Performance of spiking neural networks on event data for embedded automotive applications

Loïc Cordone

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Performance des réseaux de neurones à spikes sur des données événementielles pour des applications automobiles embarquées

Loïc CORDONE

Laboratoire d'Électronique, Antennes et Télécommunications (LEAT)
& Renault Software Factory

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur

Dirigée par : Benoît Miramond, LEAT
Co-encadrée par : Philippe Thierion, Renault
Soutenue le : 16 Décembre 2022

Devant le jury, composé de :
Timothée Masquelier, Directeur de Recherche, CerCo
Emre Neftci, Professeur, Forschungszentrum Jülich
Jean Martinet, Professeur des Universités, I3S
Ryad Benosman, Professeur des Universités, Meta AI
Lyes Khacef, Docteur, Sony R&D
Amos Sironi, Docteur, Prophesee
Philippe Thierion, Ingénieur, Renault Software Factory
Benoît Miramond, Professeur, LEAT

Renault Group

Cifre

Jury

Loïc Cordone

Performance des réseaux de neurones à spikes sur des données événementielles pour des applications automobiles embarquées

Président du jury

Jean Martinet, Professeur des Universités, Université Côte d'Azur.

Rapporteurs

Timothée Masquelier, Professeur des Universités, CNRS.

Emre Neftci, Professeur, Forschungszentrum Jülich.

Examinateurs

Ryad Benosman, Professeur des Universités, Meta AI.

Lyes Khacef, Docteur, Sony R&D.

Amos Sironi, Docteur, Prophesee.

Philippe Thierion, Ingénieur, Renault.

Benoît Miramond, Professeur des Universités, Université Côte d'Azur.

Résumé

Performance des réseaux de neurones à spikes sur des données événementielles pour des applications automobiles embarquées

Aujourd'hui, les voitures embarquent de plus en plus d'algorithmes intelligents, appelés systèmes d'aides à la conduite (ADAS), qui cherchent à éviter l'apparition de situations dangereuses pouvant mener à des accidents. Ces algorithmes embarqués possèdent ainsi des contraintes très élevées en matière de latence, performance et consommation d'énergie. Les réseaux de neurones, vaguement inspirés par le fonctionnement des neurones biologiques, sont aujourd'hui les algorithmes d'intelligence artificielle les plus performants. Ils sont capables de répondre aux critères de latence et de performance demandés par les algorithmes embarqués automobiles, mais leur implémentation sur des architectures conventionnelles (CPU/GPU) résulte en une consommation énergétique élevée, accentuée par le fait qu'ils doivent tourner en permanence.

Une solution à ce problème pourrait résider dans l'utilisation de réseaux de neurones à spikes (SNNs), un type de réseau de neurones plus biologiquement plausible où les couches communiquent entre elles uniquement par le biais d'impulsions électriques appelées spikes, asynchrones et binaires. Grâce à ce fonctionnement, les SNNs promettent une consommation d'énergie moins élevée sur des architectures hardware spécialisées dites neuromorphiques. Ces architectures sont composées d'unités de mémoire et d'unités de calcul parallélisées et distribuées, comme c'est le cas dans notre cerveau. Ainsi, les besoins en énergie sont moindres car ils n'ont qu'à traiter des valeurs binaires sparses.

Un obstacle important à l'utilisation des SNNs pour traiter des problèmes de vision par ordinateur réside justement dans cette représentation de l'information sous forme de spikes. Une piste prometteuse pour représenter l'information visuelle sous forme de spikes est, encore une fois, de s'inspirer de la biologie. Un nouveau type de caméra, appelée caméra événementielle, capture l'information visuelle en utilisant des pixels photo-récepteurs détectant indépendamment les changements de luminosité. La sortie de ce type de caméra prend donc la forme d'événements, où chaque changement de luminosité est représenté par la position et le temps

(à la microseconde près) où il s'est produit et une valeur binaire indiquant si le changement de luminosité est positif ou négatif. En pratique, les changements de luminosité représentent le mouvement des objets, ainsi la caméra ne sort des événements qu'aux endroits et aux moments où un mouvement s'est produit. Ces événements sont binaires et extrêmement sparses, ils peuvent donc être vus comme des spikes et représentent une entrée idéale pour les SNNs.

Ainsi, nous étudions dans cette thèse la performance des réseaux de neurones à spikes pour le traitement de données événementielles, dans le but de construire des algorithmes intelligents qui soient performants, rapides et peu consommateurs d'énergie. Afin d'attaquer des problèmes de classification de données événementielles, nous avons développé de nouvelles méthodes d'apprentissage de SNNs basés sur des convolutions sparses et mis au point de nouvelles techniques qui permettent l'entraînement de SNNs très profonds (plus de 100 couches), atteignant des résultats à l'état de l'art en terme de précision et de sparsité. Nos résultats nous ont ensuite permis de traiter un problème plus pertinent pour des applications automobiles embarquées : la détection de voitures et de piétons sur des données événementielles, pour la première fois résolue avec des SNNs. Nous montrons aussi que les SNNs sont plus efficaces énergétiquement que des ANNs équivalents dans certaines conditions.

Les contributions de cette thèse en termes d'entraînement de SNNs, de modèles de réseaux et de représentation des données événementielles permettent d'atteindre de nouveaux sommets en matière de performances, rendant un peu plus réaliste l'utilisation des SNNs et des caméras événementielles dans des applications automobiles embarquées réelles à faible consommation d'énergie.

Mots clés — intelligence artificielle, réseaux de neurones à spikes, calcul événementiel, caméras événementielles, réseaux de neurones, systèmes embarqués

Abstract

Performance of spiking neural networks on event data for embedded automotive applications

Today, cars are increasingly equipped with intelligent algorithms, known as advanced driver assistance systems (ADAS), which seek to avoid the occurrence of dangerous situations that could lead to accidents. These embedded algorithms have very high constraints in terms of latency, performance and energy consumption. Neural networks, loosely inspired by the functioning of biological neurons, are today the most powerful artificial intelligence algorithms. They are able to meet the latency and performance requirements of automotive embedded algorithms, but their implementation on conventional architectures (CPU/GPU) results in high energy consumption, accentuated by the fact that they need to run continuously.

One solution to this problem may lie in the use of spiking neural networks (SNNs), a more biologically plausible type of neural network in which artificial neurons communicate with each other solely through asynchronous, binary electrical pulses called spikes. Because of this operation, SNNs promise lower power consumption on specialized hardware architectures, called neuromorphic architectures. These architectures are composed of parallelized and distributed memory and computing units, as it is the case in our brain. Thus, they require less energy as they only have to process sparse binary values.

A major obstacle to the use of SNNs to address computer vision problems is precisely this representation of information in the form of spikes, which is not the representation used for images. A promising way to represent visual information in the form of spikes is, once again, to draw inspiration from biology. A new type of camera, called an event camera, captures visual information using photoreceptor pixels that independently detect changes in brightness. The output of this type of camera therefore takes the form of events, where each change in brightness is represented by its position, time (to the microsecond precision) and a binary value indicating whether the change in brightness is positive or negative. In practice, the changes in brightness represent the movement of objects, so the camera only outputs events at the locations and times a movement has occurred. These events are binary and extremely sparse, so they can be seen as spikes and represent an ideal input for SNNs.

In this thesis, we study the performance of spiking neural networks for event data processing, with the objective of designing intelligent automotive algorithms that are efficient, fast, and energy-efficient. To tackle event data classification problems, we have developed new methods for training SNNs based on sparse convolutions and have developed new techniques that allow the training of very deep SNNs (more than 100 layers), achieving state-of-the-art results in terms of accuracy and sparsity. Our results then allowed us to address a problem more relevant to in-vehicle applications: the detection of cars and pedestrians on event data, for the first time solved with SNNs. We also show that SNNs are more energy efficient than equivalent ANNs under certain conditions.

The contributions of this thesis in terms of SNNs training, network models, and event data representation achieve new heights in performance, making the use of SNNs and event cameras in real-world, low-power automotive applications a bit more realistic.

Keywords — artificial intelligence, spiking neural networks, event-based processing, event cameras, neural networks, embedded systems

Acknowledgements

First of all, I would like to thank my thesis director Benoît Miramond, who accompanied and encouraged me during these 3 years of thesis, and for giving me the freedom to conduct my thesis as I wished. I would also like to thank Sonia Ferrante for making this thesis possible, and Philippe Thierion for having taken charge of the supervision of my thesis. I am also grateful to Andrea Ancora, who motivated me to tackle complex problems thanks to his communicative enthusiasm and his pertinent remarks.

I would like to express my gratitude to Timothée Masquelier, Emre Neftci, Jean Martinet, Ryad Benosman, Lyes Khacef and Amos Sironi for accepting to be part of my PhD defense jury. It is a source of pride to defend my thesis in front of these renowned researchers. In particular, I thank Timothée for closely following my work, which led to insightful discussions and remarks.

I obviously thank all the members of the eBrain team for all the discussions and laughs shared, whether it was around a Powerpoint or a beer. In particular, I would like to thank Pierre-Emmanuel Novac, my colleague who put up with me during countless coffee breaks, which were often surprisingly productive. I also thank my favorite chess partners: Luc, Edgar, Thomas and Laurent, even if the games were more pleasant for me than for them.

I would also like to thank all the friends I have made within Renault: Yasmine for the warm welcome and our long coffee breaks, Greg for our heated tech discussions, Matt for the many times you have been our tour guide, Céline for your skiing abilities and Yanis for your joie de vivre.

A warm thank you to my close friends William and Maria, for all dinners, trips, discussions we had. Thanks also to Maeva and Jules, our board game partners where the evenings always end with tears of laughter.

I also want to thank my eternal whiffers, Florent, Paul and Valentin, who have been with me most of the nights of this thesis in the essential quest of scoring more goals than strangers on the opposite side.

A special thanks to all my childhood friends, with whom I still have the chance to share amazing moments: Camille, Marion, Pierre, Bastien, Alice, Virginie, Lauriane, Louis, Victor, Marine, Mathieu. We grew up together and I can't thank you enough for all the laughs, classes, parties, birthdays and dramas we lived together, and the many ones to come.

Finally, I would like to thank my family, Cédric, Serena, my mother Sonia for her unconditional support, and those I consider as my family and who have made me the person I am, Nadine and Patrick Baschet. I also thank the extended Rohart family for their warm welcome and making me feel at home.

At last, I thank from the bottom of my heart Kélyane who supports and loves me, this thesis would not have been possible without her by my side.

You have all contributed to this thesis, in a way or another, and for that, I sincerely thank you,

Loïc Cordone, Dr.

Contents

List of Figures

List of Tables

List of Abbreviations

ACC . . . . . Accumulation operation.
ADAS . . . . . Advanced Driver Assistance Systems.
ANN . . . . . Artificial Neural Network.
BN . . . . . Batch Normalization.
BPTT . . . . . BackPropagation Through Time.
CNN . . . . . Convolutional Neural Network.
DNN . . . . . Deep Neural Network.
FC . . . . . Fully-Connected.
IF . . . . . Integrate-and-Fire neuron.
IoU . . . . . Intersection over Union.
HDR . . . . . High Dynamic Range.
LIF . . . . . Leaky Integrate-and-Fire neuron.
MAC . . . . . Multiply-And-Accumulate operation.
MLP . . . . . Multi-Layer Perceptron.
NLP . . . . . Natural Language Processing.
mAP . . . . . mean Average Precision.
PLIF . . . . . Parametric Leaky Integrate-and-Fire.
ReLU . . . . . Rectified Linear Unit.
RNN . . . . . Recurrent Neural Network.
SGD . . . . . Stochastic Gradient Descent.
SNN . . . . . Spiking Neural Network.
SSD . . . . . Single-Shot multibox Detector.
STDP . . . . . Spike-Timing-Dependent Plasticity.
TBPTT . . . . Truncated BackPropagation Through Time.
ViT . . . . . Vision Transformer.
YOLO . . . . . You Only Look Once.

1 Introduction

Contents

1.1 Motivation

The human brain is the most formidable processing system, capable of processing, integrating and coordinating the information it receives from the sensory organs to determine the instructions sent to the rest of the body. Composed of 80 billion neurons connected with over 150,000 kilometers of nerve fibers and 150 trillion synapses, the brain has a massively parallel architecture that requires only 20 watts to operate, an incredible efficiency that is a product of millions of years of evolution. The brain is still largely misunderstood: while individual neurons and synapses are considered well defined, it is still unclear how the exchange of simple electrical potentials leads to the emergence of complex functions such as language or cognition.

Loosely inspired by biological neural networks, Artificial Neural Networks (ANNs) are a family of artificial intelligence algorithms that have been very successful in the last ten years. They have advanced the state of the art across multiple domains, especially computer vision and Natural Language Processing (NLP). Today, the most powerful ANNs are composed of millions of artificial neurons spread over

hundreds of layers, making their simulation on classical Von Neumann architectures (CPUs and GPUs) energy intensive, making difficult their use in embedded systems.

Yet it is by being embedded that these powerful networks can have the biggest impact. Cars, for example, are increasingly equipped with Advanced Driver Assistance Systems (ADAS), intelligent algorithms designed to avoid the occurrence of dangerous situations that could lead to accidents. The high performance of neural networks on perception could improve their effectiveness, further lowering the number of accidents and potentially save lives. ADAS are embedded in the vehicle, and as such have very strong constraints in terms of performance, latency and energy consumption. While ANNs are able to meet the latency and performance requirements of automotive embedded algorithms, their implementation often results in high energy consumption, accentuated by the fact that they need to run continuously.

A possible solution would be the use of *Spiking Neural Networks* (SNNs), a more biologically plausible type of neural network in which artificial neurons communicate with each other solely through asynchronous, binary electrical pulses called spikes. Because of this operation, SNNs promise lower power consumption on specialized architectures, called neuromorphic architectures or *neuromorphic hardware*. These architectures are composed of parallelized and distributed memory and computing units, as it is the case in our brain. Moreover, they require less energy as they only have to process sparse binary values, removing the need for more energy-intensive multiplication operations.

Yet, a major obstacle to the use of SNNs to address computer vision problems is precisely this representation of information in the form of spikes, which is not the conventional representation used for images. Representing visual information in the form of spikes is, however, something that is done by our eyes, and once again we can draw inspiration from biology and design a new type of camera. Entitled *event cameras*, these sensors capture visual information using photoreceptor pixels that independently detect changes in brightness, a behavior close to what happens in our eyes. The output of this type of camera takes the form of *events*, where each change in brightness is represented by its position, its time (to the microsecond precision) and a binary value indicating whether the change in brightness is positive or negative. In practice, the changes in brightness represent the movement of objects, so the camera only outputs events at the locations and times a movement has occurred.

Event cameras provide a lot of interesting properties for embedded automotive applications. They have a high temporal resolution ($1\mu s$), enabling the capture of very fast movements without motion blur. They are also high dynamic range

(HDR) by design, which means it is very robust to extreme brightness conditions (e.g. night-time, sun glares) and to abrupt brightness changes (e.g. tunnel exit). On top of that, they are also low-power and output events only when the scene has changed, minimizing useless processing. And since events are binary and extremely sparse, they can be seen as spikes and represent an ideal input for SNNs.

In this thesis, we study how an end-to-end biologically-inspired approach that processes event data with spiking neural networks can be used to design intelligent embedded automotive algorithms that are high-performance, fast and energy-efficient.

1.2 Objectives

In this thesis, we aim to increase the performance of spiking neural networks for event data processing, in order to design intelligent automotive algorithms that are efficient, fast, and energy-efficient. In particular, we want to take SNNs to the next level, from processing toy examples with few-layers networks to the tackling of real-word automotive use-cases. This requires the use of the latest advances in terms of SNNs training methods, SNNs frameworks, available event datasets ; and the development of new methods and architectures to tackle problems previously out of the reach of SNNs.

Our quest for performance with SNNs on event data is coupled with the various constraints imposed by embedded automotive applications, thus all our developments are guided by embedded considerations such as the size of our networks, their sparsity, and their energy consumption.

To illustrate the performance increase of SNNs on event data and prove their usefulness, we will tackle real-world automotive use-cases that could, one day, become part of an ADAS. Such a use-case, chosen as our main objective, is the complex task of object detection, enabling the detection of dangers around the car such as other vehicles and pedestrians.

1.3 Contributions

As a first step towards real-world complex use-cases, we tackled event data classification problems, first on a simple gesture recognition problem. To do so, we developed new methods for training SNNs based on sparse convolutions, leading to the design of performing sparse SNNs, comparable in performance with ANNs. We then tackled the classification of automotive event data, through the recognition of cars.

More representative of a real use-case in terms of data size, this task required the development of new techniques and architectures that allow the training of very deep SNNs (more than 100 layers), achieving state-of-the-art results in terms of accuracy.

Having proved that complex SNNs architectures are able to learn complex patterns on large event datasets, our results then allowed us to address a problem more relevant to in-vehicle applications: the detection of cars and pedestrians on event data, for the first time tackled with SNNs.

Based on these works, we then propose two lightweight, scalable SNN architectures, ST-VGG and ResCat-SNN, able to further improve our results both in classification and object detection on event data, while being easier to embed than our previous networks.

Finally, in order to demonstrate and quantify the energy efficiency of SNNs compared to ANNs, we have developed an analytical estimation of the power consumption of any SNN, independent of particular hardware implementation choices. On three different tasks, keyword spotting, car recognition and image classification, our SNNs reach comparable or better results than equivalent ANNs while their estimated consumption is 6 to 8 times inferior.

Overall, the contributions of this thesis in terms of SNNs training, network models, and event data representation achieve new heights in performance, making the use of SNNs and event cameras in real-world, low-power automotive applications a bit more realistic.

1.4 Outline

This manuscript is organized as follows. Chapter 2 presents in details the different concepts and methods used in our work: ANNs, SNNs, event cameras, neuromorphic hardware. Chapter 3 presents our work on the development of new methods and new SNNs architectures to tackle the classification of event data, applied to gesture recognition and car recognition. Based on our findings, we present in Chapter 4 how we designed and trained the first spiking neural networks capable of doing object detection on complex, real-world event data in an automotive context. In Chapter 5, we propose new SNN architectures named ST-VGG and ResCat-SNN, which, along new training methods, reach new performance heights while being simpler to implement on embedded real-time systems than our previous SNNs. In Chapter 6, we end our thesis work by discussing the embeddability of SNNs, providing an analytical estimation of their energy efficiency compared to classical ANNs. Finally, Chapter 7 concludes the manuscript and outlines future works enabled by our thesis.

2

State of the art

Contents

This chapter outlines the theoretical concepts and methods that serve as a foundation for our work. In the first section, we briefly introduce traditional Artificial Neural Networks (ANN) to better understand their differences with Spiking Neural Networks (SNN) which are introduced in the second section. Third, we explore the literature on the training of spiking neural networks and the available frameworks for their development. Afterwards, we discuss event-based processing, from event cameras to machine learning methods applied to event data. Finally, we present the neuromorphic hardware architectures landscape.

2.1 Artificial neural networks

Artificial Neural Networks (ANNs) are computing systems inspired by the biological neural networks of animal brains. An ANN is a collection of connected units or nodes called artificial neurons, loosely modeling the biological neurons. Each connection, similarly to a biological synapse, can transmit a signal to other neurons, usually a real number. The output of artificial neurons is computed by some non-linear function of the sum of its inputs, such as sigmoid or Rectified Linear Unit (ReLU) (see Equations 2.1 and 2.2). This non-linear function is called an *activation function*.

$$\text{sigmoid}(x) = \frac{1}{1 + \exp -x}, \tag{2.1}$$

$$\text{ReLU}(x) = \max(0, x) \tag{2.2}$$

The connections, or synapses, typically have a weight that increases or decreases the strength of the signal at a connection. These weights can be learned by the network itself: during a phase called *training*, examples are input to the network and an error is computed between the network prediction and the expected result (the *target* or *label*). The network then adjusts its parameters (*weights*) according to a learning rule to minimize this error. After a sufficient number of these adjustments, the network can be considered trained and is ready for doing prediction on unknown data, a phase called *inference*. This training method is known as *supervised learning*.

ANNs are typically constructed by aggregating neurons into layers (see Fig. 2.1).

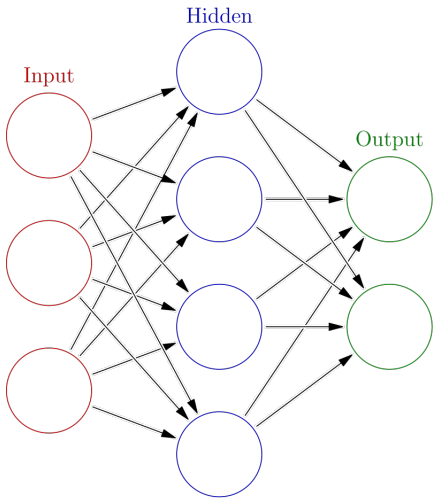


Figure 2.1: An Artificial Neural Network is an interconnected group of computational units called artificial neurons.

Neural networks computations were first modeled in (McCulloch and Pitts, 1943), and (Rosenblatt, 1958) designed the first artificial neural network called the *perceptron*, composed of a single output layer. But it was quickly shown that single-layer perceptrons are only capable of learning linearly separable patterns, which reduced research interest in neural networks for several decades. In (Hornik, Stinchcombe, and White, 1989), it has been shown that a Multi-Layer Perceptron (MLP) of only two layers was sufficient to approximate any non-linear separable function with an arbitrary precision. But the training of multi layered neural networks has remained a challenge until (Rumelhart, Hinton, and Williams, 1986) and (Rumelhart and McClelland, 1986) introduced the *backpropagation* algorithm to compute efficiently the gradient of a loss function with respect to the weights of the network for a single input–output example. The modern form of backpropagation learning algorithm for neural networks was presented in (Le Cun, 1987). Backpropagation computes the gradient of the loss function with respect to each weight using the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule. It is coupled with gradient methods for training multilayer networks, the most common being gradient descent or its stochastic approximation the Stochastic Gradient Descent (SGD).

But multi-layer perceptron models require a high number of neurons and connections to tackle complex classification tasks, especially for large input data such as images. Inspired by the visual cortex of cats (Hubel and Wiesel, 1959), Kunihiko Fukushima introduced the *neocognitron* (Fukushima, 1980), an ANN which use convolutional layers (and not fully-connected (FC) layers as in a MLP). Using the mathematical convolution operation, a convolutional layer (see Fig. 2.2) is composed of units whose receptive fields cover a patch of the previous layer. The weights of the convolution operation are shared across multiple neurons and are learned. This leads to a lower number of parameters to train and better results on images.

The first proof of the performance of these so-called Convolutional Neural Networks (CNNs) was showed in (Lecun et al., 1998) which introduced LeNet-5, a pioneering 7-level convolutional network trained with backpropagation capable of classifying hand-written digits digitized in 32x32 pixel images. The processing of higher-resolution images requires deeper convolutional neural networks, CNNs have therefore been limited for a long time by the computational resources available. In the 2010s, with the availability of large amount of data and cheap efficient computational resources (GPUs), CNNs and neural networks in general became more powerful and efficient than most of their competitors. Over the years, the

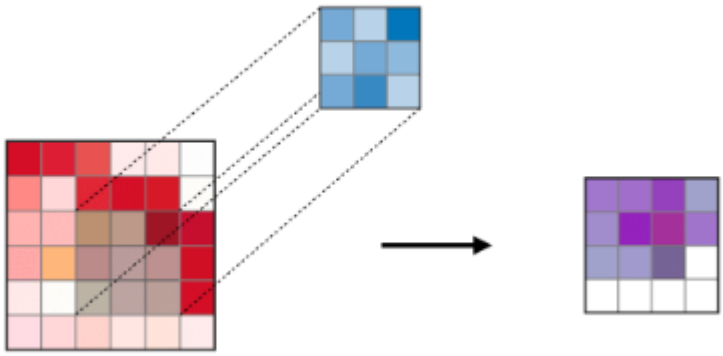


Figure 2.2: A convolutional layer uses learnable blocks of units (called *filters*, in blue) that scan the input data and perform convolution operations.

training of networks with more and more layers, an approach now called *deep learning*, has led to results that surpass traditional algorithms performance in many areas: computer vision, speech recognition, natural language processing, and more.

The introduction of batch normalization (Ioffe and Szegedy, 2015) and residual connections (K. He et al., 2016) have helped to mitigate the *vanishing gradient problem*, which prevents networks with a large number of layers from being trained correctly with backpropagation due to the gradient becoming vanishingly small in the first layers of the network, thus hindering the update of their weights. This led to the design of convolutional neural networks with hundreds of layers, with networks like Inception-ResNet (Szegedy et al., 2017) and some years later EfficientNet (Tan and Le, 2019) pushing ever higher the performances in various computer vision tasks: image classification, object detection, image segmentation, etc.

However, the implementation of large neural networks on conventional hardware (CPU/GPU) results in high power consumption, making their integration on embedded systems difficult. We explore in this thesis a more hardware friendly and energy efficient variant of neural networks called spiking neural networks.

2.2 Spiking neural networks

Spiking Neural Networks are a type of artificial neural networks that more closely mimic biological neural networks. In addition to modeling neurons and synapses, SNNs incorporate the concept of time into their operating model. Each neuron of an SNN has a *membrane potential* representing its membrane electrical charge. The input information charge the neuron potential, and when it reaches a specific value, called the *threshold*, the neuron fires a *spike* to the neurons it is connected to. Thus, SNNs do not transmit information at each propagation cycle as it is

the case with traditional ANN, and the information transmitted, the spikes, are binary and discrete in time.

2.2.1 Neuron models

Along the years, several neuron models have been proposed to model the biological neurons. Hodgkin and Huxley (1952) proposed a mathematical model to explain the mechanisms underlying the initiation and propagation of action potentials in the squid giant axon. Their model, the Hodgkin-Huxley (HH) neuron model, is a set of nonlinear differential equations that approximates the electrical characteristics of neurons, perhaps the closest to the real functioning of biological neurons. But the numerical integration of the equations is so computationally expensive that simplifications of the HH neuron are most often used to model spike neural networks.

