) **N-MNIST confusion matrix**, for the best thresholds vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 18, 1]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 96.83%. 76
- 5.25 **Accuracy-latency trade-off highlighted by inference curves for the hybrid implementation.** It was carried out for three different Θ_m in order to highlight the latency introduced due to higher thresholds. $\Theta_{m-1} = [1, 1, 1, 1, 1][1, 6, 1]$, $\Theta_{m-2} = [1, 0.5, 1, 0.5, 1][1, 6, 1]$, $\Theta_{m-3} = [1, 0.5, 1, 0.5, 1][1, 28, 7]$. The histogram in the background of the subplot (B) can also be observed in Figure 2.19 and allows to observe the three saccades and the corresponding inference. The histogram is given without scale, just to outline the distribution of spikes through time in order to make the link between the spikes distribution and the inference curves. 77
- 5.26 **Influence of input thresholding on the classification accuracy.** This was done with the hybrid implementation on the full test set (10k samples). The gray histogram in the background represents the spikes per sample distribution obtained in Figure 2.18. Simulations are run with $N_{cores} = 25$. Mean = 4117 spikes/sample. 78
- 5.27 **Space-time representation** of N-MNIST. 12th training sample, label = 3. 79

5.28	N-MNIST reported accuracies for spiking networks. BEST ANN = best accuracy reported for ANN network, FC = Fully Connected, BP = Backpropagation, OFF = Offline training (conversion or hardware), CONV = Convolution. Data compiled in [71] and [31, 87, 92, 120]. The classification accuracy of the IF, AIF and HYBRID implementations are given alongside the one of the continuous-valued CNN.	80
5.29	Network structure used for N-CARS. All connections are not shown. . .	81
5.30	N-CARS frame-based reconstructed dataset. The samples shown here are the five first samples of the N-CARS test set (one per column). Each row corresponds to a different accumulation time (ACCU) as indicated.	82
5.31	N-CARS learning curves: presence of overfitting.	83
5.32	N-CARS learning curves, without overfitting. Dropout (0.3) was applied. Test accuracy resulting of this training was 87.2%.	83
5.33	Error in Conv and FC layers. In each sub-figure, the first set of activation maps is the output of the layer on KERAS. The second set is the activation maps of the corresponding layer in the spiking simulator and the last set is the difference between the two first sets, activation map by activation map. RMSE is used as the error metric. This was done here for the third sample of the N-CARS training set (label = 1). Only the hybrid implementation was considered with $\Theta_{m-1} = [1, 0.25, 1, 0.5, 1][1, 8, 1]$ and $\Theta_{m-2} = [1, 0.25, 1, 0.5, 1][1, 10, 5]$	84
5.34	IF implementation. (A) N-CARS accuracy surface , over a grid of different combinations of hidden-output thresholds. Inference was carried out on 300 samples from the training set for each combination. The yellow star corresponds to the highest accuracy 77% on the surface which was obtained for $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 20, 4]$. (B) N-CARS confusion matrix , for the best thresholds vector $\Theta_m = [1, 0.5, 1, 0.5, 1][1, 20, 4]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 68.71%.	85

5.35	AIF implementation. (A) N-CARS accuracy surface , over a grid of different combinations of hidden-output thresholds. Inference was carried out on 300 samples from the training set for each combination. The yellow star corresponds to the highest accuracy 92% on the surface which was obtained for $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 5, 19]$. (B) N-CARS confusion matrix , for the best thresholds vector $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 5, 19]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 82.25%.	86
5.36	Hybrid implementation. (A) N-CARS accuracy surface , over a grid of different combinations of hidden-output thresholds. Inference was carried out on 300 samples from the training set for each combination. The yellow stars correspond to the highest accuracies 90.33% on the surface which was obtained for $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 8, 1]$ and $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 8, 4]$. (B) N-CARS confusion matrix , for the best thresholds vector $\Theta_m = [1, 0.25, 1, 0.5, 1][1, 8, 1]$ obtained in (A), inference was carried out on the full test set. The resulting accuracy was 81.97%.	87
5.37	Influence of input thresholding on the classification accuracy. This was done with the hybrid implementation on the full test set (14650k samples). The gray histogram in the background represents the spikes per sample distribution obtained in Figure 2.22. Simulations are run with $N_{cores} = 25$. Mean = 6139 spikes/sample.	87
5.38	N-CARS reported accuracies for spiking networks. BEST ANN = best accuracy reported for ANN network, FC = Fully Connected, BP = Back-propagation, GABOR-SNN = Gabor [151, 152, 153, 154] filters for Conv layers followed by SNN training with BP. Data compiled in [30, 31, 39]. The classification accuracy for the three combinations (IF, AIF and HYBRID implementation) proposed in this section are given. The continuous-valued CNN classification accuracy is also shown in order to compare the classification accuracy before and after mapping.	89
5.39	Hardware resources needed by neural networks proposed in this work for (A) N-MNIST and (B) N-CARS.	90
5.40	Influence of the quantization of the Conv weights on the classification accuracy for (A) N-MNIST and (B) N-CARS.	91
5.41	Comparison between original and 4-bits quantized weights distributions for the N-MNIST network.	92

5.42	Influence of the quantization on the Conv1 weights of the N-MNIST network for (A) 4 bits resolution and (B) 32 bits resolution. OFF/ON channels are shown for each kernel.	92
5.43	Estimation of the evolution of the memory needed with respect to the resolution of the neurons and synapses in the proposed networks. The analysis was carried out for N-MNIST and N-CARS networks. The gray dashed line represents the memory (RAM) available on different models of the SPARTAN-VII device. These results are given as a coarse estimation of the necessary storage (lower bound): practical limitations related to FPGA programming should be considered to have a better estimation. More details about the estimation are in given in Appendix A.2.4.	94
5.44	Evolution of the static power for different FPGA devices. Image from SPARTAN-VII data sheets, Appendix A.3.	94
5.45	Summary of the classification accuracy performance for each step of the proposed strategy. Key improvements are highlighted. Symbols details are given in Figure 3.6.	95
A.1	Illustration of the event-based encoding for visual signals. The evolution of $\log(f_{x,y}(t))$, i.e. the log-illuminance through, time measured at a pixel [x,y] is shown. The resulting two-polarities asynchronous spikes generated by the DVS can be observed when $\log(f_{x,y}(t))$ crosses a predefined threshold ΔI , i.e. the contrast sensitivity. Image from [36].	102
A.2	By visual inspection of the original N-MNIST dataset, events seemed to happen more frequently in the bottom part of the image. However, the cropping was carried out with respect to the center of the image.	102
A.3	Schematic block chain proposed in this thesis (given for illustration purposes).	103
A.4	Comparison between original and 8-bits quantized weights distributions for the N-CARS network.	104
A.5	Influence of the quantization on the Conv1 weights of the N-CARS network for (A) 8 bits resolution and (B) 32 bits resolution. OFF/ON channels are shown for each kernel.	104
A.6	Influence of the quantization on the membrane potential of a N-MNIST Conv neuron for (A) 4 bits resolution (B) 8 bits resolution and (C) 32 bits resolution. This is done for the two first samples of the training set.	105

List of Tables

2.1	Chip characteristics and measured performances compared with the state of the art of spiking neuromorphic circuits. The original version is available in [7]	18
5.1	KERAS parameters for MNIST dataset learning.	57
5.2	KERAS parameters for N-CARS reconstructed dataset learning	83

Bibliography

- [1] Forrest Iandola, “Exploring the design space of deep convolutional neural networks at large scale”, in: *arXiv preprint arXiv:1612.06519* (2016).
- [2] Md Zahangir Alom et al., “A state-of-the-art survey on deep learning theory and architectures”, in: *Electronics* **8.3** (2019), p. 292.
- [3] Evangelos Stamatias et al., “An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data”, in: *Frontiers in neuroscience* **11** (2017), p. 350.
- [4] Peter U Diehl et al., “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing”, in: *2015 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015, pp. 1–8.
- [5] Lucas Pastur-Romay et al., “Deep artificial neural networks and neuromorphic chips for big data analysis: pharmaceutical and bioinformatics applications”, in: *International journal of molecular sciences* **17.8** (2016), p. 1313.
- [6] Shih-Chii Liu and Tobi Delbruck, “Neuromorphic sensory systems”, in: *Current opinion in neurobiology* **20.3** (2010), pp. 288–295.
- [7] Charlotte Frenkel et al., “A 0.086-mm² 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS”, in: *IEEE transactions on biomedical circuits and systems* **13.1** (2018), pp. 145–158.
- [8] Gregory Kevin Cohen, “Event-based feature detection, recognition and classification”, PhD thesis, Paris 6, 2016.
- [9] Chi-Sang Poon and Kuan Zhou, “Neuromorphic silicon neurons and large-scale neural networks: challenges and opportunities”, in: *Frontiers in neuroscience* **5** (2011), p. 108.
- [10] Emre O Neftci et al., “Event-driven random back-propagation: Enabling neuromorphic deep learning machines”, in: *Frontiers in neuroscience* **11** (2017), p. 324.
- [11] Bodo Rueckauer et al., “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification”, in: *Frontiers in neuroscience* **11** (2017), p. 682.
- [12] Kwabena Boahen, “Neuromorphic microchips”, in: *Scientific American* **292.5** (2005), pp. 56–63.