Several neuron models proposed along the years are compared in (Izhikevich, 2004), by considering their biological plausibility, based on a list of 20 neuro-computational features, and their implementation cost expressed as the number of Floating Point OPerations per Seconds (FLOPS) required to simulate the model for 1 ms. The results of this comparison are illustrated in Fig. 2.3. The higher the bio-plausibility of a neural model, the higher its implementation cost. As a result, the simplest neuron models are often the most widely used for large-scale simulations of spiking neural networks, especially for machine learning tasks as they require thousands or even millions of neurons.

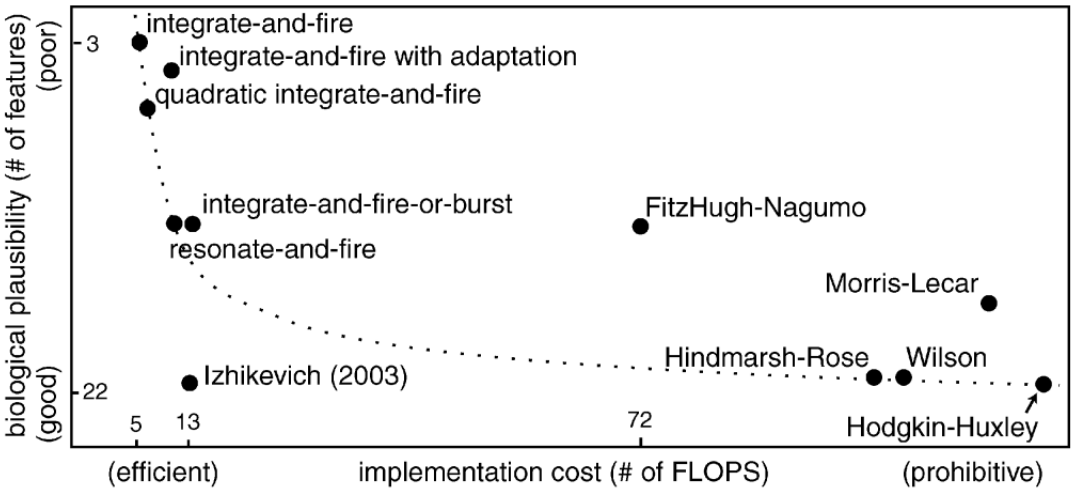


Figure 2.3: A comparison of the neuro-computational properties of several neuron models, presented in (Izhikevich, 2004).

The Izhikevich neuron model has a good bio-plausibility while being efficient, but in practice it is difficult to use in neural networks built for machine learning tasks as it use a quadratic function for its membrane potential update.

Therefore, for the design of our spiking neural network, we mainly used in this thesis the Integrate-and-Fire (IF) and Leaky Integrate-and-Fire (LIF) neuron models, introduced in 1907 by Lapicque (Abbott, 1999). In the IF model, a neuron with a membrane capacity C is represented by its membrane voltage V which evolves in time during stimulation with an input current $I(t)$ according to Equation 2.3:

$$I(t) = C\frac{dV(t)}{dt} \tag{2.3}$$

When an input current is applied, the membrane voltage increases with time until it reaches a constant threshold V_{th}, at which point a spike is emitted and the voltage is reset to its resting potential V_{rest}.

A slightly more biologically plausible version of the IF neuron is the Leaky Integrate-and-Fire neuron, where the model includes a *leak* term in the membrane potential equation, reflecting the leak current over time present in biological neurons' membranes. The LIF neuron models the leak by a linear resistor R in parallel with the membrane, we can thus define the dynamics of a LIF according to Equation 2.4:

$$I(t) = \frac{V(t) - V_{rest}}{R} + C\frac{dV(t)}{dt} \tag{2.4}$$

If we introduce the membre time constant $\tau_m = RC$ and multiply Eq. 2.4 by R, we obtain the standard form of a LIF neuron:

$$\tau_m\frac{dV(t)}{dt} = -(V(t) - V_{rest}) + RI(t) \tag{2.5}$$

The solution of this differential equation with an initial condition $V(t_0) = V_{rest} + \Delta u$ is given by Equation 2.6:

$$V(t) - V_{rest} = \exp(-\frac{t - t_0}{\tau_m})\Delta u \text{ for } t > t_0 \tag{2.6}$$

Thus, in the absence of input, the membrane potential decays exponentially to its resting value, with τ_m the characteristic time of the decay, in seconds.

In order to use this LIF model in neural networks operating on discrete timesteps, the previous equations can be discretized:

$$V[n] - V_{rest} = \exp(-\frac{t - t_0}{\tau_m})V[n-1] + S[n], \tag{2.7}$$

with $S[n]$ the input spikes at timestep n.

An illustration of a LIF neuron is provided in Fig. 2.4. Spikes are input to the neuron at different time and being summed in the internal membrane potential of the LIF. The membrane potential over time is depicted in the center graph. The membrane potential decreases over time when no spike is received. When the potential reaches the threshold, the neuron emits one spike at this precise time to every neuron it is connected to.

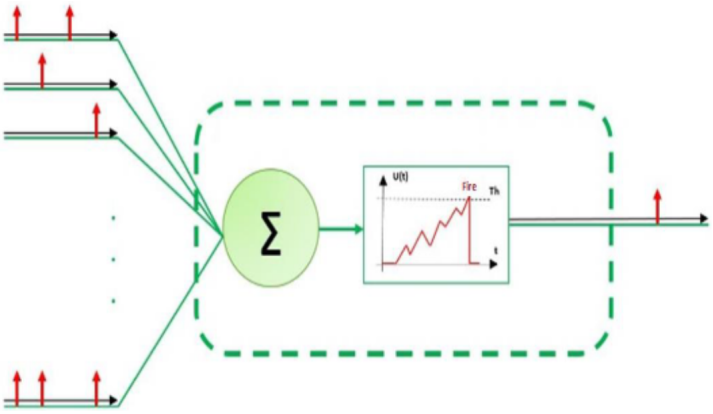


Figure 2.4: A Leaky Integrate-and-Fire neuron model.

2.2.2 SNN design and information encoding

Similar to ANNs, spiking neural networks are designed by stacking interconnected layers of neurons. The neuron model, usually an IF or a LIF, acts as an activation function: it either outputs 0 or 1 depending on the membrane potential value. Therefore, each layer communicates with the following one using spikes. SNNs layers are heavily inspired by those of ANNs: fully-connected layers, convolution layers and pooling layers are often used as is or with slight modifications (e.g. adding lateral connections to convolutional layers, as in Cheng et al., 2020). But in SNNs, these layers operate on binary values, effectively eliminating the multiplication operations between the input data and the learned weights, as these are the most energy consuming operations in a neural network.

Another major difference of SNNs with ANNs is that they operate over time. Usually, this is modeled using time steps. However, the vast majority of information processed by neural networks, i.e. images or speech, can not be represented by a train of spikes. In order to be usable by SNNs, information therefore needs to be encoded. Encoding strategies can be broadly divided in three categories (Auge et al., 2021): rate, temporal and direct encoding.

Rate coding

As the name suggests, rate coding (Adrian and Zotterman, 1926) is based on the spikes' firing rates to represent information: a specific value is encoded by the number of spikes emitted over a pre-specified time window. A common use-case of rate coding is to convert each pixel value of images to spike trains of different frequency, sampling spikes from a Poisson distribution. Thus, to accurately represent N values we need at least N timesteps. Since the number of timesteps is directly correlated to the precision of the possible values encoded, rate coding leads to the generation of a lot of spikes over a large number of timesteps, which drastically impacts computational and energy efficiency.

Despite its significant flaws, rate coding is nonetheless a popular encoding scheme for SNNs as it is easy to implement on a variety of data, and can be used to convert ANNs into SNNs with a low performance loss, as we will discuss in Section 2.3.1.

Temporal coding

Biological systems need to respond almost instantly to stimuli: for example, S. Thorpe, Fize, and Marlot (1996) showed that the human brain needs less than 150 ms to process a complex visual task. This has led to the creation of information encoding that uses the precise timing of spikes instead of their mean firing rates: temporal coding. The most two common types of temporal coding are the Time To First Spike (TTFS) coding and the rank order coding. The TTFS coding scheme (Johansson and Birznieks, 2004) transmits information on the arrival of the first spike, enabling a fast transmission speed. On the other hand, rank order coding (Simon Thorpe and Gautrais, 1998) is based on the firing order of a population of neurons, without considering their precise timing. Using either of these encodings in SNNs can significantly reduce the number of spikes and improve inference speed compared to rate coding (Rueckauer and S.-C. Liu, 2018, Park et al., 2020), but it still requires tens of timesteps to attain good performance (Kheradpisheh and Masquelier, 2020, W. Guo et al., 2021).

The difference between rate coding and temporal coding is illustrated in Fig. 2.5.

Direct coding

The supposed energy savings of SNNs due to the nature of their operations are often counterbalanced by an increase of the processing time (Lemaire, 2022). Most often, this is due to the large number of timesteps, which is a consequence of the information coding chosen for a particular type of data. Direct coding aims to

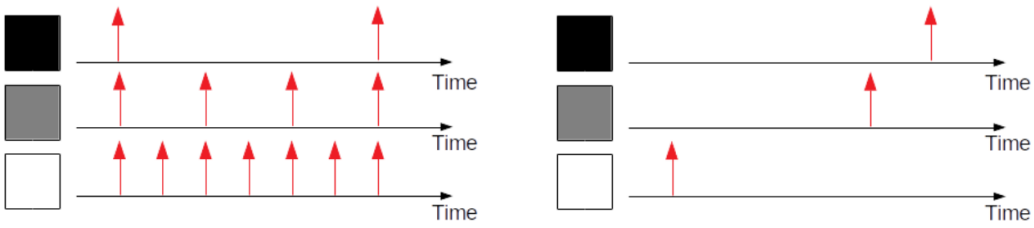


Figure 2.5: Difference between rate coding (left) and temporal TTFS coding (right) when encoding pixel intensities. Illustration derived from (Lemaire, 2022).

solve this problem by having the network itself learn a spike encoding relevant to the considered task on a fixed number of timesteps T. Therefore, the raw data is repeatedly (T times) input to the first layer of the network, that will then outputs spikes. The first layer is directly encoding the data as spike trains, which allows the training of SSNs on a extremely low number of timesteps (as low as 2). The downside of this approach is that the first layer of the network, often the most expensive, has to compute multiplication operations since the input data is not longer binary spikes. Recently, several works showed that direct coding provides better performance and better energy efficiency both theoretically, as we will see in Section 6, and experimentally (Y. Kim et al., 2022) than rate coding.

Native coding

Finally, the best possible spike encoding is the one that is unnecessary: if we can train SNNs directly on data made up of spike trains, we can be sure that we will not lose precision over the original data. Biological spike trains and event data, whether from event cameras or silicon cochleas (Yang et al., 2016), are data types naturally compatible with SNNs. In this thesis, we explore the processing of event data directly with spiking neural networks, avoiding the use of specific encoding that imposes a high number of timesteps or complicates the training.

2.3 Training of spiking neural networks

A major drawback of SNNs is the non-differentiable activation function, meaning that backpropagation, the most widely used learning algorithm in ANNs, cannot be directly employed with SNNs. This section presents the three main training processes of spiking neural networks: converting a trained ANN, or directly training an SNN using either local unsupervised learning (STDP) or global supervised learning (spike-based backpropagation).

2.3.1 Conversion ANN-SNN

The most commonly used method for creating an SNN is by converting an already trained ANN into an SNN (Diehl et al., 2015). The main idea is to replace the ReLU activation functions in the ANN by IF neurons, as they bear functional equivalence. By keeping the same learned weights and using rate coding to encode the input data, it is possible to approximate with spikes the analog values originally generated by the ANN. Some constraints must be respected during the ANN training to be able to do the conversion: the neurons generally have no biases (batch normalization thus can not be used either) and a particular weight normalization has to be used to help regulate firing rates. Converting an ANN into an SNN induces a performance drop, although the most recent conversions have managed to provide comparable performance with their ANN counterpart (Sengupta et al., 2019), (Tavanaei et al., 2019), (Ding et al., 2021).

However, even if it is relatively simple to implement, designing SNNs by conversion is for us the worst approach for multiple reasons. Since the input data is deteriorated by rate coding, and that the converted layers only approximates the original analog values, we are sure to be inferior to the equivalent ANN. This could be acceptable if we could provide gains elsewhere, but the claimed energy-saving benefits of SNNs are difficult to encounter when converted SNNs usually have a very high number of timesteps, impacting its latency and thus its energy consumption.

Due to these limitations, the spiking community is now leaning more towards direct training of SNNs, using either unsupervised rules such as STDP or variants of the backpropagation learning rule adapted to SNNs.

2.3.2 Spike-Timing Dependant Plasticity (STDP)

Spike-Timing-Dependant Plasticity (STDP) (Song, Miller, and Abbott, 2000), (Toyoizumi et al., 2004), is a bio-plausible unsupervised learning rule where weights are changed based on the delay between the firing of the presynaptic and postsynaptic neurons. On top of being unsupervised, STDP is also a local learning rule: an update only requires information about the uphill and downhill neurons. Basically, the principle of STDP learning is the following: if an output spike is generated shortly after receiving an input spike, the connection is potentiated, and inversely when an input spike does not cause an output spike, the connection is depreciated. This principle is illustrated in Fig. 2.6.

The fact that the STDP rule is local makes it more hardware-friendly than most supervised rules, especially since the computation itself is much simpler

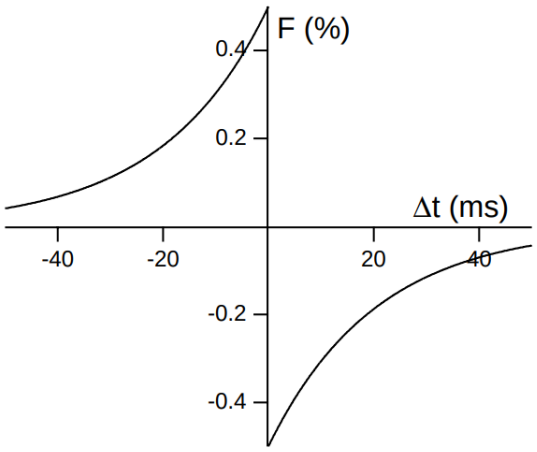


Figure 2.6: The STDP modification rule, potentiating or depreciating the connection depending on the timing of the presynaptic and postsynaptic spikes, as described in (Song, Miller, and Abbott, 2000).

that the computation of chained derivatives. But to date, SNNs trained with the STDP rule are generally not competitive with supervised learning Srinivasan et al., 2020, although recent progress in the training of convolutional SNNs with STDP (Kheradpisheh, Ganjtabesh, et al., 2018) or reward-modulated STDP (Mozafari et al., 2019) could close this gap in the future. It is however unlikely that STDP could compete with supervised learning approaches when the task becomes more difficult than classification on small images.

2.3.3 Backpropagation on spikes

Spikes are emitted following an Heaviside step function, which is nondifferentiable. Indeed, its derivative is zero everywhere except at 0, where it is not defined. This makes spiking neurons unsuitable for gradient-based optimization. Numerous works have studied how to overcome this limitation and enable the training of SNNs directly with gradient-based learning rules.

SpikeProp (Bohte, Kok, and La Poutré, 2002) was one of the first supervised learning method for SNNs. Its general approach is to formulate SNNs in a way that ensures well-behaved gradients, directly suitable for optimization with backpropagation. To achieve this, they considered the outputs of spiking neurons to be a set of firing times, corresponding to a temporal coding setting. They used a feedforward multilayer SNN in which a connection between two neurons consists of a defined number of sub-connections, each with a different delay and a trainable weight (see Fig. 2.7). SpikeProp is then an error-backpropagation learning rule obtained by linearizing the analytic expressions of firing times for hidden units, enabling

the analytical approximation of hidden layer gradients. But the main limitation of SpikeProp is that it requires each neuron to emit exactly one spike per trial, as a non-spiking neuron would not have a firing time and thus prevent its gradient calculation. This limits the learning capabilities of the network and goes against the low-power approach of SNNs by having all its computing units constantly active.

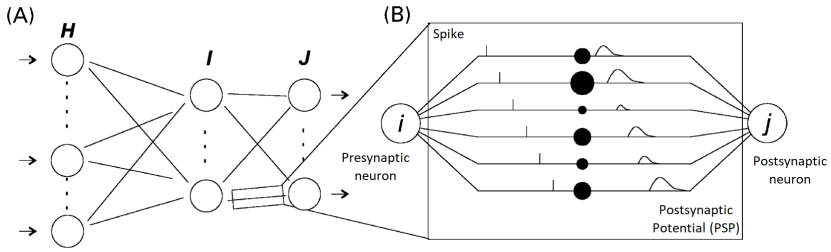


Figure 2.7: (A) SNN architecture used in (Bohte, Kok, and La Poutré, 2002), (B) connection consisting of multiple delayed synaptic terminals. Illustration adapted from (Bohte, Kok, and La Poutré, 2002).

Another approach to have well-defined gradients in SNNs is to use a rate coding scheme, and to consider that information is carried by the spike rate. For some neuron models, notably the IF and LIF neurons, the firing rate depends smoothly on the neuron input, and can be thus used for gradient-based learning. This is the approach used in (Hunsberger and Eliasmith, 2015) and (E. O. Neftci, Augustine, et al., 2017), attaining competitive performance on standard machine learning benchmarks such as CIFAR10 (82.95% accuracy) and MNIST (98.27%). But since these approaches are based on rate coding, they are as inefficient as the SNNs designed by conversion. They usually require high firing rates and thus a high number of timesteps to encode precise values, which once again reduces the low-power appeal of SNNs.

Instead of changing the model definition in order to obtain well-defined gradients, a now popular approach to the direct training of SNNs is the surrogate gradient learning rules.

To circumvent the non-differentiability of spikes in SNNs, the main idea of surrogate gradient learning is to use two distinct functions in the forward and backward passes: an Heaviside step function in the former, and a differentiable approximation of the Heaviside in the latter, such as a sigmoid function (see Fig. 2.8).

This approximation is sufficient for updating the weights of the SNNs using backpropagation. Indeed, (E. O. Neftci, Mostafa, and Zenke, 2019) demonstrated that SNNs constitute a special case of Recurrent Neural Networks (RNNs), and as such, can be trained using the same techniques. RNNs operate on a fixed number of

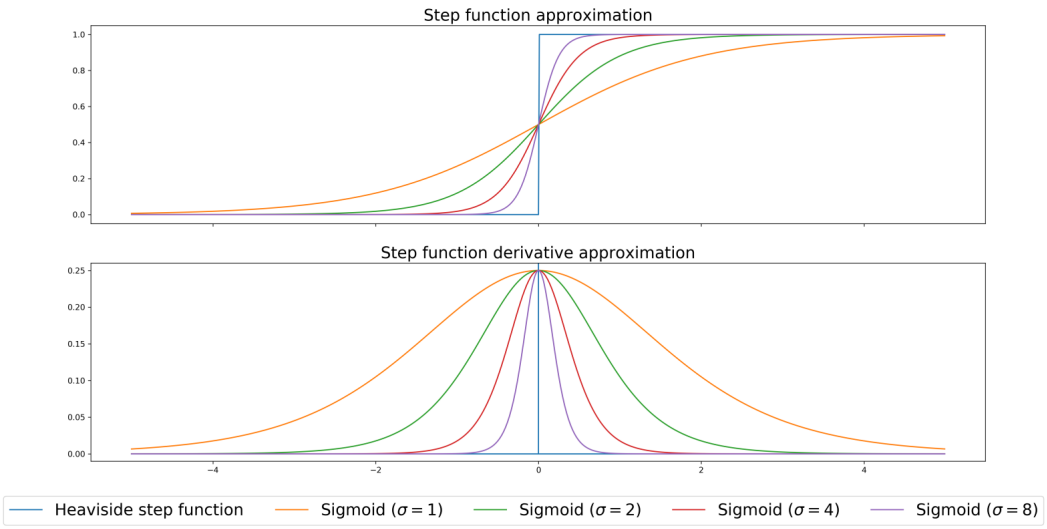


Figure 2.8: Approximation of the Heaviside step function with a sigmoid function with different σ, with the derivative approximations rescaled to have a maximum of 0.25. Illustration taken from (Zimmer et al., 2019).

timesteps, and in order to be trained with backpropagation the recurrence is *unrolled*: an auxiliary network is created by making copies of the original for each time step (see Fig. 2.9). Therefore, the unrolled network is simply a deep network with shared feedforward weights and recurrent weights, on which the standard backpropagation can be applied. The process of applying backpropagation to an unrolled network is known as backpropagation through time (BPTT). An important problem with BPTT is that the higher the number of timesteps, the deeper the unrolled network will be, which can make the training difficult (vanishing gradient problem) or computationally expensive. For these reasons, the number of timesteps of SNNs trained with surrogate gradient learning is generally lower than with other methods.

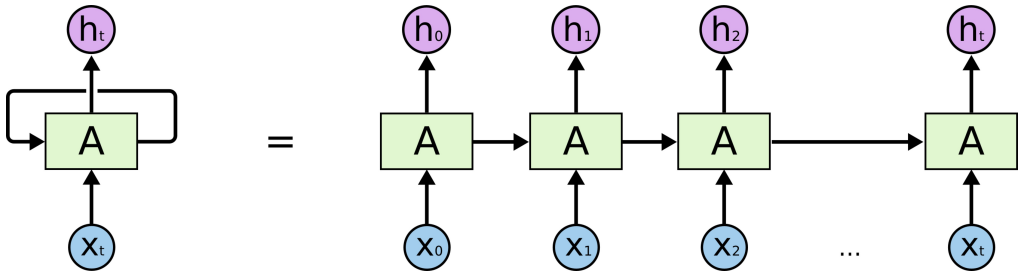


Figure 2.9: Unrolling of a recurrent net in order to train it with backpropagation through time (BPTT).

Surrogate gradient learning has been implemented with different surrogate functions in several works. Bohte, 2011 approximated the spiking neuron non-linearity by using a truncated quadratic function, essentially using a ReLU function

as the surrogate derivative. Years later, (Zenke and Ganguli, 2018) proposed SuperSpike, a three factor learning rule to train multi-layer SNNs of LIF neurons using a fast sigmoid as the surrogate function; while SLAYER (Shrestha and Orchard, 2018) used an exponential function, reporting competitive performance on a range of neuromorphic classification datasets (99.2% on NMNIST, 93.64% on DVS Gesture). S2NET (Zimmer et al., 2019) is another work using a surrogate gradient approach to train convolutional SNNs for speech recognition, using this time a parametrized sigmoid function as the surrogate function (see Fig. 2.8).

In our opinion, surrogate gradient learning rule is the best way to train complex and powerful SNNs. By providing an easy and customizable way of training SNNs with backpropagation, surrogate gradient has enabled the SNNs to take a major step forward. Formerly limited to toy problems and shallow networks, SNNs with hundred of layers and millions of parameters are now trained quickly on more and more complex problems, as we shall show in this thesis.

2.3.4 Spiking neural networks training frameworks

Frameworks for building and simulating large neuronal networks are vital to the tackling of real-world tasks with SNNs. Initially, these frameworks were aimed at the neuroscientific community, but in the last few years many frameworks based on popular deep learning frameworks (PyTorch, TensorFlow) have emerged. We briefly introduce the existing frameworks for SNNs in this section.

Simulators of spiking neural networks

Mainly used by neuroscientists, simulators of spiking neural networks provide an easy way of designing complex networks of spiking neurons. Also called brain simulators, their main features are computational performance, code simplicity for describing neuron models and synapses, and the integration with parallel HPC platforms. The most popular simulators are NEURON (Carnevale and Hines, 2006), NEST (Gewaltig and Diesmann, 2007) and Brian (Stimberg, Brette, and Goodman, 2019).

They provide a multitude of neuron and synapse models, along with efficient methods to define and connect large networks. Different neuron and synapse models can coexist inside a network, and multiple connections with different properties can link any two neurons. They usually implements learning by using variants of STDP, and even let the possibility to implement new learning rules.

These simulators are written in C or C++, to benefit from their speed and computational efficiency. However, they use Python as a second language for

the description of neural networks as a matter of simplicity. This means that the computational architecture is processed by an interpreter, implying that the network structure does not change during simulation. As such, Davison et al. (2009) proposed PyNN , a simulator-independent Python package for building spiking neuronal networks models that would run without modification on any simulator supported (NEURON, NEST and Brian). By providing a library of neurons, synapse and synaptic plasticity models that work the same on all the supported simulators, PyNN offers a powerful, high-level interface for the simulation of neuronal network models.

However, using these brain simulators for tackling complex machine learning tasks is difficult, as organizing neurons in layers and using operations like convolutions require some implementation efforts. Also, they generally do not propose a supervised learning rule as they do not include an automatic differentiation engine. For all these reasons, a different kind of SNNs simulators have emerged, specialized in machine learning tasks.

Machine learning oriented SNN frameworks

With the widespread use of neural networks frameworks such as PyTorch and TensorFlow, designing large neural networks with high computational efficiency on CPUs and GPUs have never been easier. These frameworks provide a high-level Python interface that can be simply used to define highly fast and efficient new layers and learning rules. These frameworks provide an easy way to add new operations and new layers, and as a result several SNN frameworks are now available.

BindsNET (Hazan et al., 2018) was the first spiking neural network framework geared towards the development of biologically inspired algorithms for machine learning. BindsNET uses PyTorch operations to construct SNNs with usual machine learning operations (convolutions, pooling) that can run on GPU devices seamlessly. The learning is done primarily with the STDP rule and its variants. Thanks to the GPU acceleration, large networks (over several thousands of neurons) have a lower simulation time using BindsNet than the popular brain simulators (Brian, NEST), proving the importance of leveraging existing deep learning frameworks for the construction of deep SNNs.