-
- [13] Kwabena A Boahen, “Point-to-point connectivity between neuromorphic chips using address events”, in: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* **47.5** (2000), pp. 416–434.
 - [14] Robert R Schaller, “Moore’s law: past, present and future”, in: *IEEE spectrum* **34.6** (1997), pp. 52–59.
 - [15] Mark Horowitz, “1.1 computing’s energy problem (and what we can do about it)”, in: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, IEEE, 2014, pp. 10–14.
 - [16] John Backus, *Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs*, ACM, 2007.
 - [17] Giacomo Indiveri, Elisabetta Chicca, and Rodney J Douglas, “Artificial cognitive systems: From VLSI networks of spiking neurons to neuromorphic cognition”, in: *Cognitive Computation* **1.2** (2009), pp. 119–127.
 - [18] Giacomo Indiveri and Timothy K Horiuchi, “Frontiers in neuromorphic engineering”, in: *Frontiers in neuroscience* **5** (2011), p. 118.
 - [19] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco, “A 128x128 1.5% Contrast Sensitivity 0.9% FPN 3 μ s Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers”, in: *IEEE Journal of Solid-State Circuits* **48.3** (2013), pp. 827–838.
 - [20] Charlotte Frenkel, Jean-Didier Legat, and David Bol, “A 65-nm 738k-Synapse/mm² Quad-Core Binary-Weight Digital Neuromorphic Processor with Stochastic Spike-Driven Online Learning”, in: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2019, pp. 1–5.
 - [21] Eustace Painkras et al., “SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation”, in: *IEEE Journal of Solid-State Circuits* **48.8** (2013), pp. 1943–1953.
 - [22] Mike Davies et al., “Loihi: A neuromorphic manycore processor with on-chip learning”, in: *IEEE Micro* **38.1** (2018), pp. 82–99.
 - [23] Filipp Akopyan et al., “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip”, in: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **34.10** (2015), pp. 1537–1557.
 - [24] Ben Varkey Benjamin et al., “Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations”, in: *Proceedings of the IEEE* **102.5** (2014), pp. 699–716.

-
- [25] Misha A Mahowald and Carver Mead, “The Silicon Retina”, in: *SCIENTIFIC AMERICAN May* (1991), p. 7.
 - [26] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt, “A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS”, in: *IEEE Journal of Solid-State Circuits* **46.1** (2010), pp. 259–275.
 - [27] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck, “A 128x128 120 dB 15 μ s Latency Asynchronous Temporal Contrast Vision Sensor”, in: *IEEE journal of solid-state circuits* **43.2** (2008), pp. 566–576.
 - [28] Vincent Chan, Shih-Chii Liu, and Andr van Schaik, “AER EAR: A matched silicon cochlea pair with address event representation interface”, in: *IEEE Transactions on Circuits and Systems I: Regular Papers* **54.1** (2007), pp. 48–59.
 - [29] Cheston Tan, Stephane Lallee, and Garrick Orchard, “Benchmarking neuromorphic vision: lessons learnt from computer vision”, in: *Frontiers in neuroscience* **9** (2015), p. 374.
 - [30] Xavier Lagorce et al., “Hots: a hierarchy of event-based time-surfaces for pattern recognition”, in: *IEEE transactions on pattern analysis and machine intelligence* **39.7** (2016), pp. 1346–1359.
 - [31] Amos Sironi et al., “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.
 - [32] Bo Zhao et al., “Feedforward categorization on AER motion events using cortex-like features in a spiking neural network”, in: *IEEE transactions on neural networks and learning systems* **26.9** (2014), pp. 1963–1978.
 - [33] Himanshu Akolkar et al., “What can neuromorphic event-driven precise timing add to spike-based pattern recognition?”, in: *Neural computation* **27.3** (2015), pp. 561–593.
 - [34] RED, *RED Digital Cinema*, URL: <https://www.red.com/dsmc2> (visited on 07/23/2019).
 - [35] Garrick Orchard et al., “HFirst: a temporal approach to object recognition”, in: *IEEE transactions on pattern analysis and machine intelligence* **37.10** (2015), pp. 2028–2040.
 - [36] Xavier Lagorce et al., “Spatiotemporal features for asynchronous event-based data”, in: *Frontiers in neuroscience* **9** (2015), p. 46.

-
- [37] José Antonio Pérez-Carrasco 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”, in: *IEEE transactions on pattern analysis and machine intelligence* **35.11** (2013), pp. 2706–2719.
- [38] Vasudev Bhaskaran and Konstantinos Konstantinides, *Image and video compression standards: algorithms and architectures*, vol. 408, Springer Science & Business Media, 1997.
- [39] Henri Rebecq et al., “Events-to-video: Bringing modern computer vision to event cameras”, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3857–3866.
- [40] Prophesee, *Event-based-vision*, URL: <https://www.prophesee.ai/2018/06/25/event-based-vision-2/> (visited on 07/24/2019).
- [41] Peter O’Connor et al., “Real-time classification and sensor fusion with a spiking deep belief network”, in: *Frontiers in neuroscience* **7** (2013), p. 178.
- [42] Ian H Witten et al., *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann, 2016.
- [43] Radwa Elshawy, Mohamed Maher, and Sherif Sakr, “Automated Machine Learning: State-of-The-Art and Open Challenges”, in: *arXiv preprint arXiv:1906.02287* (2019).
- [44] Bernard Marr, *Forbes: How Much Data Do We Create Every Day?*, URL: <https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#75e6d29560ba> (visited on 07/07/2019).
- [45] Bernard Espinasse and Patrice Bellot, “Introduction au Big Data Opportunités, stockage et analyse des mégadonnées”, in: **33** (2017), URL: <http://www.lsis.org/espinasseb/Supports/BD/Article-BigData-TI-2016.pdf>.
- [46] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, *Foundations of Machine Learning*, MIT Press, 2018.
- [47] Kamran Kowsari et al., “Rmdl: Random multimodel deep learning for classification”, in: *Proceedings of the 2nd International Conference on Information System and Data Mining*, ACM, 2018, pp. 19–28.
- [48] Yoshua Bengio et al., “Greedy layer-wise training of deep networks”, in: *Advances in neural information processing systems*, 2007, pp. 153–160.

-
- [49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, “Imagenet classification with deep convolutional neural networks”, in: *Advances in neural information processing systems*, 2012, pp. 1097–1105.
 - [50] Qi-jun Zhang and Kuldip C Gupta, *Neural networks for RF and microwave design (Book+ Neuromodeler Disk)*, Artech House, Inc., 2000.
 - [51] Abhronil Sengupta et al., “Going deeper in spiking neural networks: VGG and residual architectures”, in: *Frontiers in neuroscience* **13** (2019).
 - [52] Amirhossein Tavanaei et al., “Deep learning in spiking neural networks”, in: *Neural Networks* (2018).
 - [53] Arash Ardakani, Carlo Condo, and Warren J Gross, “Sparsely-connected neural networks: towards efficient vlsi implementation of deep neural networks”, in: *arXiv preprint arXiv:1611.01427* (2016).
 - [54] Ian T Jolliffe and Jorge Cadima, “Principal component analysis: a review and recent developments”, in: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **374.2065** (2016), p. 20150202.
 - [55] J. Ross Quinlan, “Induction of decision trees”, in: *Machine learning* **1.1** (1986), pp. 81–106.
 - [56] Chih-Chung Chang and Chih-Jen Lin, “LIBSVM: A library for support vector machines”, in: *ACM transactions on intelligent systems and technology (TIST)* **2.3** (2011), p. 27.
 - [57] Gregory K Cohen et al., “Skimming digits: neuromorphic classification of spike-encoded images”, in: *Frontiers in neuroscience* **10** (2016), p. 184.
 - [58] Jonathan C Tapson et al., “Synthesis of neural networks for spatio-temporal spike pattern recognition and processing”, in: *Frontiers in neuroscience* **7** (2013), p. 153.
 - [59] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, and Timothée Masquelier, “Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition”, in: *Neurocomputing* **205** (2016), pp. 382–392.
 - [60] Yann LeCun et al., “Backpropagation applied to handwritten zip code recognition”, in: *Neural computation* **1.4** (1989), pp. 541–551.
 - [61] Yann LeCun et al., “Gradient-based learning applied to document recognition”, in: *Proceedings of the IEEE* **86.11** (1998), pp. 2278–2324.
 - [62] Foram S Panchal and Mahesh Panchal, “Review on methods of selecting number of hidden nodes in artificial neural network”, in: *International Journal of Computer Science and Mobile Computing* **3.11** (2014), pp. 455–464.