SlayerTorch (Shrestha and Orchard, 2018) is another PyTorch-based framework that implements the Spike LAYer Error Reassignment (SLAYER) supervised rule for training SNNs. It supports the learning of both synaptic weights and axonal delays. The latest version, SLAYER 2.0, is now included in LAVA (Intel, 2022), a more general framework for neuromorphic processing that targets in particular the Intel Loihi neuromorphic hardware.

Another notable frameworks are Norse (Pehle and Pedersen, 2021), a recent SNN framework based on PyTorch that allows the training of SNNs with the SuperSpike learning rule, and Nengo-DL (Rasmussen, 2019). Nengo-DL is based on TensorFlow / Keras and extends the SNN simulator Nengo (Bekolay et al., 2014) with machine learning operations and the rate-coding based backpropagation proposed in (Hunsberger and Eliasmith, 2015). The main asset of Nengo is the diversity of target hardwares, as multiple addons allow the implementation of Nengo networks on FPGA or neuromorphic hardwares such as Loihi and SpiNNaker.

Finally, SpikingJelly (Fang, Y. Chen, et al., 2020) is yet another PyTorch-based framework for training SNNs using the surrogate gradient learning rule that is substantially faster than its competitors. Indeed, SpikingJelly contains neuron-specific CUDA kernels, either generated with torch or with CuPy, making possible the simulation of millions of LIF neurons in less than a second on GPU. SpikingJelly represents for us the best framework for the design of large, complex SNNs over tens of timesteps, making the training of SNNs almost as simple and fast as that of DNNs.

The Table 2.1 summarizes the different SNN frameworks learning rules and target hardware. For now, LAVA and Nengo are the only frameworks capable of porting an SNN network directly to neuromorphic hardware, but we are aware that research is underway to port SNNs trained with SpikingJelly on Loihi and other neuromorphic hardware.

Table 2.1: Comparison between the most popular machine learning oriented spiking neural networks frameworks.

Frameworks	Based on	Learning rules	Target hardware
BindsNet	PyTorch	STDP, Hebbian	CPU/GPU
LAVA	PyTorch	SLAYER	Loihi, CPU/GPU
Nengo-DL	TF/Keras	Rate-coding BP, Oja, BCM	Loihi, SpiNNaker, FPGA, CPU/GPU
Norse	PyTorch	BP with SG, STDP	CPU/GPU
snnTorch	PyTorch	BP with SG	CPU/GPU
SpikingJelly	PyTorch	BP with SG, STDP	CPU/GPU

2.4 Event-based processing

Event cameras, or event sensor or event-based sensor, are bio-inspired sensors that differ from conventional frame cameras in the sense that they don't capture images

at a fixed rate. Instead, they output a stream of *events*, each event containing the time, location and sign of a brightness change.

Events are binary, discrete in time and space: they can be seen as spikes. Event data are therefore a type of data that is natively compatible with SNNs. But due to their sparse nature, the literature is still struggling to process event data. We present in this section a brief overview of event-based processing.

2.4.1 Event cameras

Traditional sensors acquire full images at a rate specified by an external clock (e.g., 60 fps). Inspired by the spiking nature of biological visual pathways, event cameras capture visual information through the means of a grid of independent photoreceptive pixels, each outputting asynchronously an event when a bightness (log intensity) variation is detected. The output of this kind of sensor compared to a classical frame is depicted in Fig. 2.10

Figure 2.10: Output of an event sensor (on the right). To reconstruct this frame, events have been accumulated over a fixed time window equivalent to the duration of a single traditional frame (60ms), with blue and black events representing ON and OFF events respectively. Figure taken from (Prophesee, 2021).

More precisely, each pixel of the event sensor memorizes the log intensity each time it sends an event, and continuously monitors for a change of sufficient magnitude from this memorized value. When the change exceeds a threshold, the camera emits an event, which contains the x, y location, the time t and the 1-bit polarity p of the change (brightness increase or decrease). The schematic of the operation of a DVS pixel is illustrated in Fig. 2.11. The log intensity of the scene changes in continuous time (upper graph), and each time it is greater than the ON threshold, it produces an ON event and is then reset (lower graph). Similarly,

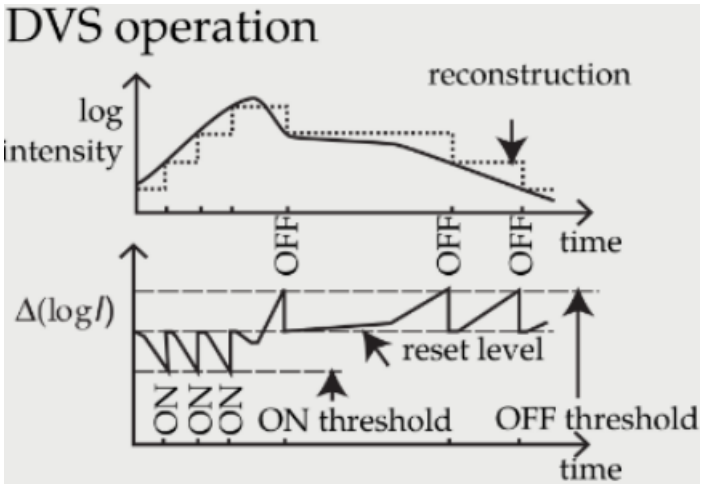


Figure 2.11: Schematic of the operation of a DVS pixel, converting light into events. Illustration from (Neil, 2017).

when the log intensity decreases until it reaches the OFF threshold, an OFF event is produced and the log intensity is reset.

The camera outputs events using a shared digital output bus, most often using address-event representation (AER) readout (Boahen, 2004). Event cameras output depends on the amount of motion or brightness change in the scene, the faster the motion, the more events per second are generated. The output bus can become saturated, perturbing the times that events are sent. Available event cameras have readout ranges ranging from 2MHz to 1200MHz, which is in direct correlation with the resolution of the camera.

Due to their operation, event cameras offer attractive properties. They offer a *high temporal resolution*, events are readout digitally with a 1MHz clock, meaning that events are detected with a microsecond resolution. They can therefore capture very fast motions, without suffering from motion blur. They are *low latency*, since each pixel works independently it can output events in less than a millisecond in real world scenes. Event cameras have by design a very High Dynamic Range (HDR) of $140dB$, notably higher than the $60dB$ of frame-based cameras. Therefore, they are able to acquire information in the moonlight without problems, since the pixels operate independently in logarithmic scale. Finally, they are *low power*, as power is only used to process changing pixels. Most cameras use about 10mW, compared to frame-based cameras that consume a hundred times more power (>1W).

The principal event cameras manufacturers are iniVation, Prophesee, Samsung and CelePixel. IniVation released in 2008 the pioneering DVS128 (Lichtsteiner, Posch, and Delbruck, 2008) with a resolution of 128×128 pixels. In 2011, Prophesee presented their first event camera ATIS with a 304×240 resolution (Posch, Matolin,

and Wohlgenannt, 2011). Over the years, the resolution of event cameras kept increasing to attain 1 megapixel: the Prophesee GEN4 CD (Finateu et al., 2020), the Samsung DVS-Gen4 (Suh et al., 2020) and the CelePixel CeleX-V (S. Chen and M. Guo, 2019) all offer a resolution superior to 1280×720 pixels. The resolution is limited by the pixel size, and it is still unclear if manufacturers will be able to continue increasing the resolution of their event cameras, which is still much inferior to that of the frame-based cameras. Another limitation of event sensors is that they do not capture color, but recent advances have resulted in the development of color event cameras such as the iniVation Color-DAVIS346 (Taverni et al., 2018), although it has not yet been proven that color provides better results on event data as it is the case with traditional frames.

Table 2.2: Comparison of principal characteristics of available commercial event cameras. Values taken from (Gallego, Delbrück, et al., 2022).

Event Camera		Year	Resolution (pixels)	Latency (µs)	Dynamic range (dB)	Power cons. (mW)	CMOS tech. (nm)
iniVation	DVS128	2008	128×128	12	120	23	350
	DAVIS346	2017	346×260	20	120	10-170	180
Prophesee	ATIS	2011	304×240	3	143	50-175	180
	GEN4 CD	2020	1280×720	20-150	>124	32-84	90
Samsung	DVS-Gen4	2020	1280×960	150	100	130	65/28
CelePixel	CeleX-V	2019	1280×800	8	120	400	65

A brief comparison of commercial event cameras is depicted in Table 2.2. In this thesis, we mostly work with the Prophesee event cameras, the ATIS and the GEN4 CD, as they represent a good compromise between resolution, dynamic range and power consumption. Moreover, Prophesee event cameras were already successfully used to record automotive event data (N-CARS dataset, Sironi et al., 2018).

2.4.2 Event data representation

Processing the unconventional output of event cameras often requires the transformation of microsecond events into alternative representations.

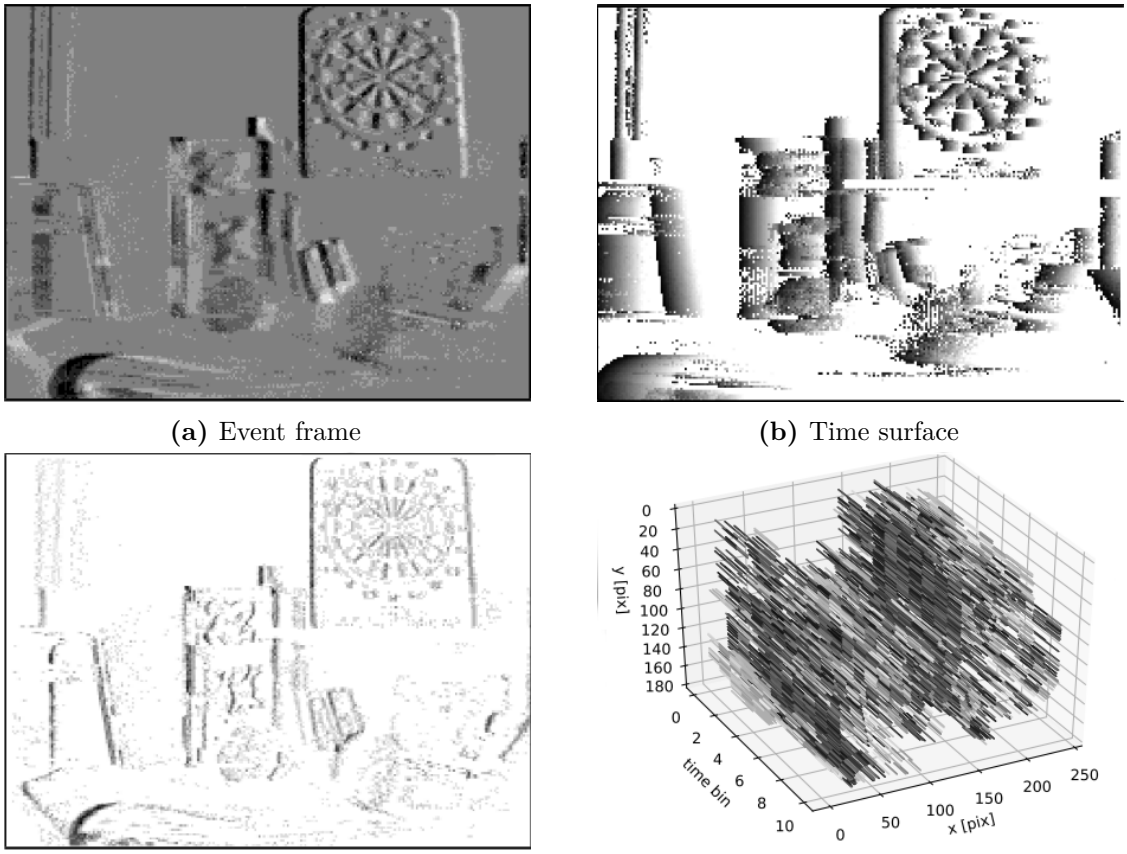
- The simplest way to use event data for supervised learning is to accumulate events pixel-wise over a period of time, either by counting them or by accumulating their polarities, essentially creating an event frame, also called a 2D histogram (Maqueda et al., 2018). Having reconstructed an image, algorithms designed for computer vision are able to work in spite of the differences between event frames and natural images.

- Time surfaces (Lagorce et al., 2017) are another way of representing events as a frame, except that here each pixel stores a single time value, representing the timestamp of the last event that occurred at those coordinates. Time surfaces are therefore images where the highest values correspond to a more recent motion. Since the pixels only keep the last timestamp, time surface highly compress information, resulting in a degraded performance when a high number of events are produced.

- Motion-compensated event frame, or image of warped events (IWE) (Gallego, Rebecq, and Scaramuzza, 2018), are a representation that aims to reconstruct an event image by realigning spatially group of events who are considered to be part of the same movement. The realignment is done by optimizing an objective function that measure the contrast of the histogram of warped events. The resulting motion-compensated images have sharp edges, providing a more intuitive representation of visual information than the events, which is in turn more effectively processed by classical computer vision algorithms.

- Regarding representations that attempt to preserve the temporal information of events, we can mention 3D point sets and voxel grids. 3D point sets (Sekikawa, Hara, and Saito, 2019) simply consider the temporal dimension of events as a spatial dimension, allowing their processing as a sparse 3D point cloud. Voxel grids are a space-time histogram of events, where each voxel represents a particular pixel and time interval. By avoiding the collapsing of events on a single 2D frame, voxel grids better preserve the temporal information of events. Events belonging to the same voxel are usually summed, even if other accumulations are possible. Voxel grids allow the arbitrary division of the sample duration in multiple timesteps, which makes it a particularly suitable representation for algorithms operating over time, such as spiking neural networks. We discuss voxel grids more in detail in Section 3.1.1.

All of these representations are represented in Fig. 2.12. In this thesis, as we will process event data with spiking neural networks, we will mainly use voxel grids and even propose a new event representation called *voxel cube* that will retain as much temporal information as voxel grids while having a smaller number of timesteps. Voxel cubes will be presented in Section 3.1.2.



(a) Event frame **(b)** Time surface

(c) Motion-compensated event image **(d)** Voxel grid

Figure 2.12: Several event representations. (a) Event frame, (b) Time surface with last timestamp per pixel (darker pixels indicate more recent time), (c) Motion-compensated event image (Gallego, Rebecq, and Scaramuzza, 2018), (d) Voxel grid on 10 time bins, dark and bright pixels indicate negative and positive events. Illustrations derived from (Gallego, Delbrück, et al., 2022).

2.4.3 Event processing methods

Processing event-by-event data is typically addressed by deterministic and probabilistic filters such as convolutions or Kalman filters. They have been successfully used for feature extraction (Brosch, Tschechne, and Neumann, 2015) or in SLAM systems (Weikersdorfer and Conradt, 2012). Filters have the advantages of handling asynchronous data in low latency, while aggregating information from multiple sources. But the best results for the event-by-event processing are achieved with ANNs and SNNs, trained on event data. Usually, an ANN is trained with the dense representations presented in Section 2.4.2, and is then converted to an SNN that processes the data event-by-event (Rueckauer, Lungu, et al., 2017).

The overwhelming majority of event processing methods need to process several events together to achieve sufficient performance. The different representations

for grouping events together have been presented in Section 2.4.2. Event frames are therefore commonly used as input for image-based learning methods such as Random Forests (H. Li, G. Li, and Shi, 2016), SVMs (Fleury, Vacher, and Noury, 2010) and DNNs (Maqueda et al., 2018). Time surfaces have been used in particular to train CNNs for computing optical flow in (A. Zhu et al., 2018). Methods working on voxel grids require in general more computations, but are able to provide better results thanks to the better preserved temporal information. In (A. Z. Zhu et al., 2019), voxel grids are used for the unsupervised learning of a DNN autoencoder for optical flow, depth and egomotion estimation; while (Rebecq et al., 2019) use voxel grids to train a deep CNN to generate real images from events.

The most performing processing methods today are based on deep neural networks processing dense representations of events, which therefore do not benefit from the main advantages of event cameras. Indeed, the high temporal resolution and low latency are most often destroyed by the event representation and the processing times. The low power consumption of event cameras is irrelevant if the processing with DNNs is preponderant. In such a case, the only benefit of event data remaining is the high dynamic range. But since the performances attained with event data are still inferior to those attained with natural frames, it appears difficult to see the point of using event data for computer vision applications.

In these conditions, we consider that spiking neural networks are the only processing methods capable of taking advantage of event data, as they are natively compatible with events and can be faster and more power efficient than their ANN counterpart. We will present in Section 3.2 a literature review focused on the usage of SNNs for the processing of event data.

2.5 Neuromorphic hardware architectures

The design of an end-to-end event based processing chain promises low latency and low power consumption. But implementing this chain on traditional Von Neumann hardware (CPUs and GPUs) does not provide these benefits.

Inspired by the mammal brain, researchers have designed *neuromorphic hardware*, massively parallel hardware accelerators for SNNs. Neuromorphic processors are composed of numerous interconnected physical computational units capable of processing spikes, offering better locality than standard architectures, therefore yielding low power and low latency. Neuromorphic hardware is still an emerging research topic, although commercial products are beginning to appear.

Neuromorphic architectures are generally divided into digital and mixed analog/digital implementations. Analog systems use the physical characteristics of electronic devices as part of the computation of the system, while digital systems rely on Boolean logic-based gates. As a consequence, analog systems operate asynchronously on continuous values, and the digital systems usually rely on discrete values synchronously, i.e. with a clock.

Although appealing, analog systems are significantly more noisy than digital systems, and far less mature. Mixed analog/digital neuromorphic hardware include SynSense DYNAP-SEL (Moradi et al., 2018) and Braindrop (Neckar et al., 2019). Braindrop is a research chip packing 4k neurons with analog computations and communications, resulting in an power consumption as low as 150 µW. SynSense's DYNAP-SEL features 1k analog neurons and up to 80k configurable synaptic connections, while its power consumption depends on the firing rate of the neurons, ranging from 200µW (0Hz) to 1mW (100Hz). The number of neurons still being small, these analog neuromorphic have few results to their credit, but they remain an promising research area to monitor in the coming years.

One of the first neuromorphic hardware was SpiNNaker (S. Furber et al., 2013), a full custom digital, massively parallel system composed of many small integer units (ARM cores). The intercommunication between the cores can handles a large number of very small messages (spikes), and the processing unit are software-defined, which allow the implementation of various neuron models at the cost of a lower hardware acceleration.

In 2015, IBM presented TrueNorth (Akopyan et al., 2015), a pioneering neuromorphic hardware ASIC. Each TrueNorth chip is composed of 1M digital neurons and 256M synapses distributed over 4096 cores. This heavily parallelized operation leads to an impressive power consumption of only 70mW.

Some years later, Intel unveiled Loihi (Davies et al., 2018), a neuromorphic research test chip composed of 128 cores capable of modeling 130k digital neurons and 130M synapses. Fabricated on Intel's 14nm process, the chip consumes only 110mW on a real keyword spotting use-case (Blouw et al., 2019), compared to 650mW consumed by the inference stick Intel Movidius. Loihi chips can be connected together to scale the number of neurons and synapses available, up to 100M neurons and 100 billion synapses with 768 chips. A second version entitled Loihi 2 has been released in 2021 (Orchard et al., 2021), providing 1M neurons per chip on a smaller surface (31mm^2 compared to 60mm^2) while being ten times faster and allowing programmable neurons and generalized spikes. The power consumption

is expected to remain the same. Finally, Intel also distinguishes itself from other neuromorphic chip manufacturers by offering a complete software framework named Lava for porting algorithms to Loihi, as we discussed in Section 2.3.4.

The first neuromorphic processor IP was commercialized in 2019 by Brainchip. Named Akida (Brainchip, 2020), it was presented as an hardware accelerator for any artificial intelligence tasks. They propose an entire toolchain to convert pretrained DNNs into SNNs with up to 4-bits activation spikes, reducing the loss of accuracy coming from the conversion. Brainchip pretended in a promotional video (Brainchip, 2021) that its Akida chip is able to run simultaneously three deep SNNs for three different tasks while not consuming more than 31mW, but the lack of peer reviewed papers using Akida makes the comparison with other neuromorphic architectures difficult.

A brief comparison of these architectures capabilities and power consumption is given in Table 2.3.

Table 2.3: Summary of the different neuromorphic hardware.

Neuromorphic hardware	**Type**	**# neurons**	**# synapses**	**Power cons.**
Braindrop	Mixed	4k	-	150µW
SynSense DYNAP-SEL	Mixed	1k	80k	200µW
SpiNNaker	Digital	1k	1M	1W
IBM TrueNorth	Digital	1M	256M	70mW
Intel Loihi	Digital	130k	130M	100mW
Intel Loihi 2	Digital	1M	120M	100mW
Brainchip Akida	Digital	1.2M	100B	20mW

These different neuromorphic hardware represent exciting and promising leads for the implementation of neuromorphic processing toolchains, and we hope that this thesis, by improving the processing of event data by SNNs, will lead to the implementation of real-world use-cases that take advantage of their benefits. In fact, we are working closely with the people developing SPLEAT (Abderrahmane et al., 2022), a neuromorphic architecture implemented on FPGA, so that the spiking neural networks developed in this thesis can run on low-power neuromorphic hardware.

2.6 Conclusion

In this section, we presented a comprehensive overview of the works conducted on spiking neural networks and event-based processing. Our literature review highlights

that spiking neural networks are as much a processing problem as an information coding problem. The data usually processed by digital systems is not directly compatible with the spike paradigm, which limited for a long time the development of SNNs. Recent advances in information encoding and training methods have open up new possibilities for spiking neural networks. In particular, the supervised training of SNNs with surrogate gradient now makes it possible to tackle more complex problems with deeper networks. In the meantime, the most popular spike encoding of data, rate coding, is showing its limitations: subpar performance and a high number of timesteps that results in high processing times and high power consumption, in contradiction with the promise of the SNNs.

On the other hand, event cameras have developed over the past ten years and are now mature and available commercially. Event data encounter the same problem of information encoding as SNNs, as few processing methods are capable of handling the asynchronous and sparse binary output of event cameras. The majority of the works was therefore focused on the representation of event data enabling the use of the most performing methods, which most of the time involves the destruction of its main benefits: sparsity, temporal resolution, binarity.

Finally, neuromorphic hardware have also experienced significant developments in recent years, and although it is still too early to consider them mature, they make a reality the promise of the SNNs to be low-power processing methods. These tangible results are a motivation to develop and improve existing spiking neural networks, because the energy gains provided will be proportional to the task difficulty, as the existing methods (e.g. deep neural networks) still have significant computational requirements.

In order to reach an efficient and performing end-to-end processing chain for embedded automotive applications, we have therefore focused in this thesis on the development of spiking neural networks to process event data, enabling their use on real-world automotive problems such as classification or object detection. Our contributions in terms of training, data representation and models have allowed us to reach new heights in performance, making the low-power use of SNNs and event cameras in real products, such as a car, a bit more realistic.

3

Classification on event data

Contents

Classification of unknown data is the most common problem addressed by neural networks. Their ability to learn features from high-dimensional data (e.g. images) using a large number of labels often makes them superior than other approaches. Due to their very nature, event data are more difficult to process than simple images, and they usually require tailor-made neural networks to achieve competitive results.

In this chapter, we explore how to tackle the classification of event data using spiking neural networks. Our main contributions are the introduction of sparse spiking convolutions to obtain highly sparse SNNs, and the design of very deep SNNs (more than 100 layers) using a new event representation, batch normalization layers and concatenation-based residual connections. Our networks reach new heights for SNNs, whether in terms of sparsity or accuracy.

This chapter is organized as follows: first, we discuss the different methods of encoding event data we used to feed them into neural networks. Second, we present the different models of spiking neural networks proposed for event data processing in the literature. Third, we describe how we tackled the classification of event data using sparse spiking convolutional neural networks. Finally, we present our work on the classification of automotive event data, a more complex task which served as a basis for our further work.

3.1 Event data representation

As we have already discussed in Section 2.4.2, multiple methods exist to transform the highly temporal and sparse event data in a representation more suitable for processing. We have also seen that SNNs are natively compatible with SNNs, as event can be interpreted as spikes. However, in order to process event data in modern deep learning models, we still need to convert them into a dense representation, as no satisfactory method allows to maintaining the extremely fine temporal resolution of event sensors (usually 1 microsecond). As stated in Section 2.4.2, the representation that least damages the nature of the event data is the *voxel grids*, as we can easily tune the number of timesteps and thus control the loss of temporal information.

We present in this section how we used voxel grids to represent event data for our SNNs, and we introduce a new representation called *voxel cubes* which aims to contain the same temporal information as the voxel grids while having fewer timesteps.

3.1.1 Voxel grids

As a reminder, a voxel grid is a representation where microsecond events are accumulated on larger time windows Δt seconds while keeping the spatial resolution unchanged. As such, each voxel represents a pixel and a time interval. For a recording of events of duration d seconds, we have thus gathered the events on $\frac{d}{\Delta t} = T$ timesteps. Usually, the events are stored in the form of a 4D $CTHW$ tensor, with C the number of channels, T the number of timesteps, H and W the

height and width of the data. We can then use spiking neural networks directly on event data, as they will now operate on T timesteps.

It is important to note that unlike other works, the voxel grids we use in this thesis are constructed with a binary accumulation on time windows: we do not sum the events nor save their precise timestamps, we only record if at least one event has been emitted in the time window Δt using an OR operation. This constraint leads to a loss of information but we consider this approach to be the most suitable for a real-time use case where the system would process the flow of events on the fly. With this choice, it is as if we had modified the temporal resolution of the event camera to be Δt seconds.

3.1.2 Voxel cubes

In order to keep the high temporal resolution of event data, we need to have a large number of timesteps T, which increases linearly the number of computations of the SNN and thus the inference time and the energy consumed.