-
- [63] Nitish Srivastava et al., “Dropout: a simple way to prevent neural networks from overfitting”, in: *The journal of machine learning research* **15.1** (2014), pp. 1929–1958.
- [64] Sergey Ioffe and Christian Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, in: *arXiv preprint arXiv:1502.03167* (2015).
- [65] Wikipedia, *Von Neumann Bottleneck*, 2019, URL: https://en.wikipedia.org/wiki/Von_Neumann_architecture (visited on 08/12/2019).
- [66] Charlotte Frenkel et al., “A fully-synthesized 20-gate digital spike-based synapse with embedded online learning”, in: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2017, pp. 1–4.
- [67] Giacomo Indiveri, Federico Corradi, and Ning Qiao, “Neuromorphic architectures for spiking deep neural networks”, in: *2015 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2015, pp. 4–2.
- [68] Giacomo Indiveri and Shih-Chii Liu, “Memory and information processing in neuromorphic systems”, in: *Proceedings of the IEEE* **103.8** (2015), pp. 1379–1397.
- [69] Runchun Wang and André van Schaik, “Breaking Liebig’s law: an advanced multipurpose neuromorphic engine”, in: *Frontiers in neuroscience* **12** (2018).
- [70] Yongqiang Cao, Yang Chen, and Deepak Khosla, “Spiking deep convolutional neural networks for energy-efficient object recognition”, in: *International Journal of Computer Vision* **113.1** (2015), pp. 54–66.
- [71] Amirhossein Tavanaei et al., “Deep learning in spiking neural networks”, in: *Neural Networks* (2018).
- [72] James V Stone, *Principles of neural information theory: Computational neuroscience and metabolic efficiency*, Sebtel Press, 2018.
- [73] Horace B Barlow et al., “Possible principles underlying the transformation of sensory messages”, in: *Sensory communication* **1** (1961), pp. 217–234.
- [74] John L Hennessy and David A Patterson, *Computer architecture: a quantitative approach*, Elsevier, 2011.
- [75] Bharath Ramesh et al., “DART: distribution aware retinal transform for event-based cameras”, in: *IEEE transactions on pattern analysis and machine intelligence* (2019).
- [76] Hongmin Li et al., “Cifar10-dvs: An event-stream dataset for object classification”, in: *Frontiers in neuroscience* **11** (2017), p. 309.

-
- [77] AilabRPG-University of Zurich and ETH Zurich-SpaceX-NeuroVision-Garrick Orchard, *Compilation of neuromorphic sensors videos (YouTube)*, <https://www.youtube.com/watch?v=DFMvQ7r0UA8>, https://www.youtube.com/watch?v=0q6ap_OSBAk, <https://www.youtube.com/watch?v=bTRM0jHK0sY>, <https://www.youtube.com/watch?v=KqIRza6dB4E>, <https://www.youtube.com/watch?v=t0CeNnEm0Wo>, <https://www.youtube.com/watch?v=LauQ6LWTkxM>, URL: <https://www.youtube.com/watch?v=eomALySSGVU> (visited on 08/12/2019).
 - [78] Prophesee, *Frame-based vs Event-based vision: comparison*, URL: <https://www.prophesee.ai/about-prophesee/> (visited on 07/24/2019).
 - [79] Peter R Huttenlocher, *Neural plasticity*, Harvard University Press, 2009.
 - [80] Hesham Mostafa et al., “Automated synthesis of asynchronous event-based interfaces for neuromorphic systems”, in: *2013 European Conference on Circuit Theory and Design (ECCTD)*, IEEE, 2013, pp. 1–4.
 - [81] Eugene M Izhikevich, “Which model to use for cortical spiking neurons?”, in: *IEEE transactions on neural networks* **15.5** (2004), pp. 1063–1070.
 - [82] Ning Qiao et al., “A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses”, in: *Frontiers in neuroscience* **9** (2015), p. 141.
 - [83] Saber Moradi et al., “A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)”, in: *IEEE transactions on biomedical circuits and systems* **12.1** (2017), pp. 106–122.
 - [84] Laurence F Abbott and Terrence Joseph Sejnowski, *Neural codes and distributed representations: foundations of neural computation*, Mit Press, 1999.
 - [85] David Heeger, “Poisson model of spike generation”, in: *Handout, University of Stanford* **5** (2000), pp. 1–13.
 - [86] Fabrizio Gabbiani and Christof Koch, “Principles of spike train analysis”, in: *Methods in neuronal modeling* **12.4** (1998), pp. 313–360.
 - [87] Laxmi R Iyer, Yansong Chua, and Haizhou Li, “Is neuromorphic mnist neuromorphic? analyzing the discriminative power of neuromorphic datasets in the time domain”, in: *arXiv preprint arXiv:1807.01013* (2018).
 - [88] Garrick Orchard et al., “Converting static image datasets to spiking neuromorphic datasets using saccades”, in: *Frontiers in neuroscience* **9** (2015), p. 437.
 - [89] Prophesee, *Prophesee website*, URL: <https://www.prophesee.ai> (visited on 07/24/2019).

-
- [90] Amos Sironi et al., “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.
 - [91] Bernabé Linares-Barranco Amirreza Yousefzadeh Teresa Serrano-Gotarredona, *MNIST-DVS dataset*, 2013, URL: <http://www2.imse-cnm.csic.es/caviar/MNISTDVS.html> (visited on 08/15/2019).
 - [92] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer, “Training deep spiking neural networks using backpropagation”, in: *Frontiers in neuroscience* **10** (2016), p. 508.
 - [93] Bipin Rajendran, “Algorithms and System Design for Brain-Inspired Spiking Neural Networks (SNNs) - Slides Presentation”, in: (2019), p. 64.
 - [94] Valentino Braitenberg, “Brain size and number of neurons: an exercise in synthetic neuroanatomy”, in: *Journal of computational neuroscience* **10.1** (2001), pp. 71–77.
 - [95] Peter Dayan and Larry F. Abbott, “Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems”, in: 2001.
 - [96] Dr Martin J. Spencer, “BMEN90002 Neural Information Processing, 2nd Semester 2018 - University of Melbourne, Slides Lecture 4 - Action Potential”, in: (2018), p. 38.
 - [97] Dr Martin J. Spencer, “BMEN90002 Neural Information Processing, 2nd Semester 2018 - University of Melbourne, Slides Lecture 3 - Synapses”, in: (2018), p. 20.
 - [98] Eugene M Izhikevich, “Polychronization: computation with spikes”, in: *Neural computation* **18.2** (2006), pp. 245–282.
 - [99] Dr Martin J. Spencer, “BMEN90002 Neural Information Processing, 2nd Semester 2018 - University of Melbourne, Slides Lecture 5 - Neural Coding (1)”, in: (2018), p. 37.
 - [100] Zachary F Mainen and Terrence J Sejnowski, *Reliability of Spike Timing in Neocortical Neurons*, tech. rep. 5216, 1995, pp. 1503–1506, URL: <https://redwood.berkeley.edu/wp-content/uploads/2018/08/mainen-sejnowski.pdf>.
 - [101] Alexander Borst and Frédéric E Theunissen, “Information theory and neural coding”, in: *Nature neuroscience* **2.11** (1999), p. 947.
 - [102] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen, “Spike-based strategies for rapid processing”, in: *Neural networks* **14.6-7** (2001), pp. 715–725.
 - [103] Werner M Kistler and Werner M Kistler, *Spiking neuron models: single neurons, populations, plasticity*, Cambridge University Press, 2002.
 - [104] Wolfgang Maass and Christopher M Bishop, *Pulsed neural networks*, MIT press, 2001.