Inspired by event cubes (Prophesee, 2022), we proposed in (Cordone, Miramond, and Thierion, 2022) a novel event representation called *voxel cubes* to mitigate this issue. In voxel cubes, each time window Δt is subdivided in n micro time bins lasting therefore $\frac{\Delta t}{n}$ seconds. Events belonging to a micro time bin will be stored in the channels dimension, providing finer temporal information to the first layer of the network than a voxel grid with the same number of timesteps. The number of channels C is now equals to $2 \times n$, each polarity being stored in n channels.

Contrary to event cubes, each event contributes only to the time bin where it falls into, and the accumulation of multiple events in the same micro time bin is binary. This loss of information is justified by the need to keep binary inputs, in order to leverage the energy efficiency of spiking neural networks running on specialized hardware, even if the latest neuromorphic hardware no longer seem to be limited by this constraint (see Akida and Loihi 2 in Section 2.5). An illustration of this representation is proposed in Fig. 3.1.

3.2 SNNs models for event data in the literature

In the last few years, spiking neural networks models have evolved dramatically. As the design of SNNs by conversion of pre-trained ANNs has shown its limitations, direct training of spiking neural networks has become increasingly common. Besides the various training methods developed, already discussed in Section 2.3, there has also been a significant effort on how to design spiking neural networks, whether

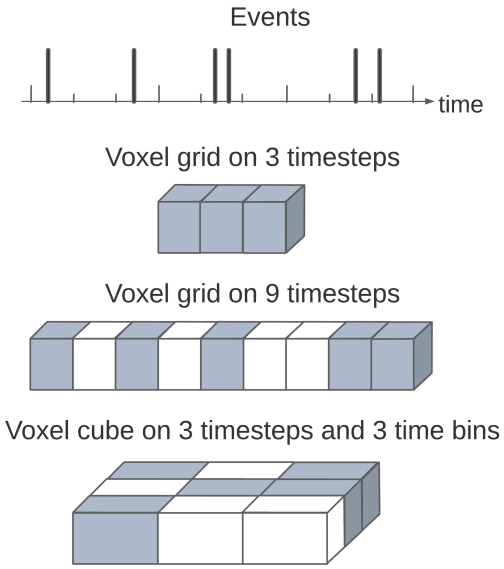


Figure 3.1: Binary voxel cube representation. Here, only one polarity is depicted. Voxel cubes exploit the channel dimension to preserve as much temporal information as a large number of timesteps would.

in terms of topologies or operations used. In this chapter, we review works that have designed SNNs to process event data, although most of the models presented here are also capable of processing static data.

3.2.1 Feedforward SNNs models

Spiking neural networks are built by replacing the non-linear activation functions of an ANN, generally a ReLU, by a neuron model. The neurons now acts as an activation layer, effectively generating spikes from input features. The first learning rules for SNNs were not performing well, it is therefore logical that the first SNNs have the same topologies as the first ANNs that appeared in the early 2000s. These SNNs are feedforward networks using convolution layers or fully-connected layers, with max pooling to reduce the size of the data throughout the network. At the end, a flatten operation or a global average pooling is applied before the final fully-connected layers. These networks rarely exceeds ten layers. This kind of models, with different learning rules, are for instance present in (Shrestha and Orchard, 2018) (8-layers CNN), (W. He et al., 2020) (2-layers MLP and 4-layers CNN), (Kaiser, Mostafa, and E. Neftci, 2020) (3-layers CNN) and (Fang, Yu, Y. Chen, Masquelier, et al., 2021) (6-layers CNN).

3.2.2 Recurrent SNNs models

Although SNNs can already be considered as an recurrent neural network, some works have tried to reproduce explicitly the recurrent operations used in ANNs with spike operations.

In particular, (Xing, Di Caterina, and Soraghan, 2020) proposed a spiking variant of a ConvLSTM cell named Spiking Convolutional Recurrent Neural Network cell (SCRNN cell), illustrated in Fig. 3.2. In this cell, input feature map and the state spiking feature map are combined to generate an output feature map and the next state spiking feature map. Their network is composed of three SCRNN cells and a final fully-connected layer. Unfortunately, the added complexity of this approach doesn't provide a gain in performance compared to previous works. Therefore, it seems that recreating a memory and recurrence mechanism in addition to the ones present in neuron potentials is not beneficial for classification tasks.

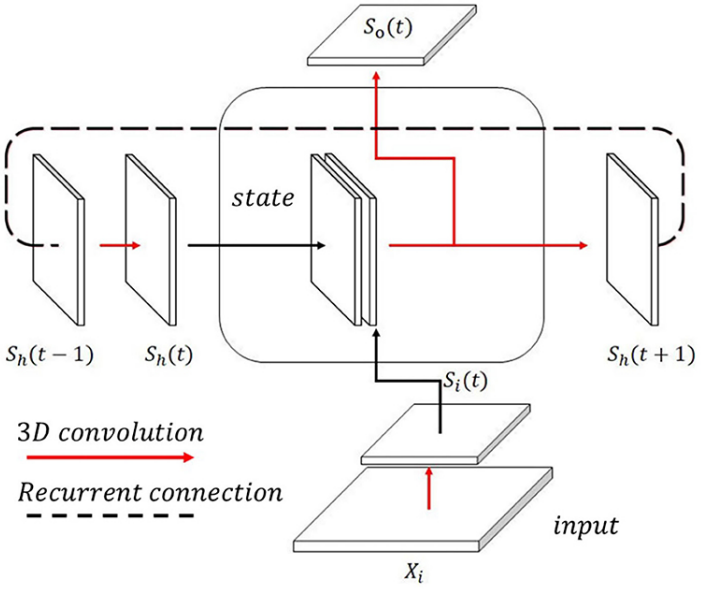


Figure 3.2: A single SCRNN cell proposed in (Xing, Di Caterina, and Soraghan, 2020).

3.2.3 Residual SNNs models

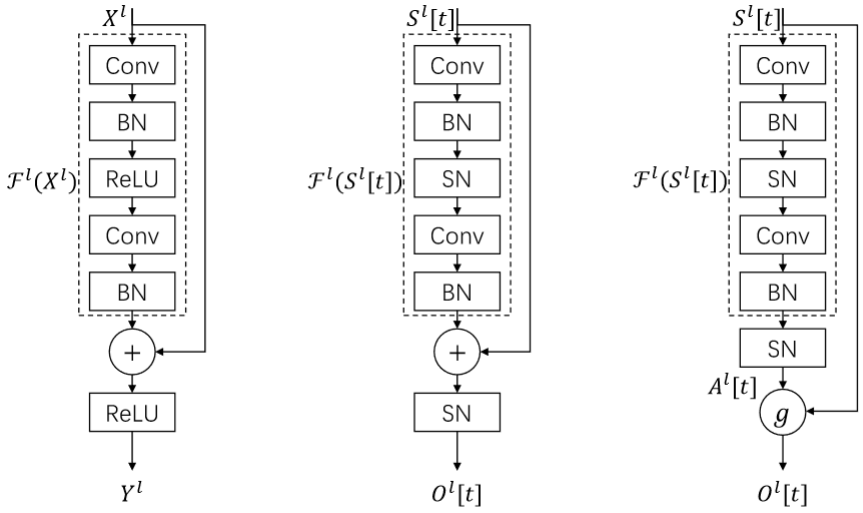
Advances in learning rules have recently made it possible to considerably increase the size of the spiking neural networks considered, especially the surrogate gradient learning rule detailed in Section 2.3.3. On top of that, the community followed the path outlined by the ANNs and sought to implement residual connections in SNNs. Residual connections solve the problem of vanishing gradient and allows an increase in the number of layers of classical neural networks, but their implementation in

SNNs are still an open research topic, as a simple addition between two spiking feature maps does not preserve their binarity.

Zheng et al. (2021) was the first to propose directly-trained spiking ResNets by introducing a pre-activation threshold dependent Batch Normalization (tdBN) operation. Using tdBN, they are able to design and train spiking ResNets with a very deep structure (50 layers), reaching high performance on both neuromorphic (96.87% on DVS Gesture) and static datasets (67.05% on ImageNet).

Another approach called Spike-Element-Wise ResNets was proposed in (Fang, Yu, Y. Chen, T. Huang, et al., 2021). The residuals blocks of a traditional ResNet, a basic Spiking ResNet and a SEW ResNet are illustrated in Fig. 3.3. A basic block in Spiking ResNet performs an addition between the output feature maps and the identity before the spiking activation function, while the SEW block implements the residual connection between the output spikes and the identity through a function g. Authors show that the basic block is not able to achieve identity mapping for all neuron models, and that it doesn't solve vanishing/exploding gradient. The SEW block solves these issues, using three different function g: ADD, AND and IAND. The three variants achieve different results, with the SEW ResNet using ADD performing significantly better: 69.26% on ImageNet with a SEW Resnet-152 ; 97.92% on DVS Gesture with a SEW ResNet-18 while the IAND and AND variants respectively reach 95.49% and 70.49% accuracy. The ADD variant performing better is not a surprise since the information transmitted between the layers is no longer binary. These discrepancies show that the the manner of making residual connections in spiking neural networks is still not clearly defined.

Based on these works, Na et al. (2022) proposed a spike-aware Neural Architecture Search (NAS) framework called AutoSNN. They noticed that the SNN architectures used in previous studies originated from conventional ANNs, and can be divided in two blocks: the VGG-styled stacked convolutional layers and max pooling layers, and ResNet-styled stacked residual blocks with skip connections. They standardized these building blocks, naming them Spiking Convolution Block (SCB) and Spiking Residual Block (SRB). They are both composed of two convolutional layers with spiking neurons, and the SRB additionally includes a skip connection before the spiking activation function (corresponding to diagram (b) in Fig. 3.3). They fixed the global architecture of the network and let their NAS algorithm choose between five candidate blocks: skip connection, SCB with kernel size of 3, SCB with kernel size of 5, SRB with kernel size of 3 and SRB with kernel size of 5. The best performing architectures are mostly composed of SRB with kernel size of 5.



(a) Basic block in ResNet (b) Basic block in Spiking ResNet (c) Spike-Element-Wise block

Figure 3.3: Residual blocks in ResNet, Spiking ResNet and SEW ResNet. Illustration from (Fang, Yu, Y. Chen, T. Huang, et al., 2021).

These results support once again the importance of having efficient residual connections in SNNs to obtain strong results.

Based on these different works, it seems clear to us that convolution blocks are fundamental to achieve good performance on computer vision tasks with SNNs, as it is the case for ANNs. Likewise, residual connections in SNNs often improve results, but the multitude of proposals has rather led us to incorporate in our networks residual connections based on concatenation, as we will discuss in Section 3.4. Sparsity, a feature shared by both SNNs and event data is yet notably absent from the literature. We rectify this oversight in the next section.

3.3 Event classification on IBM DVS128 Gesture

This section outlines the paper entitled *"Learning from Event Cameras with Sparse Spiking Convolutional Neural Networks"* (Cordone, Miramond, and Ferrante, 2021) that we presented at the International Joint Conference on Neural Networks in 2021. All of our code is available online [1].

In this work, we proposed a new method to train sparse spiking neural networks on event data. We then evaluated the performance of SNNs and CNNs with and without sparse operations in terms of accuracy, sparsity and training time on the IBM DVS128 Gesture event dataset. The proposed sparse SNN reaches higher

[1] https://github.com/loiccordone/sparse-spiking-neural-networks

accuracy while being more sparse, an important feature for the implementation of SNNs on low-power neuromorphic hardware.

3.3.1 Sparse SNNs and the timestep-wise operating mode

Sparse convolutions

A well studied method to speedup inference and minimize memory footprint of a neural network is to prune its weights, inducing sparsity in the model parameters. However, input data and activations remain dense. It also exists sparse convolutional networks that conserve the parameters dense but are able to process spatially sparse data, generating sparse tensors throughout the network. The immediate benefit of this approach is that the processing is done only on non-zero data, unlike dense ANNs, which is particularly interesting for highly sparse data such as the output of an event camera. Another benefit is that sparse convolutions (see Fig. 3.4) maintain the sparsity of the input data across layers, resulting in a sparse network, as we would see in Section 3.3.4. Sparse CNNs have been used primarily on LIDAR and depth cameras data, but some works have used them with event data. Messikommer et al. (2020) trained sparse CNNs on N-Caltech101 and N-CARS with similar accuracy as the dense network with up to 20 times less computations per event.

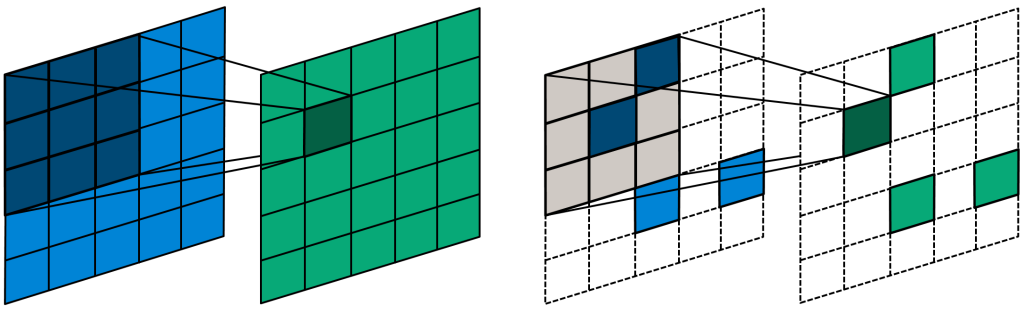


Figure 3.4: Visualization of a convolution operation on a dense tensor and on a sparse tensor. The number of operations and the results differ, as the convolution kernel only centers itself on non-null data. More details can be found in (Choy, Gwak, and Savarese, 2019).

Two PyTorch-based libraries offer sparse convolutions with a GPU acceleration: Facebook Research SparseConvNet (Graham, Engelcke, and Maaten, 2018) and NVIDIA Minkowski Engine (Choy, Gwak, and Savarese, 2019). A benchmark proposed by NVIDIA (*Benchmark - Minkowski Engine* 2022) shows that a single Minkowski Engine sparse convolution is at least twice as fast as the one in SparseConvNet, whether in the forward or in the backward pass. We therefore chose to use Minkowski Engine for the desing of our sparse spiking convolutional neural networks.

Sparse spiking convolutional neural networks

To construct a sparse spiking convolutional layer, we stack a sparse convolutional layer and spiking neurons. The sparse features output by the convolution update the neurons potentials, generating spikes at all locations where the potential has exceeded the firing threshold.

In terms of implementation, the membrane potentials are represented with a dense PyTorch tensor. Storing the membrane potentials as a sparse tensor is unfortunately very inefficient, as the addition between two sparse tensors is a very expensive operation. Indeed, since sparse tensors simply consist of an unordered list of positions and values, an addition between two sparse tensors require to iterate all positions of both positions to add their values, or to append the new position-value pair. As this addition occurs at each timestep in each layer, it severely impacts the training time.

Instead, we implement a partial solution to this problem: the sparse activations are converted to a dense representation before being added to the dense membrane potentials, and the spikes output are in the form of a sparse tensor. Therefore, our approach requires a sparse to dense conversion for the activations, and a dense to sparse conversion for the output spikes. While this limitation produces a reasonable overhead, it does not hinder the learning of the network while enabling the use of sparse convolutions in SNNs.

To the best of our knowledge, we are the first to present sparse spiking convolutional network that learns directly from spatio-temporal event data.

Timestep-wise operation

Another contribution of this work is the timestep-wise operation of our sparse SNNs, detailed in Algorithm 1.

The main difference with a traditional SNN operation (e.g. in Zimmer et al., 2019) comes from our timestep-wise approach: here the potentials are initialized and updated at the model level rather than inside each layer. This makes the traversal of the network possible timestep per timestep, and reduces the training time by a factor of L, the number of convolutional layers. Our method is therefore totally independent of the number of timesteps, since it only processes one timestep at a time.

At the time of this work, timestep-wise operation was not widespread in SNN frameworks, as the more common approach was to iterate through all the timesteps inside each layer (e.g. in Zimmer et al., 2019), an approach we call layer-wise. The

Algorithm 1 Timestep-wise spiking neural network algorithm

Inputs: number of timesteps T, input event data $X = \{X_0, ..., X_{T-1}\}$, number of convolutional layers L, spiking sparse convolutional layers $\{\texttt{sc}_0, \ldots, \texttt{sc}_{\texttt{L-1}}\}$, dropout layer `d`, linear layer `fc`

Outputs: prediction output Y

create layer potentials $\{mem_0, ..., mem_{L-1}\}$ initalized to 0, create an empty list $outs = \{\}$
for $t = 0$ to $T - 1$ **do**
  input X_t, mem_0 to $\texttt{sc}_0$, get out_t, mem_0
  for $i = 1$ to $L - 1$ **do**
    input out_t, mem_i to $\texttt{sc}_\texttt{i}$, get out_t, mem_i
  end for
  input out_t to `d`, get out_t
  input out_t to `fc`, get out_t
  append out to $outs$
end for
return Y the mean of $outs$

layer-wise operation forces each layer to wait until its preceding layer has gone through all timesteps, which impacts the training by backpropagation through time. Indeed, the unrolling of a layer-wise network make the first layers difficult to train due to the vanishing gradient problem, which is all the more true when the number of layers and the number of timesteps are high. In a timestep-wise approach, the unrolled network repeats the whole network as many times as there are timesteps, the weights of the first layers are therefore regularly reached and updated by the optimizer. It is important to note that nowadays, this timestep-wise approach has become common for training SNNs, and it is notably used by the main frameworks (e.g. SpikingJelly, SLAYER, Norse).

3.3.2 Models

In order to evaluate the performance and the efficiency of our sparse SNNs, we used four convolutional networks, noted A, B, C and D. We compared 3D convolutional neural networks with 2D spiking neural networks, with and without sparse operations. Sparse operations were implemented using Minkowski Engine (Choy, Gwak, and Savarese, 2019). The architectures are detailed in Table 3.1 and the architecture of our SNN for a single timestep is illustrated in Fig. 3.5.

The CNNs use 3D kernels of size $3 \times 3 \times 3$ while the SNNs use 2D kernels of size $k \times k$ with $k = 5$ for the first two convolutions and $k = 3$ for the last two. Every convolutional layer has a stride of 2×2 on the spatial dimensions. The time dimension is not strided and kept constant with padding for the CNNs.

Table 3.1: Architectures of the four models studied.

Model	Layer 1	Layer 2	Layer 3	Layer 4
A. CNN	`4c3-bn`	`8c3-bn`	`8c3-bn`	`16c3-bn`
B. Sparse CNN	`4sc3-bn`	`8sc3-bn`	`8sc3-bn`	`16sc3-bn`
C. SNN	`4c5`	`8c5`	`8c3`	`16c3-do`
D. Sparse SNN	`4sc5`	`8sc5`	`8sc3`	`16sc3-do`

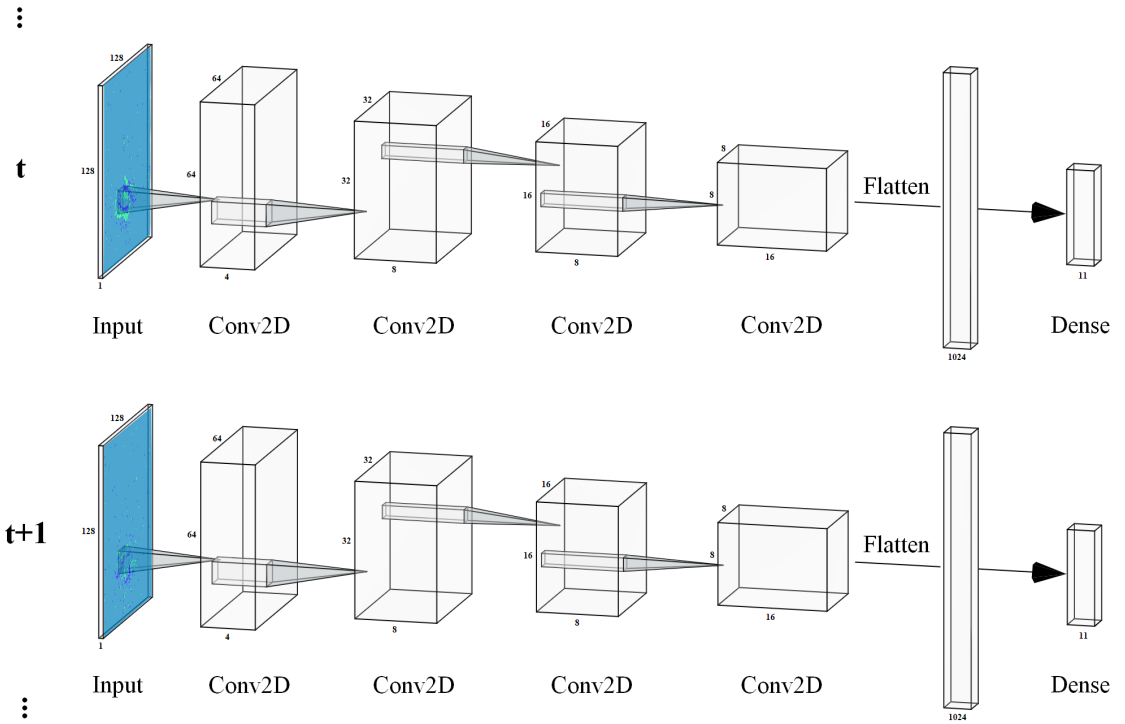


Figure 3.5: SNN architecture. Each timestep of the input event data goes through the network, updates the layers' potentials and outputs one prediction.

The notation `sc` denotes the use of a sparse convolution, `bn` denotes the use of a batch normalization layer and `do` denotes the use of a dropout layer with 0.5 probability. The activation functions for the CNNs and the SNNs are respectively ReLU and LIF neurons, implemented using Heaviside step function. Each network has a final fully-connected layer for classification.

3.3.3 Dataset

We evaluated our models on the IBM DVS128 Gesture Dataset (Amir et al., 2017), a dataset published in 2017 which contains recordings of 29 subjects performing 11 hand gestures under 3 different illuminations. The gestures were recorded using a DVS camera with a resolution of 128×128 pixels (see Fig. 3.6). Each gesture lasts

up to 6 seconds, and are provided as raw events with a microsecond timestamp, an x, y position and a polarity. Samples from the first 23 subjects are used for training and samples from the last 6 subjects for testing.

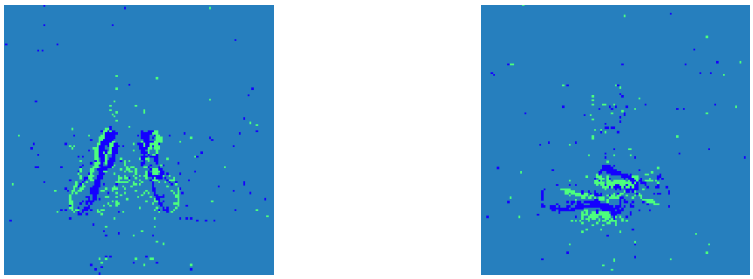


Figure 3.6: Examples of gestures from the DVS128 Gesture Dataset, where green and dark blue pixels correspond respectively to events that appear or disappear. This frame representation is obtained by the accumulation of events over a period of 20ms. The gestures depicted are a hand clap and an arm roll.

Following (Shrestha and Orchard, 2018), we provided only the first 1.5s of each gesture to our network for both training and testing, with raw events accumulated over $T = 150$ windows of $\Delta t = 10$ ms. These samples are composed of one input channel, the polarity, which is stored as either $+1$ or -1. The resulting voxel grid of shape $1 \times 150 \times 128 \times 128$ ($CTHW$) is stored as a sparse tensor for maximal efficiency. Apart from this change in representation, no pre-processing is applied to the events.

See Table 3.2 for a comparison of the temporal characteristics of our model with other SNNs from the litterature. Our method offers a good compromise between the duration of computations (less than 150 timesteps) and the respect of the data temporality (timestep of 10ms). In particular, (Fang, Yu, Y. Chen, Masquelier, et al., 2021) used a number of events to construct timesteps, which is not compatible with data coming from an event camera in real-time, and also prevents from taking advantage of data sparsity.

3.3.4 Results

The results of the training of our 4 models are presented in Table 3.3. The sparse SNN achieves the best test accuracy, beating the CNNs by 1.5%. It is interesting to note that CNNs with 3D kernels have obtained competitive results of around 90% test accuracy, in only 10 epochs of training. CNNs overfitted very rapidly, in spite of the use of high learning rate and weight decay, thus the batch normalization layers have been essential for proper learning.

Table 3.2: Comparison of the temporal characteristics with other SNNs.

Model	Timesteps train/test	Timestep duration	Sample duration train/test
SLAYER (2018)	300 / 300	5ms	1.5s / 1.5s
SCRNN (2020)	20 / 20	50ms	1.0s / 1.0s
DECOLLE (2020)	500 / 1800	1ms	0.5s / 1.8s
PLIF SNN (2021)	20 / 20	10k events/ts	400k events
This work	150 / <150	10ms	1.5s / <1.5s

Table 3.3: Test accuracy, epochs, and training time of our four SNNs.

Model	Accuracy	Epochs	Training time (per epoch)
A. CNN	90.28%	10	376s (38s)
B. Sparse CNN	90.63%	10	**228s (23s)**
C. SNN	90.28%	12	1232s (95s)
D. Sparse SNN	**92.01%**	31	2639s (85s)
SNN (2019)	87.50%	18	6332s (345s)

The SNN has approximately 2.5 times longer training time per epoch than the CNN, due to its sequential processing. While the CNN process the whole sequence in one go, the SNN has to process each timestep sequentially. The best accuracies were achieved by the SNNs, demonstrating that taking advantage of the temporal nature of the data leads to better results.

The sparse CNN and SNN both achieves better test accuracy than their dense counterparts. For the CNNs, the training time is even divided by 2, showing a great benefit of switching from dense operations to sparse operations. For the SNNs, training one epoch is faster for the sparse SNN, but the training requires more epochs leading to an overall longer training.

To validate our timestep-wise approach, we trained an additional SNN with the same architecture and hyperparameters as our model C but using the layer-wise approach used in (Zimmer et al., 2019). Our SNN accuracy was better and the training was around 4 times faster per epoch, which we believe comes from our implementation. Indeed, we hypothesize that since the training is performed on GPU, the timestep-wise operation allows an automatic merging of CUDA kernel calls for an entire network traversal, while in layer-wise this optimization is not possible.

We also measured the activity (i.e. the non-zero activations) during inference over 150 timesteps in each of the 4 convolutional layers, for each model. The results are presented in Table 3.4.

Table 3.4: Averaged non-zero activations (activity) after each layer during inference on the test set, over 150 timesteps.

Model	Conv 1	Conv 2	Conv 3	Conv 4	Total
A	643k (26%)	357k (29%)	108k (35%)	49k (31%)	1,157k
B	58k (2.4%)	83k (7%)	40k (13%)	38k (25%)	219k
C	**28k (1.1%)**	26k (2.1%)	14k (4.6%)	13k (8.5%)	81k
D	53k (2.1%)	**7.2k (0.6%)**	**5.5k (1.8%)**	**1.7k (1.1%)**	**67.4k**
SNN (2019)	38k (1.5%)	61k (4.9%)	17k (5.5%)	16k (10.4%)	132k

Even with a convenient ReLU activation function that generates less non-zero activations than other activation functions, the CNN has a sparsity of around 30% across the network, leading to nearly 1,157 non-zero activations. For a 99% sparse input data, this result shows that traditional convolutional layers damage the input sparsity, making the use of CNNs to process event data unattractive.

Using sparse convolutions inside the CNN results in a significant improvement in the network sparsity. The first layer is 10 times sparser, better preserving the sparsity of the event data. The following layers also show an improvement, resulting in a total of 219k non-zero activations, more than 5 times sparser than the classical CNN. This result alone justifies the advantage of using sparse convolutional networks for the processing of event data.

The SNNs achieve impressive sparsity: a maximum of 8.5% of the neurons spike at each layer, a number that even goes down to 0.6% for some layers. This results in a total number of spikes smaller than 81k, 3 times sparser than the sparse CNN and 15 times sparser than the CNN. The sparse SNN generates an even sparser network, with a reduction of 13k spikes compared to the dense SNN. Both models generate twice as sparse networks as a SNN using the algorithm presented in (Zimmer et al., 2019).

To the best of our knowledge, our method represents the state-of-the-art in matter of sparsity for the DVS128 Gesture dataset.

Table 3.5 compares our Sparse SNN with SOTA SNNs from the literature. When the number of parameters was unspecified, we assumed that the final output feature map was of size 4×4, probably underestimating the real number of parameters.

Table 3.5: Comparison with other SNNs.

Model	#params	Train iter.	Acc.	Real-time data	Simple operations
SLAYER (2018)	Unknown	270k	93.64%	✓	×
SCRNN (2020)	>662k	100 epochs	92.01%	✓	×
DECOLLE (2020)	>51k	160k	95.54%	✓	✓
PLIF SNN (2021)	>1,110k	74k	97.57%	×	✓
This work	14k	1.3k	92.01%	✓	✓

Although our model is not better in terms of accuracy, it is still competitive with up to 100 times less parameters and training iterations (i.e. a backpropagation pass). We reviewed if models are able to use event data coming directly coming from event camera with minimal preprocessing, representing a real time situation. Constructing frames by gathering a number of events, as done by (Fang, Yu, Y. Chen, Masquelier, et al., 2021), is not compatible with a real time situation where the number of events can vary greatly over time, and it even can prevent a prediction if the number of events is insufficient. Even if others SNNs are capable of using real-time event data, our model is the only one using binary event data. We also reviewed if the operations used in the models are compatible with a low-power neuromorphic implementation, i.e. if the operations are simple. Since our work only uses convolutions, multiplication and addition, it remains easily the simplest network to implement on neuromorphic hardware. Other works include the use of mean/max pooling, special memory cells... We could not compare sparsity with other SNNs since this work is the first to provide such a measurement.

3.3.5 Discussion on real-time inference

Because of its timestep by timestep functioning and the operations used, our method represents a plausible implementation of a low-power event-based approach. Indeed, spiking convolutions have successfully been implemented on low-power hardware (Abderrahmane et al., 2022), and our model is able to process a continuous stream of events directly coming from an event camera. As mentioned previously, our method is able to output a final prediction after any defined timestep, representing a realistic real-time inference.

We validate this claim by computing the test accuracy on DVS128 Gesture of our sparse SNN with different number of timesteps, ranging from 5 to 300. We are thus doing inference on samples lasting from 50ms to 3.0s. Results are presented in Fig. 3.7.

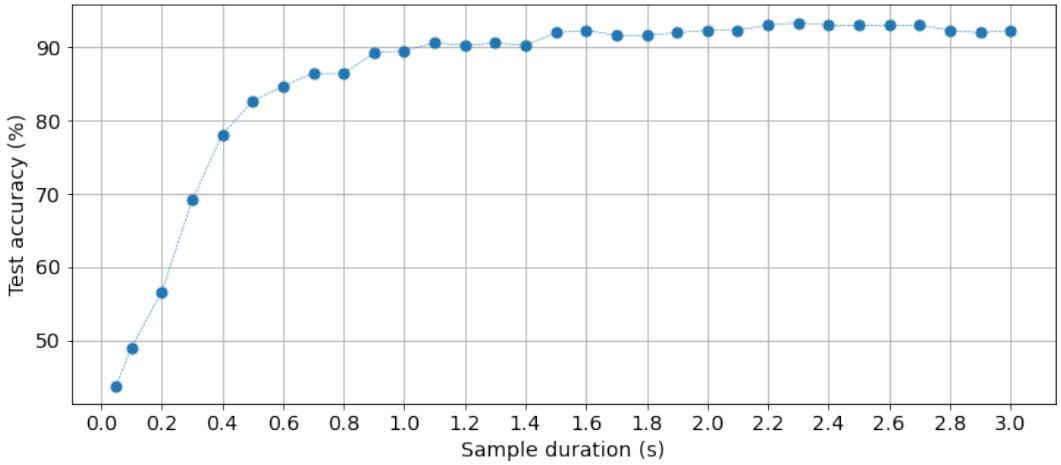


Figure 3.7: Test accuracy for samples of different duration. The network was trained with samples lasting 1.5 seconds, reaching 92.01% test accuracy.

The accuracy of our model is quite robust to the duration of the samples, as it achieves 86.46% test accuracy for 0.7s samples and even 82.64% for 0.5s samples. Accuracy drops rapidly when the number of timesteps is inferior to 50. Increasing the sample duration leads almost always to a better accuracy until the 1.5s mark, then the accuracy stays relatively constant. Our model even achieves 93.40% test accuracy for 2.3s samples. Therefore, it benefits from using more timesteps at inference than during training. Furthermore, as the number of timesteps is directly related to the number of operations performed, our work allows a trade-off between accuracy, power consumption, latency and memory.

This behavior is noteworthy for an embedded real-time inference: intermediate results can be obtained at the same rate as data is captured, and prediction can either be continued or be stopped according to the confidence, enabling even more energy savings without hindering the accuracy.

3.3.6 Conclusion

We presented a timestep-wise approach to build sparse spiking neural networks learning directly from binary event data. Our method generates highly sparse networks and is able to output a prediction at any timestep, two essential characteristics for a real-time inference on neuromorphic low power hardware. We validated our approach on the neuromorphic DVS128 Gesture dataset, achieving 93.40% test accuracy with a sparse spiking convolutional network, where the whole network does not generate more than 450 spikes per timestep.

But the target application in this work was still quite simple: the event sensor was still and in a controlled clean environment. It is not representative of real-word automotive event data captured by a moving event sensor in the wild. For this reason and in order to develop SNNs capable of processing more complex data, we then sought to tackle the automotive event datasets proposed by Prophesee.

3.4 Automotive event classification on the Prophesee datasets

This section outlines a part of the paper entitled *"Object Detection with Spiking Neural Networks for Automotive Event Data"* (Cordone, Miramond, and Thierion, 2022) that we presented at the International Joint Conference on Neural Networks in 2022. All of our code is available online [2].

Taking advantage of the latest advancements in matter of spike backpropagation - surrogate gradient learning, parametric LIF, SpikingJelly framework - and of our proposed voxel cube event encoding, we trained three different SNNs based on popular deep learning networks: VGG, MobileNet, and DenseNet. As a result, we managed to increase the size and the complexity of SNNs usually considered in the literature. We evaluated their performance on two automotive event datasets, establishing new state-of-the-art classification results for spiking neural networks.

3.4.1 Spiking VGG, Spiking MobileNet and Spiking DenseNet

Inspired by popular CNNs topologies, we designed and trained three different spiking neural networks using only strided convolutions, max pooling, batch normalization and PLIF neurons (Fang, Yu, Y. Chen, Masquelier, et al., 2021). Convolutions and max pooling have been used extensively with SNNs (Cordone, Miramond, and Ferrante, 2021, Xiao et al., 2020). Since batch normalization layers can be merged with a preceding or subsequent convolution layer at inference (see Section 5.1 for more details), it is possible to use them to train SNNs as long as they are placed before the PLIF neurons. We explore their importance in section 3.4.4.

In CNNs, the final layers used for classification need to be adapted to be compatible with spikes. To this end, we propose a spiking classifier simply composed of a layer of batch normalization, a 1×1 convolution outputting $num\_classes$ channels and PLIF neurons. By using a 1D convolution, this classifier is able to process feature maps of any size without requiring e.g. a layer of average

[2] https://github.com/loiccordone/object-detection-with-spiking-neural-networks

pooling, which would be incompatible with spikes operations. The final predictions are obtained by summing all output spikes first in the spatial dimension, then in the time dimension.

We propose spiking variants of VGG, MobileNet and DenseNet by replacing their ReLU activation functions by PLIF neurons. All spiking neural networks use the spiking classifier described above in lieu of their own classifier.

Spiking VGG

Introduced in (Simonyan and Zisserman, 2015), VGG is a convolutional neural network composed of up to 19 convolutional layers followed by 3 fully-connected layers. Our Spiking VGG replaced the final classifier but kept the same architecture, with the addition of batch normalization before each spiking convolutional layer.

Spiking MobileNet

MobileNet (Howard et al., 2017) is a model designed to be used in mobile applications that use depthwise separable convolutions, requiring less parameters and computations than normal convolutions. For our Spiking MobileNet, we dropped the activation function between the depthwise and pointwise convolutions, and moved all batch normalization layers before the convolutional layers. Removing the activation function makes our network non-spike as the inputs of the pointwise convolution are not spikes anymore. However, it can be shown that a depthwise separable convolution is equivalent to a normal convolution with specific weights. Thus, we used depthwise separable convolutions in the training of our spiking MobileNets as it provided better results (see section 3.4.4), and we return to a full-compatible SNN at inference by replacing them by their equivalent convolutions. Our Spiking MobileNet contains only one of the five identical depthwise separable layers near the end of the network. We have varied the number of filters of the first layer to obtain networks of different sizes.

Spiking DenseNet

To promote gradient propagation, ResNets use element-wise addition, an operation difficult to operate in the spike domain. Fang, Yu, Y. Chen, T. Huang, et al. (2021) proposed Spiking ResNets with different residual connections, but the one

based on an AND accumulation - in our opinion the only one compatible with an implementation on specialized hardware - produces unsatisfactory results. DenseNet (G. Huang et al., 2017) is an architecture that promotes gradient propagation by using channel-wise concatenations, which is an operation preserving the spike representation. We replaced the ReLU activations by PLIF neurons to obtain our Spiking DenseNet. We have varied the depth and growth rate (determining the number of channels) to obtain different versions of Spiking DenseNet.

3.4.2 Automotive event classification datasets

We evaluated our spiking neural networks on two automotive classification datasets: Prophesee N-CARS and a new event dataset we called *Prophesee GEN1 Automotive Classification*, generated from the object detection dataset Prophesee GEN1.

Prophesee N-CARS dataset

The Prophesee N-CARS dataset (Sironi et al., 2018) is a classification composed of 24k samples of length 100ms captured with a Prophesee GEN1 event camera mounted behind the windshield of a moving car. The samples represent either a car or background. Samples have variable size as they are cropped from recordings of resolution 304×240 pixels.

Prophesee GEN1 Classification dataset

We also generated a classification dataset from the Prophesee GEN1 Detection dataset (Tournemire et al., 2020) by cropping each bounding box (car or pedestrian) to form individual samples (see Fig. 3.8). As it is the case for N-CARS, each sample represents 100ms of events preceding the ground truth bounding box. They also have variable size since they are cropped from a 304×240 pixels resolution camera. The main difference between this dataset and N-CARS lies in the presence of a pedestrian class. Indeed, we believe that the features learned by a network are more relevant if it is trained to classify two different classes rather than one class vs. background. To avoid imbalance in the number of samples for each class, we rebalanced the training set: we undersampled the cars and oversampled the pedestrians by doing horizontal flip data augmentation. The code used to generate this classification dataset is available online (Cordone, 2022).

For both tasks, we used samples of 100ms encoded as binary voxel cubes of 5 timesteps and 2 micro time bins, as it represented the best compromise between performance and number of operations. The samples were resized to 64×64 pixels using nearest-neighbor interpolation to keep the input events binary.

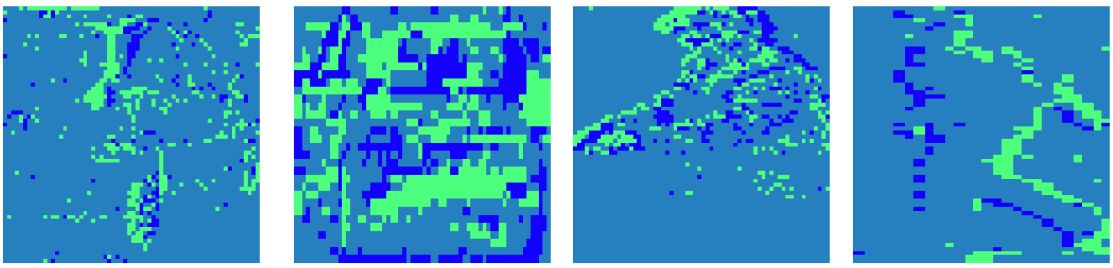


Figure 3.8: Examples of samples from the GEN1 Classification Dataset.

3.4.3 Results

The best accuracies obtained by our SNNs on the Prophesee N-CARS dataset are compared with other state-of-the-art models in Table 3.6. All of our models beat previous results for spiking neural networks and compete with the best neural networks in the literature. Our spiking DenseNets, MobileNets and VGG are all capable to exceed 90% test accuracy on N-CARS.

Table 3.6: Comparison with state-of-the-art models on Prophesee N-CARS.

Methods	Representation	Network	N-CARS acc
HATS (2018)	TimeSurface	N/A	0.902
Gabor-SNN (2018)	Spike	SNN	0.789
HybridSNN (2021)	VoxelGrid	SNN	0.77
HybridSNN (2021)	VoxelGrid	SNN-CNN	0.906
YOLE (2019)	VoxelGrid	CNN	0.927
Asynet (2020)	VoxelGrid	CNN	**0.944**
EvS-S (2021)	Graph	GNN	0.931
VGG-11 (ours)	VoxelCube	SNN	**0.933**
MobileNet-64 (ours)	VoxelCube	SNN	0.922
DenseNet121-24 (ours)	VoxelCube	SNN	0.904

But assessing the performance of SNNs is not limited to its accuracy, as multiple others features are needed to take advantage of their benefits when embedded in specialized hardware. We consider along the accuracy the following metrics:

- *Number of parameters:* embedded systems have high constraints in term of memory, therefore it is important to design networks with a low number of parameters.

- *ACCs:* spiking neural networks do not require multiplicative operations, enabling substantial energy savings on specialized hardware (see Section 2.5). Thus, we chose to report the number of operations of our SNNs by using the number of accumulations operations (ACCs), to accentuate the potential energy savings. Indeed, all spiking convolutions operations amount to ACCs, and each PLIF neuron only requires 1 ACC per timestep to update their potential. We did not count the ACCs in the batch normalization layers as they can be fused with the convolutional layers.

- *Activity:* finally, we measured the number of spikes emitted after each activation layer to represent the global activity of the network compared to a fully dense equivalent DNN. Indeed, processing events with SNNs preserves the data sparsity. On specialized hardware, computations are only performed when there are spikes, therefore an highly sparse network would consume less power than its dense counterpart. The activity is obtained by averaging the number of spikes divided by the number of activations over the whole test set. The *sparsity* is simply 1 minus the activity.

Table 3.7 provides extensive results of all our spiking neural networks on both automotive classification datasets. Once again, we can only regret that the works on SNNs in the literature do not systematically provide these metrics.

Our spiking VGG models provide the best accuracies for both datasets, while maintaining a relatively low number of ACCs per timestep. But these architectures have an high number of parameters, making it difficult to embed them. Spiking MobileNets reach high accuracies but require high numbers of parameters and ACCs per timestep, penalized by the replacement of their depthwise seperable convolutions by normal convolutions. However, they are the only models for which the accuracy increases as the model gets bigger. Finally, spiking DenseNets reach competitive accuracies while requiring a low number of parameters and a moderate amount of ACCs per timestep. Using the densely connected layers of DenseNets, surrogate gradient method has no trouble learning across 100+ spiking layers. The accuracy decreases however when the growth rate and the number of layers are both high, but we believe that better results could be achieved on these big networks with longer trainings.

For SNNs to be truly efficient, they require both low sparsity and a low number of timesteps. All of our spiking neural networks have an activity inferior to 40% on both datasets. As these SNNs operate on 5 timesteps, this means that they require at most twice the number of operations of an equivalent dense ANN. The operations

Table 3.7: Comparison between our spiking models on automotive event classification.

Models	#Params	ACCs/ts	N-CARS		GEN1 Classif.	
			Accuracy ↑	Activity ↓	Accuracy ↑	Activity ↓
VGG-11	9.23M	0.61G	**0.933**	12.04%	0.969	14.69%
VGG-13	9.41M	0.92G	0.910	14.53%	0.970	19.03%
VGG-16	14.72M	1.26G	0.905	14.91%	**0.977**	18.79%
MobileNet-16	1.18M	0.27G	0.842	17.57%	0.949	15.15%
MobileNet-32	7.41M	1.06G	0.902	18.53%	0.955	14.37%
MobileNet-64	18.81M	4.20G	**0.922**	17.14%	**0.966**	30.60%
DenseNet121-16	1.76M	1.01G	0.889	27.99%	0.970	20.31%
DenseNet169-16	3.16M	1.19G	0.893	30.12%	0.969	23.12%
DenseNet121-24	3.93M	2.25G	**0.904**	33.59%	**0.975**	27.26%
DenseNet169-24	7.05M	2.66G	0.879	34.02%	0.962	28.29%
DenseNet121-32	6.95M	3.98G	0.898	38.32%	0.966	29.46%
DenseNet169-32	12.48M	4.72G	0.825	37.48%	0.967	40.35%

would however consume less power on a specialized hardware, as they are simple ACCs (we would discuss this claim more in depth in Chapter 6). Furthermore, these sparsity results could be improved by adding a regularization term to the loss to constrain the number of spikes emitted, as it was done in (Zimmer et al., 2019).

3.4.4 Discussion

Influence of the number of timesteps and micro time bins

Spiking neural networks are recurrent neural network operating on a fixed number of timesteps T. Encoding event data in a representation that keep their temporal information can thus benefit SNNs. In section 3.1.2, we presented an event data encoding called voxel cubes that preserve the temporal information of events while minimizing the number of timesteps. Indeed, the number of computations performed by SNNs increases linearly with the number of timesteps, it is thus important to keep it small.

Fig. 3.9 plots the N-CARS accuracy of our SNNs on selected combinations of number of timesteps and micro time bins. The number of timesteps remains the most important parameter for obtaining good accuracies with SNNs. Indeed, better results are achieved when the number of timesteps increases, but only until a certain point as we achieved better results with 5 timesteps than with 10. Results obtained with 1 timestep are significantly worse, even with a high number of micro time bins, proving if necessary that SNNs need to operate on several timesteps to be performant. Increasing the number of micro time bins do not always improve the results, even if it seems to help when the number of timesteps is low.

Undoubtedly, these results depend on the samples duration and the data temporality. In our case, for samples lasting 100ms, the best compromise between number of timesteps, number of micro time bins and accuracy seems to be 5 timesteps and 2 micro time bins on N-CARS, which is the encoding format we used for all models.

Influence of Batch Normalization and PLIF neurons

Batch Normalization layers seem to be vital to the training of complex SNNs: as we can see in Table 3.8, removing them makes our networks significantly less accurate or they do not learn at all. More surprisingly, the placement of the batch normalization layers seems to also play a part in the efficiency of our networks, as placing batch normalization layers before the convolutions produce better results than placing them after. In DNNs, placing batch normalization layers before or

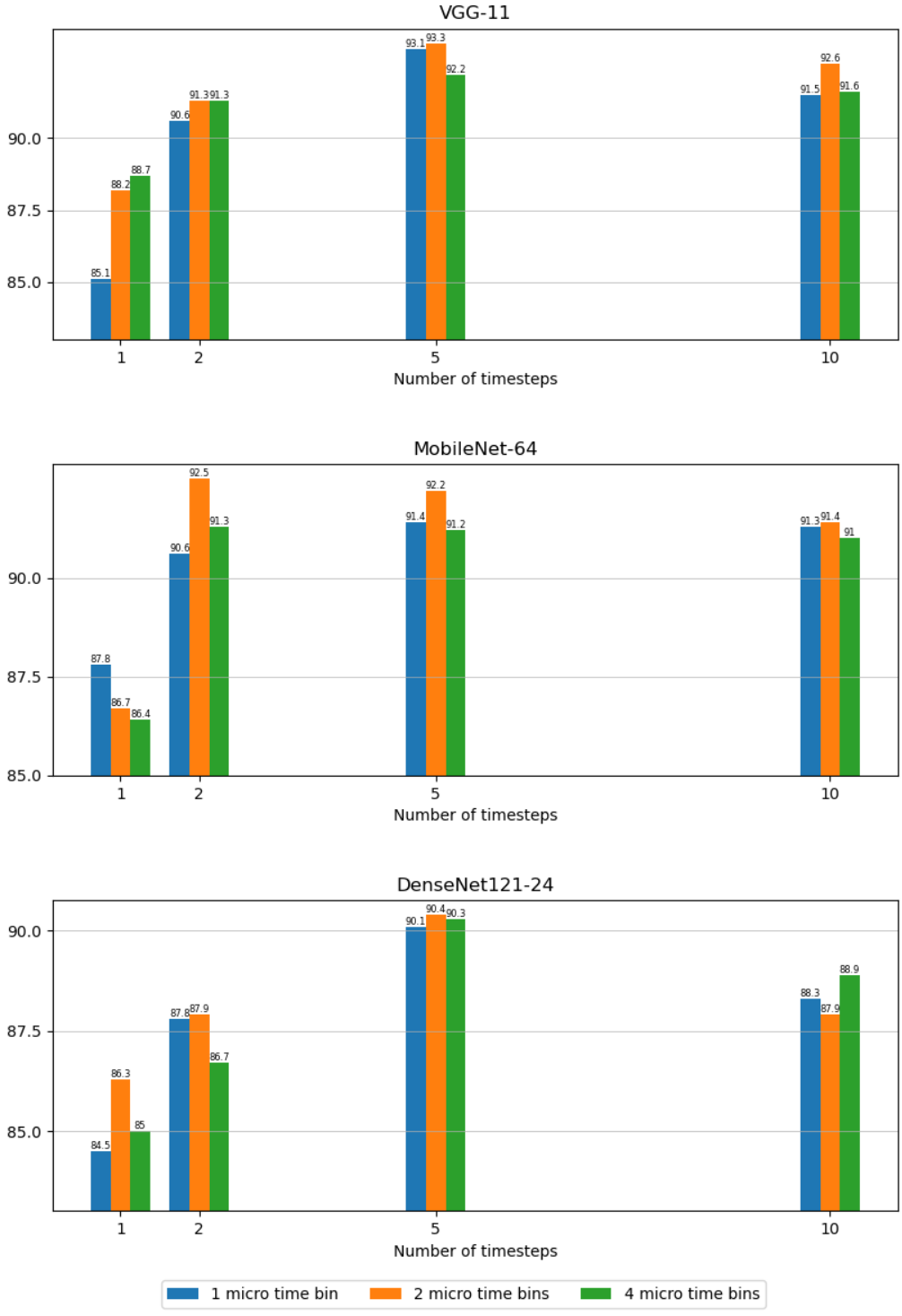


Figure 3.9: Influence of the number of timesteps and micro time bins on N-CARS.

Table 3.8: Influence of Batch Normalization when training SNNs on N-CARS (average over 3 runs).

Models	Accuracy ↑		
	BN-CONV order	CONV-BN order	No BN
VGG-11	**0.933**	0.819	0.649
MobileNet-64	**0.922**	0.812	0.736
DenseNet121-24	**0.904**	0.808	0.698

after convolutions do not make a significant difference, as both placements provide benefits in training speed and convergence.

In our case, we believe that batch normalization layers placed before convolutions are effective in SNNs because they transform highly sparse feature maps of spikes into a dense decimal representation. As a result, the weights learned by convolutions are updated through backpropagation whether they have received spikes or not. Without batch normalization, only the weights receiving spikes would have been updated meaningfully, leading to slower convergence.

We restate that batch normalization can be used when training SNNs because their parameters can be fused with the parameters of the subsequent convolutions. We will detail this procedure in Section 5.1. In light of this, we believe our findings on the placement of batch normalization layers in convolutional SNNs can make a difference in the training of larger and more complex SNNs.

On the other hand, PLIF neurons introduced in (Fang, Yu, Y. Chen, Masquelier, et al., 2021) also help during the training of our large spiking neural networks as seen in Table 3.9.

Table 3.9: Influence of PLIF neurons when training SNNs on N-CARS.

Models	Accuracy ↑	
	PLIF	LIF ($\tau = 2.0$)
VGG-11	**0.933**	0.889
MobileNet-64	**0.922**	0.801
DenseNet121-24	**0.904**	0.836

Replacing PLIF neurons with simple LIF neurons (with a time constant $\tau = 2.0$) leads to poorer accuracies for all networks. Their presence is not as important as batch normalization layers but the SNNs seem to benefit from learning different time constants for each layer. It also reduces the number of hyperparameters to be tuned, so we can only encourage their use for the training of SNNs on event data.

Influence of depthwise separable convolutions

We used depthwise separable convolutions during the training of our spiking MobileNets for two reasons: the smaller number of parameters made the trainings faster and the larger networks were able attain better accuracies. As we can see in Table 3.10, the 32 and 64 input channels variants of spiking MobileNets using depthwise separable convolutions reach higher accuracies than their normal convolution counterparts.

Table 3.10: Influence of depthwise separable convolutions when training spiking MobileNets on N-CARS.

Models	Accuracy ↑	
	Dw sp conv	**Normal conv**
MobileNet-16	0.842	**0.906**
MobileNet-32	**0.902**	0.898
MobileNet-64	**0.922**	0.807

We presume that the higher number of parameters induced by the normal convolutions makes the training of the spiking neural networks with surrogate gradient more difficult. As it is the case for the spiking VGGs, the accuracy actually drops when adding parameters to the normal convolutions MobileNets, while it increases for the MobileNets trained with depthwise separable convolutions.

However, the smaller variant seems to benefit from the added parameters as the accuracy is 5% higher with normal convolutions than with depthwise separable convolutions. We therefore recommend to use depthwise separable convolutions in SNNs only as a second resort, when the accuracy achieved with normal convolutions decreases as the networks get larger.

3.5 Conclusions and limits

In this chapter, we have outlined our steps to design SNNs directly trained on event data that are powerful, compact and sparse.

Using the surrogate gradient learning method, we first designed and trained a 4-layers convolutional SNNs on the DVS Gesture dataset by introducing sparse spiking convolutions, reaching competitive accuracy with a small, highly sparse network. Unfortunately, these sparse spiking convolutions suffer from a significant computational overhead, making it difficult to use them on deeper SNNs. In addition to this, we realized that sparse convolutions would require custom implementations

to be run on low-power neuromorphic hardware, an important obstacle to their democratization. For these reasons, we did not use sparse convolutions in the rest of our work.

Second, we designed and trained three different spiking neural networks models based on VGG, MobileNet and DenseNet, setting new state-of-the-art results on two automotive classification event datasets with SNNs. All our SNNs perform well without requiring an high number of timesteps thanks to our *voxel cube* event encoding. These SNNs are bigger and more complex than the ones usually considered in the literature, thanks to the use of batch normalization layers, concatenation-based recurrent connections and parametric LIF.

In both works, we evaluated our performance using not only the accuracy but also the size of the network (numbers of layers, number of parameters), the activity of the network (whether in percent or in number of spikes) and the number of operations (number of ACCs). We can only recommend that future works on SNNs systematically provide these measures, as they are vital to justify the spiking approach.

Results obtained in this chapter prove that spiking neural networks are now able to reach competitive performance on event data, comparable with classical neural networks. SNNs promise lower computational cost, yet they operate over multiple timesteps. One of the benefit of using surrogate gradient learning for training SNNs is that it provides us with the ability to control the number of timesteps, and therefore, we are now able to train SNNs on very small numbers of timesteps, while having the same or better performance than converted SNNs that operate on hundreds of timesteps. Our proposed voxel cube encoding even retains a little more temporal information for a given number of timesteps, slightly improving performance without adding computational overhead.

The main focus of this chapter was to develop complex SNNs architectures that are able to learn complex patterns on large event datasets, since it is a necessary step in order to tackle more complex and real-world problems than classification. Using most of our findings as a foundation, we study in the next chapter the use of SNNs to perform object detection on event data.

4

Object Detection on event data

Contents

Although classification problems are of interest, embedded automotive applications require more complexity. In a car, one of the most critical functions is danger detection. It requires low latency, reliability and must run in permanence. For these reasons, the process of event data with spiking neural networks represents a promising approach. We address this as an object detection problem, i.e. determining where objects are located in a given image (object localization) and which class each object belongs to (object recognition). Object detection on event data is still a largely undeveloped area, and even more so if their processing is done with spiking neural networks.

In this chapter, we describe the design of the first spiking neural networks capable of performing object detection on automotive real-world event data. We will start by presenting the different frameworks used in the literature to solve the task of object detection with classical neural networks. Then, we will interest ourselves in the works that tackled the object detection task with spiking neural networks. Finally, we will present our method to do object detection on event data using spiking neural networks, and what still limits us.

4.1 Neural networks for object detection

Neural networks tackling the object detection task can be described as composed of two parts: a backbone, extracting features from the input image, and an object detector, classifying and localizing objects based on these features. In this section, we present the most important neural networks detectors proposed in the literature. Detectors can divided in those that operate in two stages, and their successors that require only a single stage.

4.1.1 Two-stage detectors

A two-stage detector is a network that has a separate module to produce region proposals. In the first stage, these models attempt to locate an arbitrary number of objects proposals in an image, and in the second stage, they classify and locate those objects. These systems typically are not real-time, have complex architectures, and lack global context because they involve two distinct steps. They do, however, provide a better understanding of the steps that led to the design of their successors, the single-stage detectors.

R-CNN and Faster R-CNN

The Region-based Convolutional Neural Network (R-CNN) (Girshick et al., 2014) demonstrated how a CNN could immensely improve object detection performance. First, a region proposal module produce 2000 object candidates using Selective Search (Uijlings et al., 2013). These candidates are then warped and propagated through a CNN backbone to obtain a high dimensional feature vector for each proposal. They use trained Support Vector Machines (SVMs) to obtain confidence scores on all proposals, and they apply Non-Maximum Suppression (NMS) to keep the most relevant scored regions. Once the class had been identified, the model then predicts its bounding box using a trained bounding-box regressor. This operation is illustrated in the Fig. 4.1.

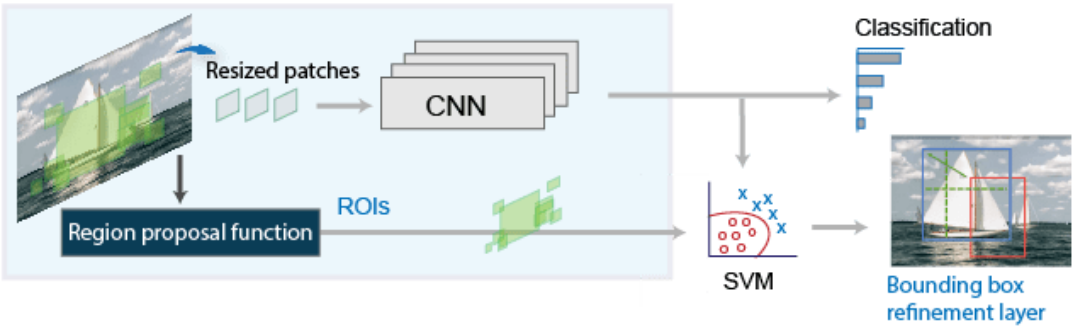


Figure 4.1: Architecture of R-CNN (Girshick et al., 2014).

R-CNN was a pioneer in the field of object detection, but it was slow (47 seconds per image in 2013, due to the repeated traversals of CNNs on overlapping candidate regions) and expensive to train. Thereafter, Ren et al. (2015) proposed Faster R-CNN, where the region proposal module is implemented by a dedicated CNN called Region Proposal Network (RPN). This RPN now outputs a set of candidate windows, along an *objectness score* which determines the likelihood of an object presence. RPN introduces the notion of *anchor boxes*, multiple manually defined bounding boxes of different aspect ratios on which the network performs a regression to position the predicted bounding boxes. The use of neural networks at every stage of the object detection allows Faster R-CNN (see Fig. 4.2) to increase the performance and to operate in near real time (5 frames per second).

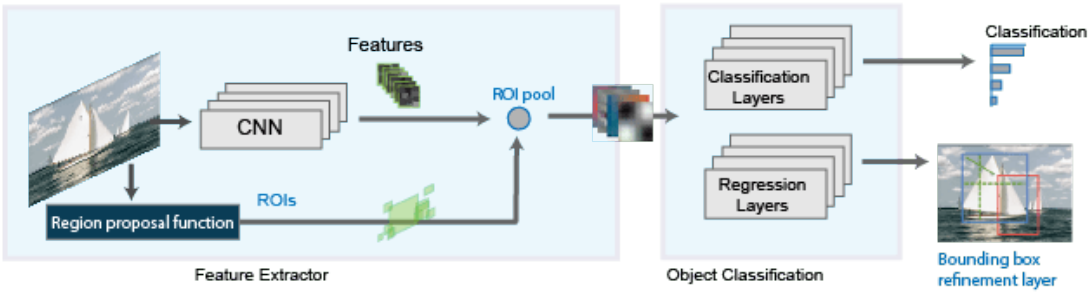


Figure 4.2: Architecture of Faster R-CNN (Ren et al., 2015).

R-FCN

Region-based Fully Convolutional Network (Dai et al., 2016) uses a combination of four convolutional neural networks to detect objects. First, a CNN backbone extract features from the input image. An intermediate feature map is input to a Region Proposal Network to identify the Region of Interest (RoI) proposals while the final feature map is fed to a classifier and a bounding box regressor.

This work brought us closer to a single neural network learning end-to-end to tackle an object detection problem, what we present in the following subsection as one-stage detectors.

4.1.2 One-stage detectors

Single-stage detectors classify and locate objects in a single shot. They employ predefined boxes and keypoints of various scales and aspect ratio to localize objects. Although they initially performed worse than the two-stage detectors, they are now superior in every respect: they outperform them in real-time performance and have a simpler design. These are the reasons of the widely usage of one-stage detectors in embedded systems.

YOLO

While two-stage detectors address the object detection as a classification task, either an object or background, YOLO or You Only Look Once (Redmon et al., 2016) reframed it as a regression problem, directly predicting the image pixels as objects and its bounding box attributes. In YOLO (see Fig. 4.3), the input image is divided into a grid of cells, and each cell is responsible for predicting the objects' center. Thus, one grid cell predicts multiple bounding boxes.

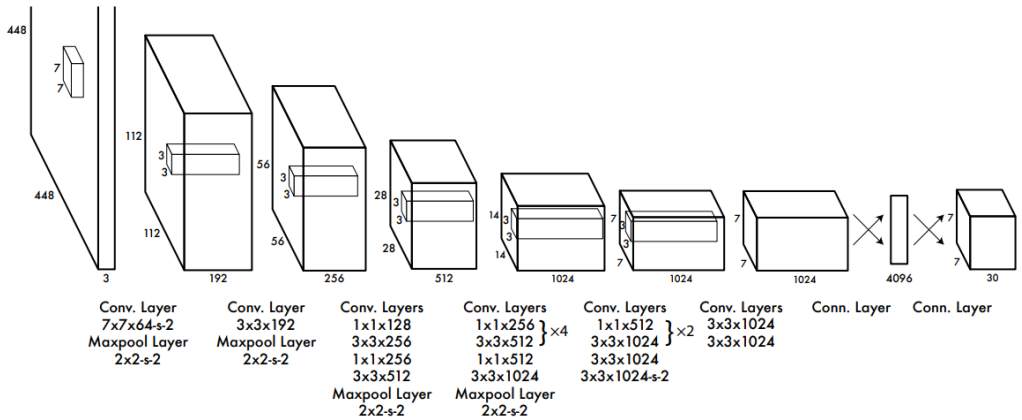


Figure 4.3: Architecture of You Only Look Once (YOLO) Redmon et al., 2016.

At the time of its publication, YOLO outperformed its one-stage real-time competitors by a wide margin in terms of accuracy and speed. It did, however, have serious shortcomings: the restriction on the number of objects allowed in each cell and poor localization accuracy for small or clustered objects. The subsequent versions of YOLO addressed these issues.

SSD

Single Shot MultiBox Detector (SSD) (W. Liu et al., 2016) use a CNN backbone combined with auxiliary convolution layers added at the end of the model. Its architecture is illustrated in Fig. 4.4. SSD recognizes smaller objects early in the network using the large feature maps, while the deeper layers are in charge of the offset of the anchor boxes and aspect ratios. During training, SSD match each ground truth box to the anchor box with the best jaccard overlap (see Fig. 4.5) and train the network accordingly. As SSD outputs thousands of predictions, the vast majority is composed of background predictions. To help the training, SSD proposed *hard negative mining* to compute the loss: instead of using all the wrong predictions, they take only those with the highest confidence scores, so that they only keep a maximum ratio of 3:1 between bad predictions of background and foreground objects.

At the time, SSD was much faster and more accurate than state-of-the-art networks such as YOLO and Faster R-CNN, but it has trouble detecting small objects. This this was somewhat mitigated afterwards by using better backbones, e.g. ResNet, but the detection of small objects remains the main limitation of SSD.

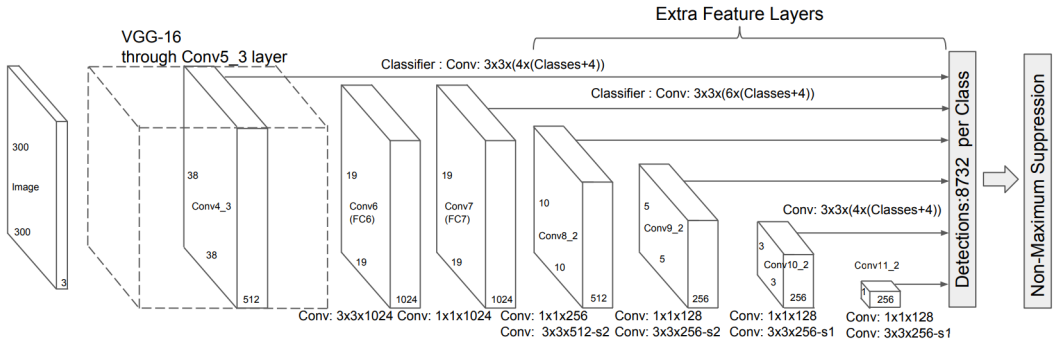


Figure 4.4: Architecture of Single-Shot Multibox Detector (SSD) (W. Liu et al., 2016).

RetinaNet

To explain the difference of accuracies between single and two-stage detectors, Lin, Goyal, et al. (2017) suggested that the "extreme foreground-background class imbalance" is impairing the performance of single-stage detectors.

Based on the cross entropy loss, they proposed a new loss called *focal loss* (see Fig. 4.6) to correct the imbalance. Object detection models output thousands of predictions, with only tens corresponding to foreground objects. As such, the overwhelming majority of predictions are background. With classical cross-entropy loss, even the correctly classified ($p_t \gg 0.5$) background predictions

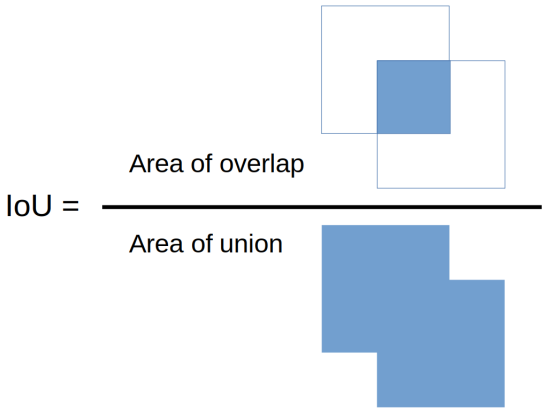


Figure 4.5: Intersection over Union (IoU), or jaccard overlap, is calculated by dividing the intersection of two elements by their union, as a way to measure their similarity.

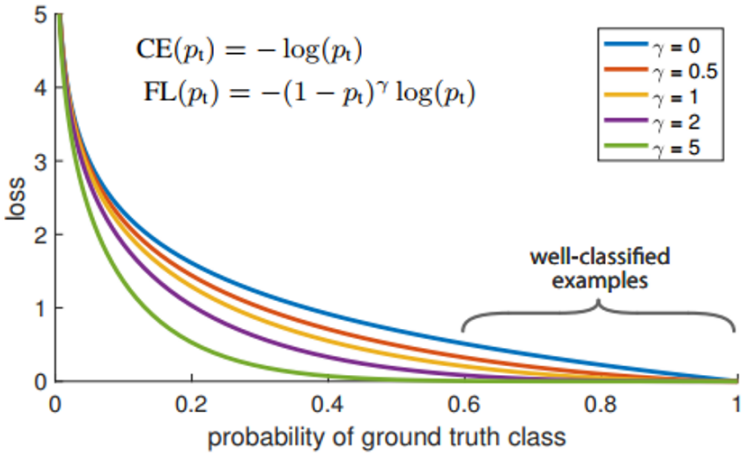


Figure 4.6: Focal Loss as introduced in (Lin, Goyal, et al., 2017), with p_t the probability predicted by the model for a sample for a given class. Cross-Entropy loss is represented by the blue line ($\gamma = 0$).

are still contributing to global loss, mitigating the contribution to the loss of a wrongly predicted object foreground. On the other hand, focal loss reduces the loss contribution of these simple samples while strongly penalizing the non-localized foreground objects.

Using this loss, they proposed RetinaNet (see Fig. 4.7), an object detector using ResNet and a Feature Pyramid Network (FPN) (Lin, Dollár, et al., 2017) as backbones and two identical subnets as classification and bounding box regressor. Each layer from the backbone is passed to the regressors, enabling the detection of objects at various scales. RetinaNet is simple to train, converges fast and is easy to implement, while reaching better performance in accuracy and run time than all two stage detectors, YOLO and SSD. RetinaNet performance comes from

the use of focal loss and from the pyramidal architecture, which significantly helps in the detection of small objects.

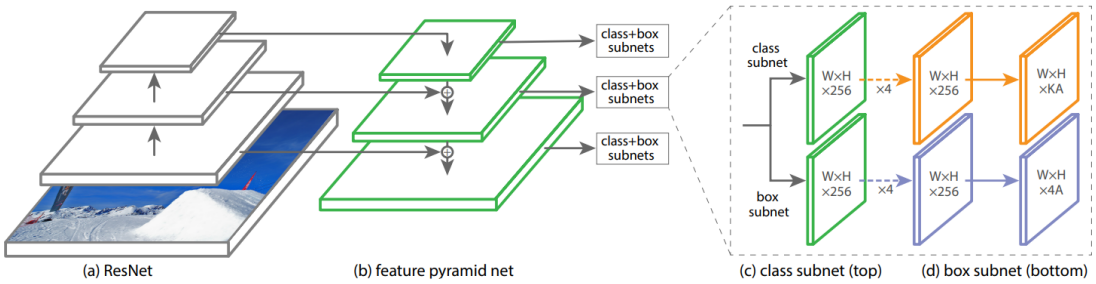


Figure 4.7: Architecture of RetinaNet (Lin, Goyal, et al., 2017).

A summary of the performance of these one-stage detectors is presented in Table 4.1. While RetinaNet has clearly the best results, but it features residual connections throughout the network, a characteristic that we found difficult to incorporate into a full SNN as we already discussed in Section 3.2.3. For its part, YOLO does not easily allow the use of another backbone, in addition to having a very large number of parameters. For these reasons, we will base our work on object detection with SNNs on SSD, while also using the focal loss introduced by RetinaNet to improve the results.

Table 4.1: Summary of the presented one-stage detectors. Values taken from (Zaidi et al., 2022).

Model	Year	Backbone	$AP_{0.5:0.95}$	$AP_{0.5}$	FPS
YOLO	2015	GoogLeNet	-	-	45
SSD	2016	VGG-16	23.20%	41.20%	46
YOLOv2	2016	DarkNet-19	21.60%	44.00%	81
RetinaNet	2018	ResNet-101-FPN	31.90%	49.50%	12
YOLOv3	2018	DarkNet-53	28.20%	51.50%	45

4.2 Object detection with spiking neural networks

While the object detection task is largely addressed by classical neural networks, spiking neural networks capable of doing object detection are still rare. In this section, we review the few attempts of using SNNs for object detection.

4.2.1 Spiking-YOLO: a converted SNN applied to frames

In an effort to tackle more complex problems than the classification of simple datasets (e.g. MNIST and CIFAR) with shallow SNNs, S. Kim et al. (2020) presented a first spiked-based object detection model called Spiking-YOLO. It is important to note that they design their spiking neural network by converting an already trained equivalent classical neural network.

Authors state that previous works attempting to design SNNs detectors using conventional DNN-to-SNN conversion methods suffer from severe performance degradation and are unable to detect any objects. They outline two possible reasons for this performance degradation after conversion: an extremely low firing rate in numerous neurons and the lack of an efficient implementation of Leaky-ReLU in SNNs.

By using channel-wise normalization, they managed to obtain higher firing rates, leading to accurate information transmission in a short period of time. Indeed, the majority of neurons generated a firing rate of up to 80% when using channel-norm, while using traditional layer-norm results in most of the neurons generating a firing rate between 0% and 3.5%. These values can be compared with the activity defined in Section 3.4.3, but since here the network is designed by conversion, the number of timesteps is radically different, leading to a vastly different number of spikes emitted.

To convert Leaky-ReLU in SNNs, they propose a signed neuron featuring imbalanced threshold (IBT). IBT neurons can interpret both positive and negative activations, and accurately and efficiently compensate for the leakage term in the negative regions of Leaky-ReLU. The proposed method also retains the discrete characteristics of the spikes by introducing a different threshold voltage for the negative region.

To validate their contributions, they converted a real-time object detection model, Tiny-YOLO, a variation of the YOLO model presented in Section 4.1.2 that is faster and more lightweight, at the cost of a lower accuracy. Using both of their contributions, they manage to reach 98% of the performance of the non-spike Tiny-YOLO on the frame datasets PASCAL VOC (Everingham et al., 2010) and MS COCO (Lin, Maire, et al., 2014), illustrated in Fig. 4.8.

While impressive, these results highlight the major issue with the design of SNNs by conversion: a very high number of timesteps. Indeed, as seen in Fig. 4.9, the SNN requires more than 1,000 timesteps to have comparable performances with the equivalent ANN, and the astonishing number of 8,000 timesteps to reach the 98% performance of the non-spike network.

Figure 4.8: Examples of samples from the PASCAL VOC 2012 (left) and MS COCO 2017 (right), the two most popular frame-based object detection datasets. PASCAL VOC contains 20 different classes, and MS COCO 91 classes.

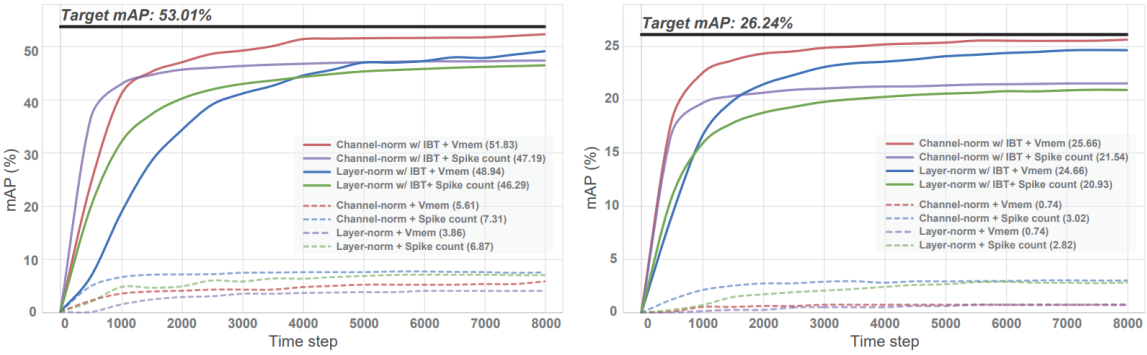


Figure 4.9: Experimental results of Spiking-YOLO on PASCAL VOC (left) and MS COCO (right) as provided in the original paper (S. Kim et al., 2020). Networks that do not use IBT neurons are represented in dotted lines, proving their importance.

For us, this number of timesteps is unsustainable for a real implementation of spiking neural networks on neuromorphic hardware. Moreover, their work target images, which do not take into account the temporal component of the SNNs as event data would. It is for these two reasons that we find more relevant the use of directly trained SNNs for object detection on event data.

4.2.2 Directly trained SNNs

We present here two works tackling object detection with spiking neural networks directly trained with supervised learning rules presented in Section 2.3.3.

DCSNN

Barchid, Mennesson, and Djéraba (2021) introduced a Deep Spiking CNN (DSCNN) to perform object localization of one object in grayscale images, as a first step towards a functional object detection solution. As they are working on images, they

need to encode the pixels as spikes. They used rate coding, which, as we discussed multiple times, produces a large number of spikes over a large number of timesteps.

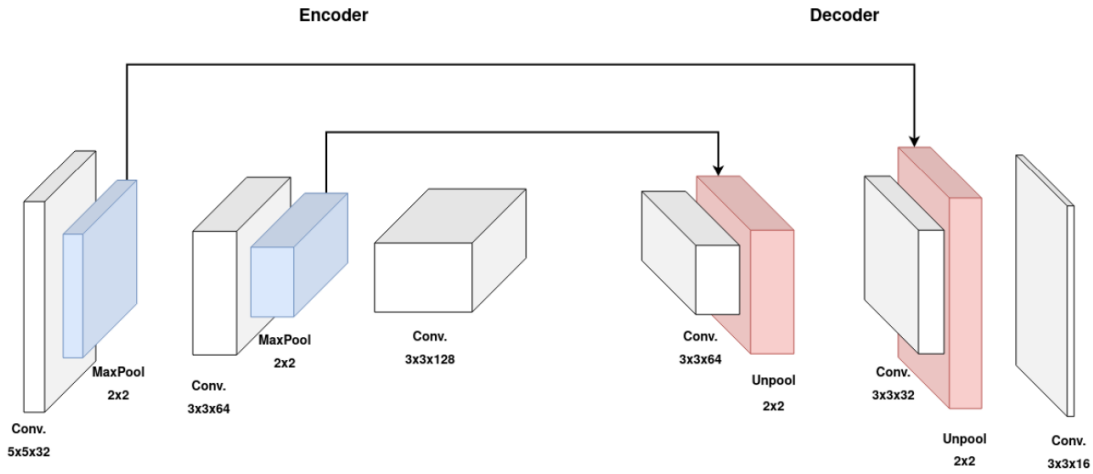


Figure 4.10: Architecture of DSCNN (Barchid, Mennesson, and Djéraba, 2021).

The architecture of DSCNN is illustrated in Fig. 4.10. Following an encoder-decoder structure, the network is composed of 6 convolutional layers, using additive residual connections. They evaluated their network on the Oxford-IIIT-Pet dataset, a classification dataset. They generated a single bounding box for each sample using the segmentation information provided in the dataset. Their network attain 63.2% mean Intersection over Union (IoU) between their predictions and the ground truth, and has a hard time localizing pets in the background as the dataset is essentially composed of close-up pictures.

This work claims to have proven the ability of SNNs trained with state-of-the-art supervised algorithms to deal with modern computer vision tasks. We do not share their opinion though, as we consider the tackled problem too trivial for being considered object detection. Nonetheless, it outlines the different requirements for the training of SNNs for object detection: an efficient spike encoding for the input, a complex network architecture, a performing supervised learning rule and a large labeled dataset.

Hybrid SNN-ANN

Recently, Kugele et al. (2021) presented Hybrid SNN-ANN (see Fig. 4.11), a network composed of a SNN backbone and an ANN head for classification or object detection task on event data. Using surrogate gradient learning, they train their hybrid network in one pass, using a different head depending on the task.

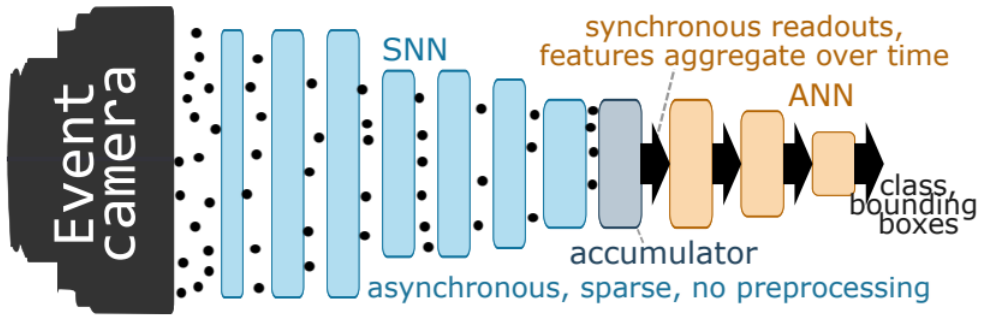


Figure 4.11: General architecture of Hybrid SNN-ANN (Kugele et al., 2021).

They used three different SNN backbones based on VGG, DenseNet and DenseSep, a variant of DenseNet that uses depthwise separable convolutions. For object detection, they combined their SNN with an ANN head composed of four convolutions, all connected to SSD heads. They evaluated their 3 networks on `shapes_translation`, a dataset representing 1356 images of 10 different shapes pinned to a blank wall (see Fig. 4.12). They used the provided event version of the images and manually labelled bounding boxes around each shape. They reached very high mAP (around 0.875 $\text{mAP}_{0.5}$) with the three networks, with the DenseNet backbone providing the best results.

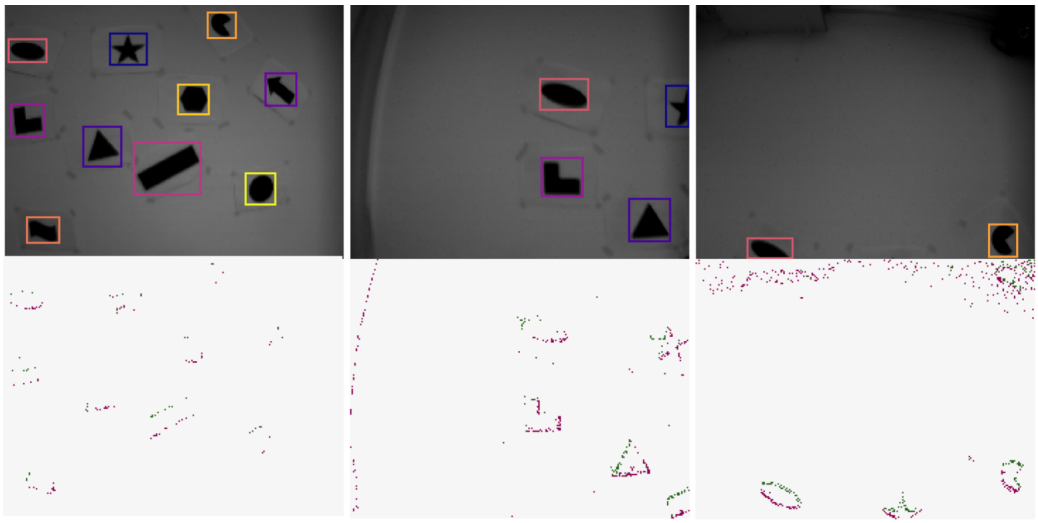


Figure 4.12: Samples from the `shapes_translation` dataset used in (Kugele et al., 2021).

This work showed that directly training SNNs on event data for object detection is possible, but they had to use ANN heads to reach these high results. Moreover, the dataset used for object detection seems very simple, as only the shapes are generating events since the event camera is fixed.

In the next section, we prove that directly training SNNs on event data for object detection on a more challenging, real-world dataset is possible without any ANN part.

4.3 Object Detection with SNNs on automotive event data

This section outlines a part of the paper entitled *"Object Detection with Spiking Neural Networks for Automotive Event Data"* (Cordone, Miramond, and Thierion, 2022) that we presented at the International Joint Conference on Neural Networks in 2022. All of our code is available online [1].

As presented in Section 3.4, we took advantage of the latest advancements in matter of spike backpropagation - surrogate gradient learning, parametric LIF, SpikingJelly framework - and of our proposed voxel cube event encoding to train three different SNNs based on popular deep learning networks: VGG, MobileNet, and DenseNet. We combined these SNNs with SSD to propose the first spiking neural networks capable of performing object detection on the complex GEN1 Automotive Detection event dataset.

4.3.1 Combining our SNN backbones with SSD

As presented in Section 4.1.2, the SSD object detection framework (W. Liu et al., 2016) consists of a backbone and multiple predictor heads. The heads take as inputs feature maps generated by the backbone at different scales to predict bounding boxes and their associated classes. To obtain a complete spiking neural network capable of doing object detection, we replaced the CNN backbone by the SNN backbones designed for classification that we presented in Section 3.4. We used spiking convolutions instead of normal convolutions in the extra layers. Therefore, the feature maps fed to the SSD heads are spikes, and since each head consists only of one convolution the whole network is indeed a SNN.

The spiking neural network operates on T timesteps, therefore the final bounding boxes and classes predicted by the network are obtained by doing a sum over the T timesteps. The output of the network still requires a post-processing to filter predictions, but we assume that this step would be done on conventional hardware outside the SNN scope.

One-shot object detectors such as SSD struggle with the class imbalance problem due to the overwhelming number of predictions classified as background. Lin, Goyal,

[1] https://github.com/loiccordone/object-detection-with-spiking-neural-networks

et al. (2017) introduced focal loss, a modulation term applied to the cross-entropy loss function that tremendously helps the learning of one-shot object detectors. Thus, we trained our spiking neural networks with focal loss, as the hard negative mining originally used by SSD did not achieved satisfying performances.

As in the original SSD architecture, we used three extra blocks of convolutions to generate smaller feature maps after our spiking backbone. Each block consists of a 1×1 spiking convolution to reduce the number of channels, followed by a 3×3 spiking convolution with a stride of 2. Once again, we used batch normalization layers before each convolution and Parametric LIF neurons.

The anchors used by our network were generated with a minimum ratio of 0.05 and a maximum ratio of 0.8, accounting for the smaller objects present in the object detection dataset we studied.

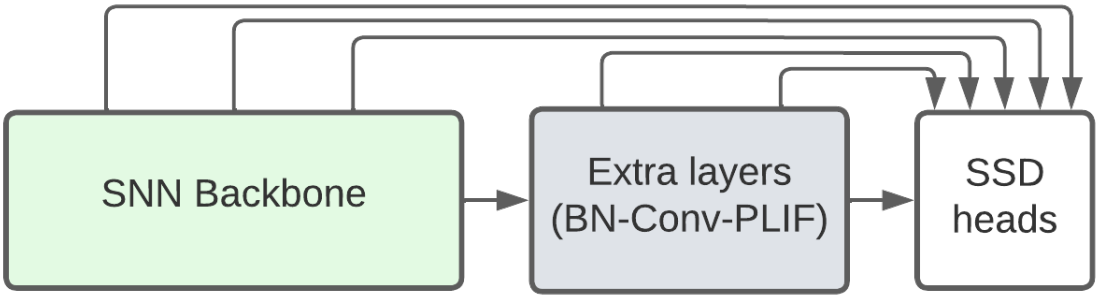


Figure 4.13: General architecture of our SNNs for object detection.

The general architecture of our spiking object detectors is shown in Fig. 4.13. Increasingly smaller feature maps generated by the SNN backbone are fed to the SSD heads, and the 3 extra blocks further reduce features maps size. We used spiking VGG, spiking MobileNet and spiking DenseNet as SNN backbones. The size of the feature maps transmitted to the SSD heads differ for each backbone, as shown in Table 4.2.

For the VGG-11 backbone, we pass on the feature maps after each one of the Max pooling layer. For Mobilenet-64, the feature maps are transmitted after the 4th, the 6th and the last depthwise separable convolution block. Finally, for the DenseNet-121, we feed to the SSD heads the feature maps after the two first transition blocks, and the final dense block. For all networks, the three extra layers use 640, 640 and 320 channels. As we can see, there are some difference between the size of feature maps output and the original SSD model, for the simple reason that they performed better. A more detailed visualization of the DenseNet architecture that also highlights the temporal operation mode of our networks is shown on Fig. 4.14.

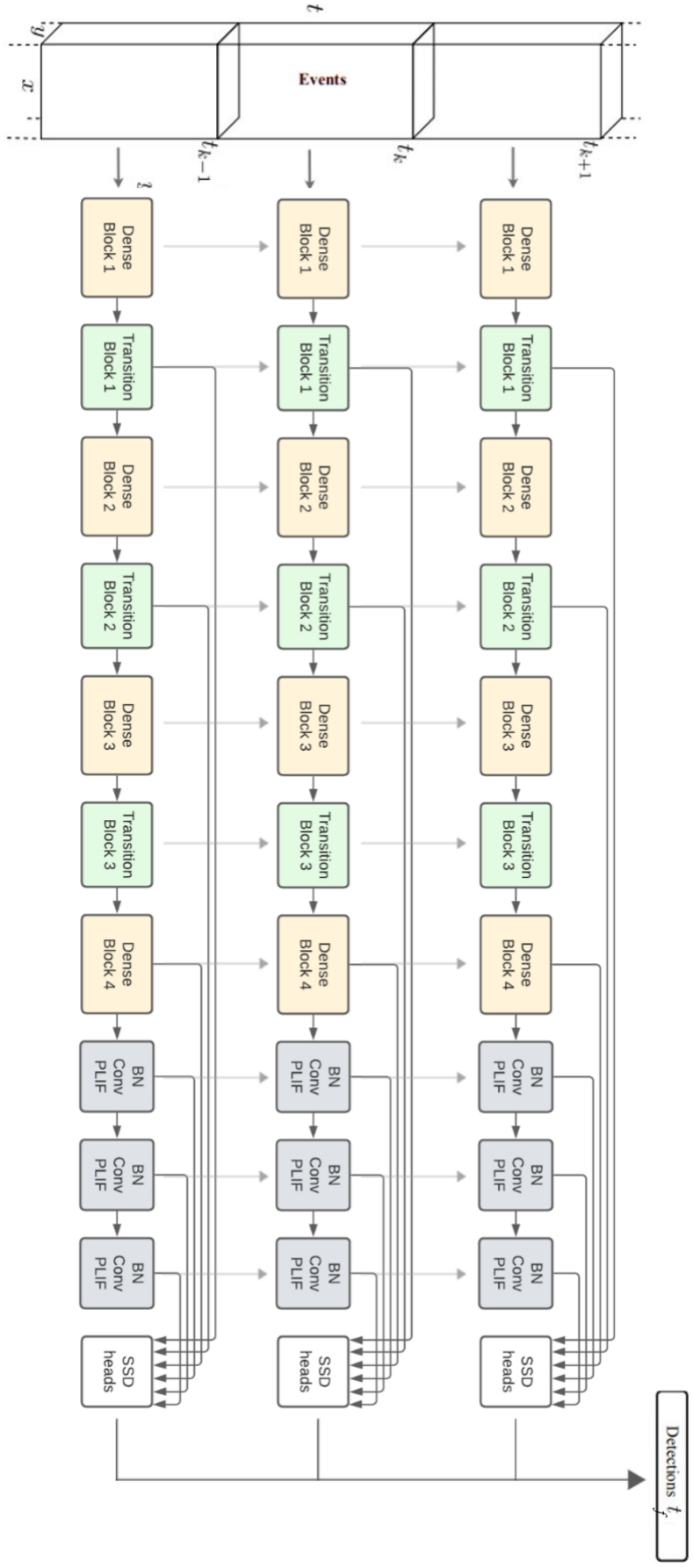


Figure 4.14: Detailed architecture of our spiking DenseNet + SSD, highlighting the temporal operating mode of our SNNs.

Table 4.2: Size (NxHxW) of feature maps transmitted by each of our spiking model to the SSD heads. A comparison with the original SSD model is also provided.

Models	VGG-11	MobileNet-64	DenseNet121	Original SSD
	256x60x76	256x30x38	192x60x76	512x38x38
Size of	512x30x38	512x15x19	384x30x38	1024x19x19
feature maps	512x7x9	1024x8x10	768x7x9	512x10x10*
transmitted	640x4x5*	640x4x5*	640x4x5*	256x5x5*
to SSD heads	320x2x3*	320x2x3*	320x2x3*	256x3x3*
	320x1x2*	320x1x2*	320x1x2*	256x1x1*

* denotes the feature maps produced by extra layers.

4.3.2 Object detection event dataset: Prophesee GEN1

Composed of 39 hours of recording, the Prophesee GEN1 Automotive Detection dataset Sironi et al., 2018 is the largest event-based dataset to date. Recorded with a Prophesee GEN1 sensor mounted on a car dashboard, it contains over 255k manually annotated bounding boxes of two classes: cars and pedestrians.

We chose to train our network only on the 100ms preceding annotated bounding boxes. Thus, our SNN makes predictions based solely on these 100ms, the membrane potentials being reset after each sample. The samples were encoded as binary voxel cubes (see Section 3.1.2) of 5 timesteps and 2 micro time bins, as it represented the best compromise between performance and number of operations on the classification tasks.

4.3.3 Results

Our object detection models used an initial learning rate of $1e^{-3}$ and were trained with a batch size of 64 over 200 epochs, using a cosine annealing learning rate scheduler that gradually decrease the learning rate towards 0. All convolutions were initialized using the Kaiming uniform method, and all batch normalization layers were initialized with a weight of 1 and a bias of 0. The Parametric LIF neurons all had an initial membrane time constant τ of 2, a membrane threshold of 1 and the ATan function as the surrogate function. Norm of the gradient values were clipped to a maximum of 1 to avoid exploding gradients.

All trainings were done with the SpikingJelly framework (Fang, Y. Chen, et al., 2020) using 16-bit automatic mixed precision, running on a 48-GB NVidia RTX A6000 and a 104-threads Intel Xeon Gold 6230R.

Our models used as backbones the best variants of our SNNs on the NCARS classification task: VGG-11, MobileNet-64, DenseNet121-24. We evaluated them on the Prophesee GEN1 Detection test set after having filtered boxes with diagonal smaller than 30 pixels as it is done in (Perot et al., 2020). The spiking backbones were pre-trained on the NCARS dataset, the extra layers and SSD head were trained from scratch. Our results are presented in Table 4.3.

Table 4.3: Comparison with state-of-the-art models on Prophesee GEN1. Our SNNs operate on 5 timesteps.

Methods	**Type**	**#Params**	**ACCs/ts**	**Activity ↓**	**mAP ↑**
Asynet (2020)	CNN	133M	-	-	0.15
MatrixLSTM (2020)	CNN	65M	-	-	0.31
RED (2020)	CNN	24M	-	-	**0.40**
VGG-11+SDD	SNN	12.64M	11.07G	22.22%	0.174
MobileNet-64+SSD	SNN	24.26M	4.34G	29.44%	0.147
DenseNet121-24+SSD	SNN	8.2M	2.33G	37.20%	**0.189**

Our spiking models achieve competitive mAP with a small number of parameters and ACCs per timestep. We reach 0.19 COCO mAP with our DenseNet121-24 + SSD model, with only 8.2M parameters and 2.33G ACCs per timestep. Our spiking models outperform a traditional neural network with over 5 times more parameters. The three models show relatively similar performance, proving that spiking backbones are able to provide meaningful spike feature maps to do object detection on real-world event data.

Some qualitative results are illustrated in Fig. 4.15. A short video demonstration is also available online [2]. As we can see, the cars are accurately localized, as long as they are not in the background. Even fast moving cars, which represent a problem for traditional cameras, are correctly identified. Our models have more trouble with pedestrians, as they often miss them or not as accurately localize them. It could comes from the samples imbalance in the Prophesee GEN1 dataset, as there are four times more cars than pedestrians in labels.

4.4 Conclusions and limits

We presented the first spiking neural networks capable of doing object detection on the real-world event dataset Prophesee GEN1, achieving 0.19mAP with less than

[2] https://www.youtube.com/watch?v=1Wcqem6u91Y

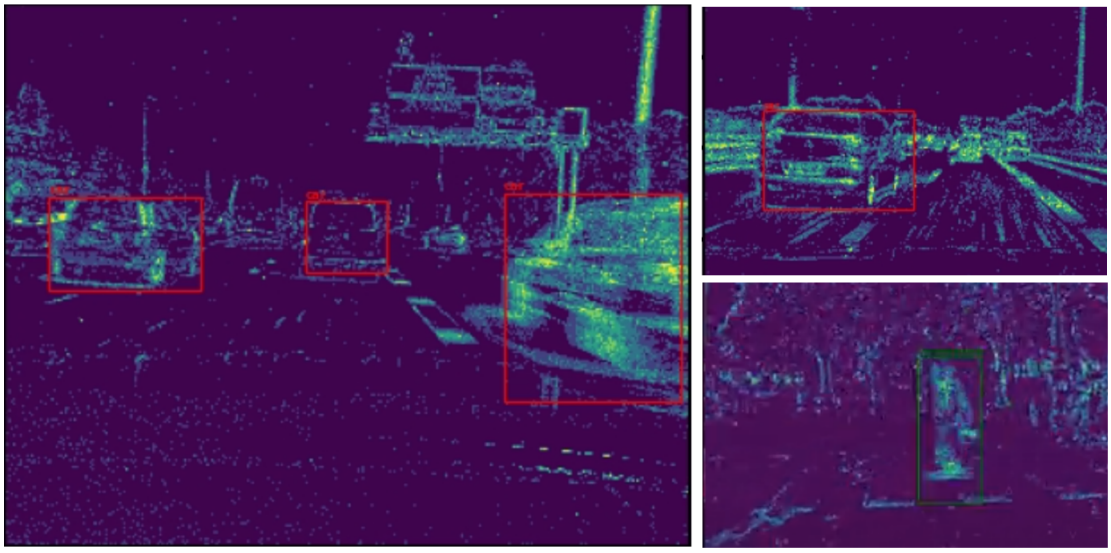


Figure 4.15: Qualitative results of our DenseNet + SDD model. Cars are detected in red and pedestrians in green.

10M parameters. All our SNNs perform well with a low number of timesteps thanks to our voxel cube event encoding. These results highlight the rapid progression of spiking neural networks in the last few years: for a long time restricted to small datasets, spiking neural networks now show their strengths when trained directly on temporal data.

But a significant performance gap still exists between our SNN approach and the literature. We expose below the reasons that explain why we are not as performing, and what we still need to improve to be competitive with classical neural networks on real-world automotive datasets.

Prophesee network RED (Perot et al., 2020), the main source of inspiration for our work, used a backbone composed of two parts: the first part is a classical CNN to extract low level features on events and the second part is made of recurrent ConvLSTM blocks to extract high-level spatio-temporal patterns, that are then fed to the SSD heads. Their architecture is shown in Fig. 4.16.

In our work, we sought to replace the two parts by a single SNN, capable of extracting both low and high-level spatio-temporal features using the convolutions and the temporal recurrence of the neurons potentials. The possible computational gains of SNNs compared to LSTM blocks are also significant.

Even if we consider that we have succeeded, the gap in performance is still very present. We have identified two main reasons for that: the reset of the neurons' potentials after each sample and the topologies of our SNNs not allowing the best training, which lead to very high hardware resources requirements for the training of SNNs on the huge Prophesee event datasets.

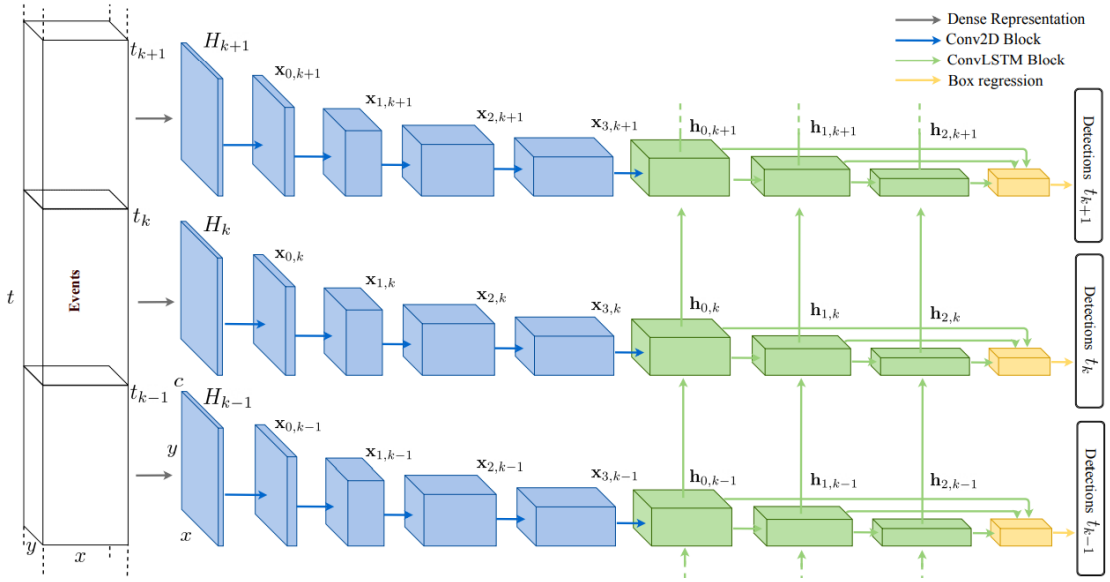


Figure 4.16: Prophesee RED architecture for object detection (Perot et al., 2020).

4.4.1 Temporal reset of SNNs

Prophesee RED processes samples lasting 50ms as a single frame, thus outputting a prediction every 50ms. The GEN1 dataset being composed of 60 seconds clips, they thus train their network on 1200 samples per clip. They implement a temporal recurrence using ConvLSTM, with memory cells initialized to 0 at the beginning of the clip. Then, the memory cells are updated throughout the whole clip, providing long-term temporal information for each prediction. Since ground truth bounding boxes are only available every 250ms at best, they train their network using truncated backpropagation through time when they encounter a ground truth label. This operation mode allows them to detect objects even if no new event has occurred, using the memory cells of ConvLSTM.

For comparison, our SNNs are trained on individuals samples of 50ms, further subdivided in T timesteps. Each sample is associated to one or multiple ground truth bounding boxes, there is no continuity between each sample. In order to have a functional training, we must therefore reset the potentials of our SNNs neurons between each sample since the samples are not connected, which prevents long-term memory that could tremendously help the performance. Moreover, this is not representative of a real time use, where the SNN will not be reset every 50ms.

Considering a learning without temporal reset requires to rethink the way data are input to the SNNs and to implement a *truncated* backpropagation through time learning algorithm for SNNs, as we will see in the Chapter 5.

4.4.2 Towards a general architecture for processing event data with SNNs

In this work, and in literature in general, we designed SNNs by reproducing popular classical neural networks topologies. In an effort to not reinvent the wheel, we reproduced in this chapter VGG, MobileNet and DenseNet. The SSD object detection framework was designed to be applied on frames, not on temporal event data, and we now believe that in order to get efficient processing of event data with SNNs we need to develop new solutions. Using classical neural network topologies had induced an important overhead in the training time of SNNs. Event data is temporal, and thus requires multiple timesteps. The surrogate gradient learning rule is an approximation, and needs to duplicate the network T timesteps times to learn using backpropagation through time. Even with the most recent SNN frameworks, this overhead is making difficult the training of performing SNNs. For these reasons, we think it is not sustainable to use SNNs as large as in non-spike networks, despite the fact that the performances reached by the SNNs are still inferior. We must then develop new SNN architectures, lightweight and powerful to tackle complex tasks such as object detection.

Based on our results, especially concerning the residual connections through concatenation (DenseNet) and the batch normalization layers, we thus seek to design a scalable, multi-purpose architecture capable of processing event data with maximum performance and minimum computational cost.

5 Designing high-performance SNNs

Contents

Previous chapters have shown that spiking neural networks are now able to reach competitive performance on event data, on real-world computer vision tasks such as classification and object detection. Enabled by the emergence of new methods (surrogate gradient learning, PLIF) and frameworks (SpikingJelly), we have developed and trained new SNN architectures by relying in part on classical

ANN architectures. However, the resulting SNNs are still inferior in performance and at the same time too big to be eventually embedded.

To solve these two problems, we propose in this chapter two new SNNs models, ST-VGG and ResCat-SNN, and solutions to improve the training of our SNNs on the object detection problem. As a result, we managed to improve our results on all event datasets with lightweight and scalable SNNs.

This chapter is organized as follows. First, we will study more thoroughly the importance of batch normalizations in SNNs, outlining reasons why the BN-CONV order performs better and how to fuse these two layers at the inference phase in order to remain in the spike domain. Second, we introduce new methods for the design of high-performance SNNs: a "patchify" stem and concatenation-based residual blocks, that we use to propose two new SNNs backbones, entitled ST-VGG and ResCat-SNN, and their variants ST-VGG+SSD and ResCat-SNN+SSD for object detection. Third, in order to improve our results on object detection with SNNs, we present how we implemented a temporal continuity during the training. Then, we validate our new high-performance SNNs on the automotive event datasets encountered during this thesis, both for classification and object detection. Finally, we discuss the influence of our contributions in the results achieved, in particular the patchify stem, the training without temporal reset for object detection and the voxel cube representation.

5.1 Batch normalization layers in SNNs

In Section 3.4.4, we found that Batch Normalization (BN) layers are vital to the training of complex SNNs, as their removal made our networks significantly less accurate on classification tasks. We also found out that the placement of the batch normalization is important, as placing BN layers before convolutions consistently produces better results than placing them after.

We investigate in this section the reasons of this improved performance, and our method to fuse a learned batch normalization layer with its following convolution, as it is surprisingly not available in the literature.

5.1.1 Importance of the BN-CONV order

In Section 3.4.4, we showed experimentally that batch normalization layers significantly contribute to the performance of our proposed SNNs (spiking VGG-11, spiking MobileNet-64 and spiking DenseNet121-24) on the event classification dataset Prophesee N-CARS. We reproduce in Table 5.1 the results obtained.

Table 5.1: Influence of Batch Normalization when training SNNs on N-CARS (average over 3 runs).

Models	Accuracy ↑		
	BN-CONV order	CONV-BN order	No BN
VGG-11	**0.933**	0.819	0.649
MobileNet-64	**0.922**	0.812	0.736
DenseNet121-24	**0.904**	0.808	0.698

In order to prove that the presence and placement of BN layers enables a better training of SNNs, we will measure the gradient flow of our spiking VGG-11 across all timesteps for the same batch during the final training epoch, where the training is almost complete and therefore the gradients are stable. The gradient flow (Mathew, 2022) is a representation of the average and maximum values of gradients in every layer, where a correct gradient flow is represented by the average gradients of the first layer being different from zero. If the average gradient values in some layers are close to zero, the training of the network will be impacted, as these layers will not be able to update their weights.

Furthermore, we will also measure the number of non-zero gradient values in each layer. We hypothesized in Section 3.4.4 that placing batch normalization layers before convolutions transform the sparse spikes into fully dense inputs, thus generating fully dense gradient values. More non-zero gradient values lead to more meaningful updates of the convolutions weights, leading to a better and/or faster training.

No Batch Normalization The gradient flow and non-zero gradient values percent for the spiking VGG-11 model without batch normalization layers are represented in Fig. 5.1. As we can see, the majority of the convolution layers have very low average gradient values and nearly half of all the gradient values are equal to zero, corresponding to an absence of input spikes. We believe that this low number of gradient combined with their low average values do not allow our network to learn properly.

Batch Normalization after convolutions When adding batch normalization after convolutions, the mean gradient values and percent of non-zero gradient are much higher, as illustrated in Fig. 5.2. The majority of layers still have relatively low gradient values and some layers have up to 40% of zero-valued gradients, which we believe still impairs the training of our SNN.

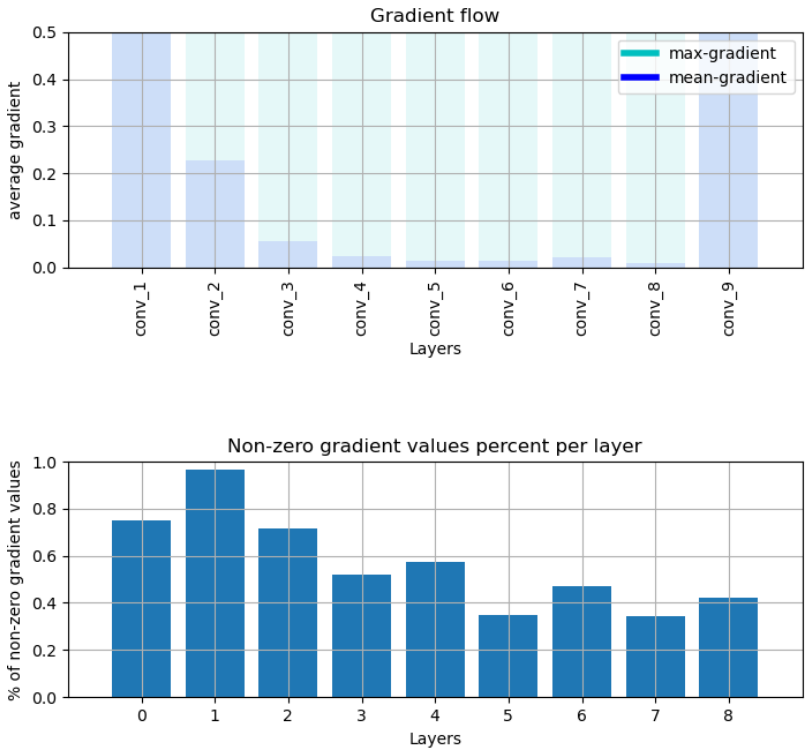


Figure 5.1: No BN. Gradient flow and percentage of non-zero gradient during the training of our spiking VGG-11 without batch normalization layers on N-CARS.

Batch Normalization before convolutions Finally, we represent the gradient flow and the percent of non-zero gradient values of our spiking VGG-11 with batch normalization before convolutions in Fig. 5.3. The average gradients all have higher values than in the CONV-BN order and the gradients are fully dense, leading to weights updates with the maximum possible information.

By analyzing the gradient values on Spiking VGG-11, we provide intuitions on the reasons why the BN-CONV order brings better performance: the gradient values are higher and more numerous than in the CONV-BN, which leads to a better and/or faster training. To draw more definitive conclusions, similar studies should be conducted on a variety of datasets and networks, as our observations could only be valid for event data for example.

5.1.2 Fusing a BN layer with a subsequent convolution

Batch normalization layers have the great advantage of being composed of linear operations. As such, their weights can be fused together with the adjacent convolutional layer weights, which allows for the complete removal of BN layers from the network at the inference phase. This weights fusion is vital in SNNs using batch

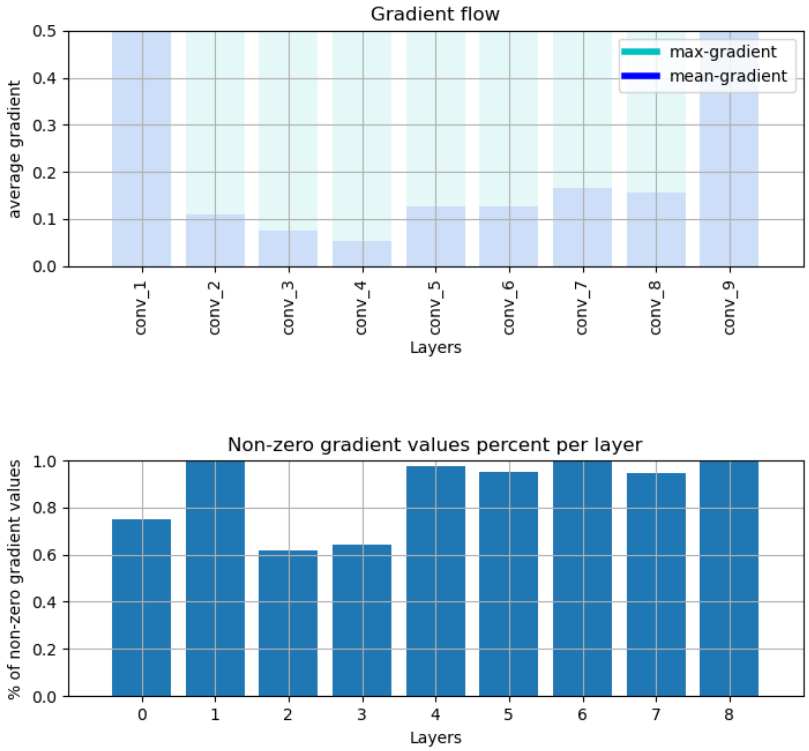


Figure 5.2: CONV-BN order. Gradient flow and percentage of non-zero gradient during the training of our spiking VGG-11 with the CONV-BN order on N-CARS.

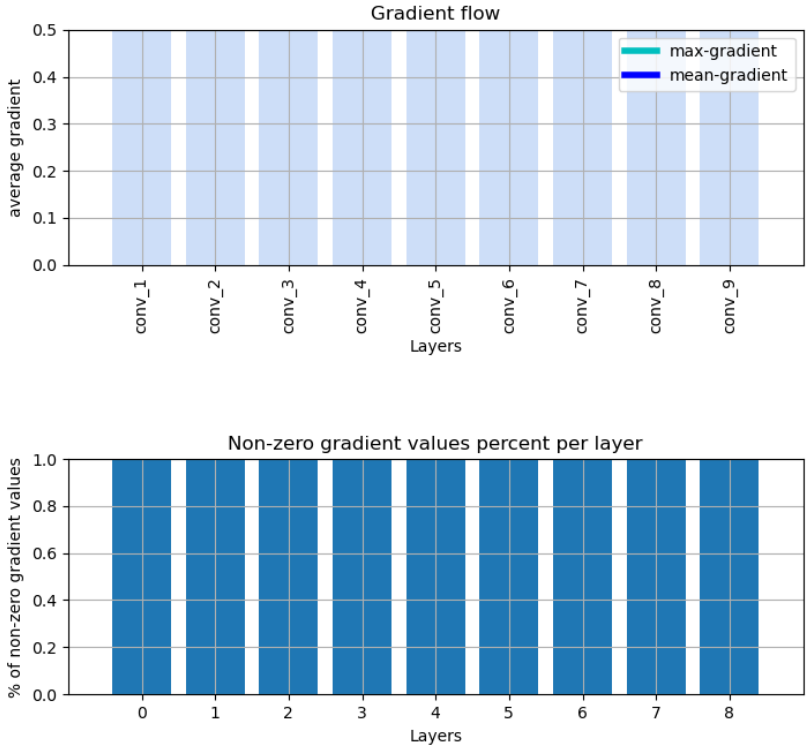


Figure 5.3: BN-CONV order. Gradient flow and percentage of non-zero gradient during the training of our spiking VGG-11 with the BN-CONV order on N-CARS.

normalization, otherwise multiplication operations between floating-point numbers would be needed in the convolutions, ruining the benefits of spiking neural networks.

However, the fusion of the batch normalization layers weights with the adjacent convolutional layer weights has only been shown with the CONV-BN order. In this section, we will demonstrate that such a fusion is also possible for the BN-CONV, with zero impact on the performance of the network as long as the padding is placed before the BN layers and not in the convolutions.

In this section, we will focus only on 2D convolutions, although everything can easily be generalized to N-dimensional convolutions.

We begin by restating the basic operation of batch normalization. Let x_i be a single element of a batch, a tensor of activations of shape $C_{in} \times H_{in} \times W_{in}$ that we want to normalize. Inside a batch normalization layer, it will be normalized as follows:

$$\hat{x}_i = \gamma \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta \tag{5.1}$$

With μ_b and σ_b^2 representing the mean and the variance computed over a batch, ϵ a small constant included for numerical stability, γ the scaling factor and β the shift factor. During training, μ and σ are recomputed for each batch while the parameters γ and β are learned during the network training. At inference, we use the exponential moving average of μ_b and σ_b, denoted μ and σ. All μ_b, σ_b, μ, σ, γ and β are vectors of size C_{in}, while ϵ is a scalar.

On the other hand, a convolution is composed of a sliding dot product (named *cross-correlation* and noted $*$) between an input and a weight matrix W of shape $(C_{out} \times C_{in} \times H_{kernel} \times W_{kernel})$, summed with a bias vector b of size C_{in}. Thus, a convolution operation following a BN layer can be expressed as follows:

$$\begin{aligned} y_i &= W * \hat{x}_i + b, \\ y_i &= W * (\gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta) + b \end{aligned} \tag{5.2}$$

Therefore, we aim to compute the new weights W' and biases b' such as:

$$y_i = W' * x_i + b' \tag{5.3}$$

Let $\phi = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$, a vector of size C_{in}. We note ϕ^* a view of ϕ of shape $(1 \times C_{in} \times 1 \times 1)$. Then, computing W' is simply the element-wise matrix multiplication (or Hadamard product noted $\odot$) product between W and ϕ^*:

$$W' = W \odot \phi^* \tag{5.4}$$

To compute b', we cannot directly compute a cross-correlation between W of shape $(C_{out} \times C_{in} \times H_{kernel} \times W_{kernel})$ and $(\beta - \mu\phi)$ of size C_{in}. For simplicity, we note $\Phi = (\beta - \mu\phi)$. In order to obtain a C_{out} output, we introduce $(\beta - \mu\phi)^*$, which has a shape of $(1 \times C_{in} \times H_{kernel} \times W_{kernel})$. It is constructed by repeating along the last two dimensions the $(C_{in} \times 1 \times 1)$ view of $(\beta - \mu\phi)$. As a result, the cross-correlation between W and $\phi^{*'}$ will result in a tensor of shape $(1 \times C_{out} \times 1 \times 1)$, which can be viewed as a C_{out} tensor. Therefore, the expression of b' is as follows:

$$b' = W * (\beta - \mu\phi)^* + b \tag{5.5}$$

Alternatively, all the above equations can be expressed in PyTorch code:

```
def fuse_bn_conv_weights(conv_w, conv_b, bn_mu, bn_sigma, bn_eps, bn_gamma, bn_beta):
    phi = bn_gamma * torch.rsqrt(bn_sigma + bn_eps)

    # W' computation
    w_prime = conv_w * phi.view(1,-1,1,1)

    # b' computation
    big_phi = (bn_beta - bn_mu * phi).view(1,-1,1,1)
    big_phi = big_phi.repeat(1, conv_w.shape[2], conv_w.shape[3])
    tmp_b = torch.nn.functional.conv2d(big_phi, weight=conv_w)
    b_prime = tmp_b.squeeze() + conv_b

    return w_prime, b_prime
```

Listing 5.1: Fusing BN and CONV weights in the BN-CONV order in PyTorch

This fusing was tested over 100 runs with random inputs, random shapes and random weights, and the absolute element-wise difference between outputs computed by the fused weights and those computed by the normal BN-CONV weights was always inferior to 1e-6, proving their equivalence.

However, an important parameter of convolutions was not taken into account: padding. If the convolution has padding, it will pad the input with the pad value (generally 0) after the batch normalization layer. After the fusing, this behavior will not longer be reproducible.

The simplest solution we found to use padding in our convolutions is to do the padding before the batch normalization layers. As a result, the BN layers parameters will be slightly erroneous but we did not notice any degradation in our SNNs trainings. Moreover, fusing this extra layer is trivial as since we just need to remove it and add equivalent padding in the fused convolution.

Therefore, our spiking convolutional layers take the form of 4 successive layers: a layer of constant padding (if necessary), a batch normalization layer, a convolution with no padding and no bias, a layer of neurons. After fusing the BN parameters and convolution weights, we simply get a convolution with padding and bias followed by a layer of neurons.

5.2 Proposed SNNs: ST-VGG and ResCat-SNN

The SNNs developed in previous chapters have made it possible to achieve high performances, at the cost of a large number of parameters and/or layers, making their embeddability difficult. In order to embed high-performance SNNs, we present in this section two new lightweight and powerful SNN backbones: SpikeThin-VGG (ST-VGG), a feedforward architecture inspired by VGG-11 and ResCat-SNN, a variant that include concatenation-based residual blocks. We will study their performance on classification and object detection tasks in Section 5.4.

5.2.1 "Patchify" stem

Classical convolutional neural networks have been the dominant architecture for vision tasks for many years, but recent experiments have shown that models based on the Transformer architecture (Vaswani et al., 2017) may exceed their performance in some settings. To process images, vision transformers use patch embeddings, grouping together small regions of the image into single input features (see Fig. **??**).

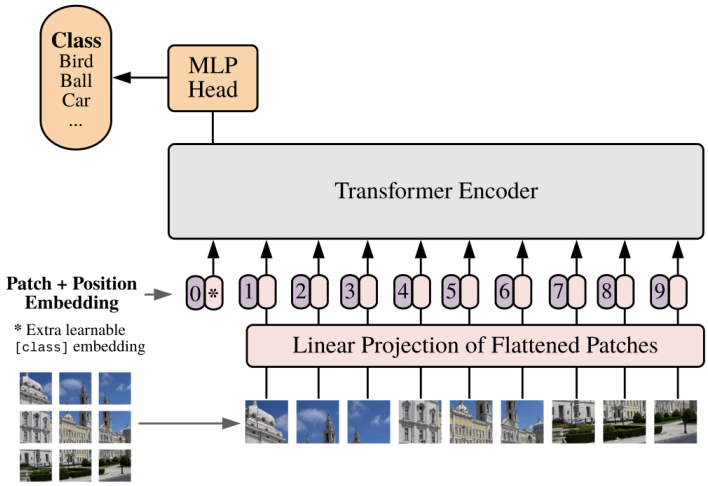


Figure 5.4: Architecture of a Vision Transformer (ViT). Since Transformers were designed to process sequences, ViT divides an image as a sequence of small patches (*patch embeddings*). Illustration taken from (Vaswani et al., 2017).

Recently, Trockman and Kolter (2022) have studied whether the performance of vision transformers does not come, at least in part, from this patching of the input image. They state that patch embeddings allow all the downsampling to happen at once, immediately decreasing the internal resolution and thus increasing the effective receptive field size, making it easier to mix distant spatial information. They also propose a simple way to implement it in CNNs: using a convolution of size p and a stride p would be equivalent to using patch embeddings of size p. Therefore, this first convolution is now usually named a "patchify" stem.

Shortly after, a high-performing CNN architecture named ConvNeXt was proposed in (Z. Liu et al., 2022), also featuring a "patchify" stem. They made the observation that a standard ResNet uses a stem composed of a 7×7 convolution layer with stride 2, followed by a max pool, which results in a $4\times$ downsampling of the input images. Inspired by vision transformers, they replaced the ResNet-style stem cell with a patchify layer implemented using a 4×4, stride 4 convolutional layer. The accuracy of their network slightly improved, suggesting that the stem cell in a ResNet may be substituted with a simpler “patchify” layer.

This approach is all the more attractive for SNNs: reducing quickly the spatial dimensions of the input data reduces the number of membrane potentials kept in memory and being able to do it in a single layer removes the need for max pooling. Furthermore, it requires less parameters and computations. For these reasons, we will now use a patchify stem for our SNNs, implemented with a 4×4, stride 4 convolutional layer.

5.2.2 ST-VGG

Inspired by our work on Spiking VGG in Chapter 3, we now present Spiking-Thin VGG (see Fig. 5.5): a lightweight 8-layers backbone that uses a patchify stem, BN-CONV-PLIF spiking blocks and no pooling. The feature maps spatial dimensions are reduced throughout the network using stride.

For classification tasks, the final layer of the network is a 1×1 convolution without PLIF, removing the need for an average pooling layer to reduce the feature maps size to 1×1. The final output of the network is obtained by summing spatially and temporally these activations.

By modifying the number of channels C output by the patchify stem, multiple variants can be designed. We will focus on three variants: 16-ST-VGG, 32-ST-VGG and 64-ST-VGG.

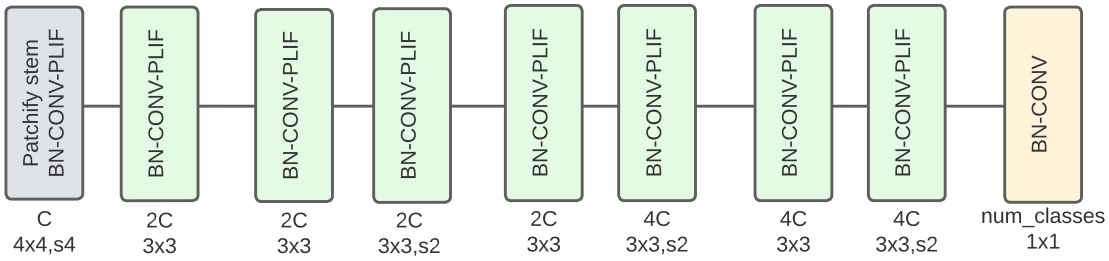


Figure 5.5: C-ST-VGG architecture. C denotes the number of channels after the patchify stem, the final feature maps having $4C$ channels. The final layer is a 1×1 convolution that outputs *num_classes* channels, the final prediction is obtained by summing spatially and temporally this output.

5.2.3 ResCat-SNN: concatenation-based residual connections

Based on the better results attained by our Spiking DenseNet in Chapter 3, we propose a variation of ST-VGG that include concatenation-based residual connections named ResCat-SNN.

We propose a concatenation-based residual block composed of two spiking convolutions and a single 1×1 spiking convolution as the residual connection (see Fig. 5.6). Indeed, as the features are concatenated the number of channels will grow rapidly, therefore the residual connection is using a 1×1 convolution to reduce the number of output channels that will be concatenated.

Then, we incorporate these concatenation-based residual blocks in our ST-VGG architecture to design a new architecture called ResCat-SNN, illustrated in Fig. 5.7. To avoid an excessive number of parameters, we reduce by half the number of channels output by the residual connection.

In the same way as for the ST-VGGs, we can construct variants of ResCat-SNN by varying the number of channels C output by the patchify stem: ResCat-SNN-16, ResCat-SNN-32, ResCat-SNN-64. We compare the number of parameters and ACCs of all our proposed variants of ST-VGGs and ResCat-SNNs in Table 5.2. Our ResCat-SNNs are slightly bigger than their equivalent ST-VGGs in parameters count but close in number of operations, because of and thanks to the addition of 1×1 convolutions.

5.2.4 Proposed networks for object detection

Similarly to what was done in Chapter 4, we can now propose new SNNs for object detection by combining our new proposed SNNs backbones with SSD heads.

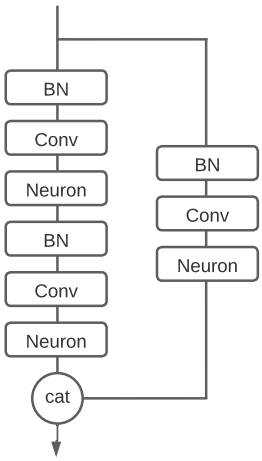


Figure 5.6: A concatenation-based residual block. The residual connection is composed of 1×1 convolution in order to reduce the number of channels and downsample the input with stride if necessary. The output is purely composed of spikes, as the concatenation is done along the channels dimension.

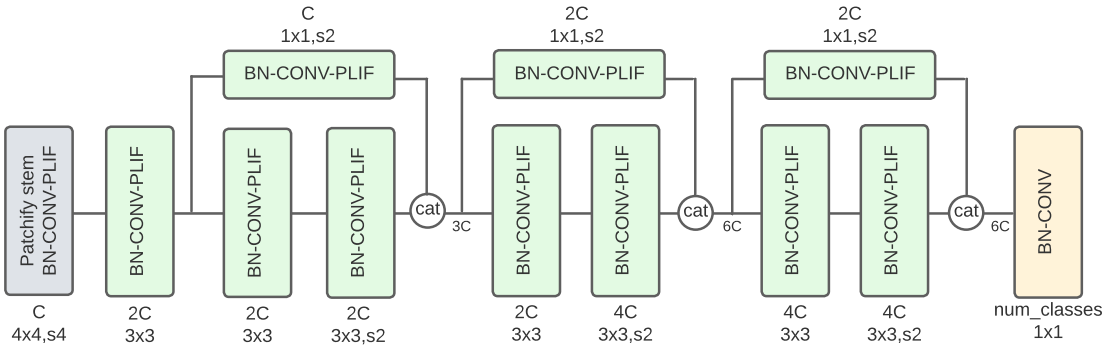


Figure 5.7: ResCat-SNN-C architecture. C denotes the number of channels after the patchify stem, the final feature maps having $6C$ channels. The residual block outputs supplementary feature maps, half of the block output channels. The features are concatenated and passed on to the rest of the network.

Compared to our previous models, we modify the extra layers to remove the 1×1 convolutions and we reduce their number of channels, in order to minimize both the number of layers and the parameters count. Our ST-VGG+SSD is presented in Fig. 5.8 and ResCat-SNN+SSD in Fig. 5.9.

Having introduced new backbones and new object detection models, we will now present our approach to increase the performance achieved by our SNNs on the object detection task, to finally dedicate Section 5.4 to the results achieved by our networks on both classification and object detection tasks on event data.

Table 5.2: Number of parameters and ACCs of our ST-VGG variants.

Model	Number of parameters	ACCs per timestep
16-ST-VGG	125k	105M
32-ST-VGG	499k	415M
64-ST-VGG	1.994M	1.651G
ResCat-SNN-16	216k	136M
ResCat-SNN-32	862k	541M
ResCat-SNN-64	3.442M	2.158G