-
- [105] Edgar Douglas Adrian, “The basis of sensation.”, in: (1928).
 - [106] Peter Lennie, “The cost of cortical computation”, in: *Current biology* **13.6** (2003), pp. 493–497.
 - [107] Sander M Bohte, Joost N Kok, and Han La Poutre, “Error-backpropagation in temporally encoded networks of spiking neurons”, in: *Neurocomputing* **48.1-4** (2002), pp. 17–37.
 - [108] Dr Martin J. Spencer, “BMEN90002 Neural Information Processing, 2nd Semester 2018 - University of Melbourne, Slides Lecture 15 - Neural Learning (3)”, in: (2018), p. 53.
 - [109] David Pérez-González and Manuel S Malmierca, “Adaptation in the auditory system: an overview”, in: *Frontiers in integrative neuroscience* **8** (2014), p. 19.
 - [110] Larry F Abbott and Sacha B Nelson, “Synaptic plasticity: taming the beast”, in: *Nature neuroscience* **3.11s** (2000), p. 1178.
 - [111] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström, “Spike-timing-dependent plasticity: a comprehensive overview”, in: *Frontiers in synaptic neuroscience* **4** (2012), p. 2.
 - [112] Prof. David B. Grayden, “BMEN90002 Neural Information Processing, 2nd Semester 2018 - University of Melbourne, Slides - Neural Models (L2)”, in: (2018), p. 37.
 - [113] Soni A Chaturvedi, “An Approach to Simulation of Leaky Integrate and Fire Model of SNN for Spike Generation.”, in: (2017).
 - [114] Prof. David B. Grayden, “BMEN90002 Neural Information Processing, 2nd Semester 2018 - University of Melbourne, Slides Lecture 15 - Neural Models (3)”, in: (2018), p. 26.
 - [115] Peter U Diehl et al., “Truehappiness: Neuromorphic emotion recognition on truenorth”, in: *2016 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2016, pp. 4278–4285.
 - [116] Andrew S Cassidy 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)*, IEEE, 2013, pp. 1–10.
 - [117] Alan L Hodgkin and Andrew F Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve”, in: *The Journal of physiology* **117.4** (1952), pp. 500–544.
 - [118] Richard FitzHugh, “Impulses and physiological states in theoretical models of nerve membrane”, in: *Biophysical journal* **1.6** (1961), pp. 445–466.

-
- [119] Richard FitzHugh, “Mathematical models of threshold phenomena in the nerve membrane”, in: *The bulletin of mathematical biophysics* **17.4** (1955), pp. 257–278.
 - [120] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu, “Phased lstm: Accelerating recurrent network training for long or event-based sequences”, in: *Advances in neural information processing systems*, 2016, pp. 3882–3890.
 - [121] Sepp Hochreiter and Jürgen Schmidhuber, “Long short-term memory”, in: *Neural computation* **9.8** (1997), pp. 1735–1780.
 - [122] Yujie Wu et al., “Spatio-temporal backpropagation for training high-performance spiking neural networks”, in: *Frontiers in neuroscience* **12** (2018).
 - [123] Shruti R. Kulkarni and Bipin Rajendran, “Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization”, in: *Neural Networks* **103**.May (2018), pp. 118–127, ISSN: 18792782, DOI: [10.1016/j.neunet.2018.03.019](https://doi.org/10.1016/j.neunet.2018.03.019), URL: <https://doi.org/10.1016/j.neunet.2018.03.019>.
 - [124] Maximilian Riesenhuber and Tomaso Poggio, “Hierarchical models of object recognition in cortex”, in: *Nature neuroscience* **2.11** (1999), p. 1019.
 - [125] Mojtaba Madadi Asl and Alireza Valizadeh, “Structure and Dynamics of Neuronal Networks Propagation Delays Determine the Effects of Synaptic Plasticity on the Structure and Dynamics of Neuronal Networks Ph . D . Dissertation Mojtaba Madadi Asl Supervisor : Dr . Alireza Valizadeh”, in: (2018).
 - [126] Siegrid Lowel and Wolf Singer, “Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity”, in: *Science* **255.5041** (1992), pp. 209–212.
 - [127] Carla J Shatz, “The developing brain”, in: *Scientific American* **267.3** (1992), pp. 60–67.
 - [128] Donald Olding Hebb, *The organization of behavior: A neuropsychological theory*, Psychology Press, 2005.
 - [129] Dr Martin J. Spencer, “BMEN90002 Neural Information Processing, 2nd Semester 2018 - University of Melbourne, Slides Lecture - Neural Learning (1)”, in: (2018), p. 33.
 - [130] Saeed Reza Kheradpisheh et al., “STDP-based spiking deep convolutional neural networks for object recognition”, in: *Neural Networks* **99** (2018), pp. 56–67.
 - [131] Timothée Masquelier and Simon J Thorpe, “Unsupervised learning of visual features through spike timing dependent plasticity”, in: *PLoS computational biology* **3.2** (2007), e31.

-
- [132] Joseph M Brader, Walter Senn, and Stefano Fusi, “Learning real-world stimuli in a neural network with spike-driven synaptic dynamics”, in: *Neural computation* **19.11** (2007), pp. 2881–2912.
- [133] Federico Corradi and Giacomo Indiveri, “A neuromorphic event-based neural recording system for smart brain-machine-interfaces”, in: *IEEE transactions on biomedical circuits and systems* **9.5** (2015), pp. 699–709.
- [134] Steve K Esser et al., “Backpropagation for energy-efficient neuromorphic computing”, in: *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.
- [135] Steven K Essera et al., “Convolutional networks for fast energy-efficient neuromorphic computing”, in: *Proc. Nat. Acad. Sci. USA* **113.41** (2016), pp. 11441–11446.
- [136] Christian Szegedy et al., “Rethinking the inception architecture for computer vision”, in: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.
- [137] Kaiming He et al., “Deep residual learning for image recognition”, in: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [138] Alex Krizhevsky, Geoffrey Hinton, et al., *Learning multiple layers of features from tiny images*, tech. rep., Citeseer, 2009.
- [139] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, “Imagenet classification with deep convolutional neural networks”, in: *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [140] Olga Russakovsky et al., “Imagenet large scale visual recognition challenge”, in: *International journal of computer vision* **115.3** (2015), pp. 211–252.
- [141] Rafael Serrano-Gotarredona et al., “On real-time AER 2-D convolutions hardware for neuromorphic spike-based cortical processing”, in: *IEEE Transactions on Neural Networks* **19.7** (2008), pp. 1196–1219.
- [142] Rafael Serrano-Gotarredona et al., “A neuromorphic cortical-layer microchip for spike-based event processing vision systems”, in: *IEEE Transactions on Circuits and Systems I: Regular Papers* **53.12** (2006), pp. 2548–2566.
- [143] Luis Camunas-Mesa et al., “A 32x32 Pixel Convolution Processor Chip for Address Event Vision Sensors With 155 ns Event Latency and 20 Meps Throughput”, in: *IEEE Transactions on Circuits and Systems I: Regular Papers* **58.4** (2010), pp. 777–790.

-
- [144] Luis Camunas-Mesa et al., “An event-driven multi-kernel convolution processor module for event-driven vision sensors”, in: *IEEE Journal of Solid-State Circuits* **47.2** (2011), pp. 504–517.
- [145] Mazdak Fatahi et al., “evt_MNIST: A spike based version of traditional MNIST”, in: *arXiv preprint arXiv:1604.06751* (2016).
- [146] Giacomo Indiveri, “A current-mode hysteretic winner-take-all network, with excitatory and inhibitory coupling”, in: *Analog Integrated Circuits and Signal Processing* **28.3** (2001), pp. 279–291.
- [147] Emre Neftci et al., “Synthesizing cognition in neuromorphic electronic systems”, in: *Proceedings of the National Academy of Sciences* **110.37** (2013), E3468–E3476.
- [148] Keras GitHub, *Keras documentation*, URL: <https://keras.io/models/sequential/> (visited on 08/10/2019).
- [149] Diederik P Kingma and Jimmy Ba, “Adam: A method for stochastic optimization”, in: *arXiv preprint arXiv:1412.6980* (2014).
- [150] Nan Zheng and Pinaki Mazumder, “A low-power hardware architecture for on-line supervised learning in multi-layer spiking neural networks”, in: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2018, pp. 1–5.
- [151] John G Daugman, “Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression”, in: *IEEE Transactions on acoustics, speech, and signal processing* **36.7** (1988), pp. 1169–1179.
- [152] Aristeidis Tsitiridis et al., “Gabor feature processing in spiking neural networks from retina-inspired data”, in: *2015 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015, pp. 1–8.
- [153] Syed Shakib Sarwar, Priyadarshini Panda, and Kaushik Roy, “Gabor filter assisted energy efficient fast learning convolutional neural networks”, in: *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, IEEE, 2017, pp. 1–6.
- [154] Shangzhen Luan et al., “Gabor convolutional networks”, in: *IEEE Transactions on Image Processing* **27.9** (2018), pp. 4357–4366.
- [155] Benoit Jacob et al., “Quantization and training of neural networks for efficient integer-arithmetic-only inference”, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.
- [156] Karen Simonyan and Andrew Zisserman, “Very deep convolutional networks for large-scale image recognition”, in: *arXiv preprint arXiv:1409.1556* (2014).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl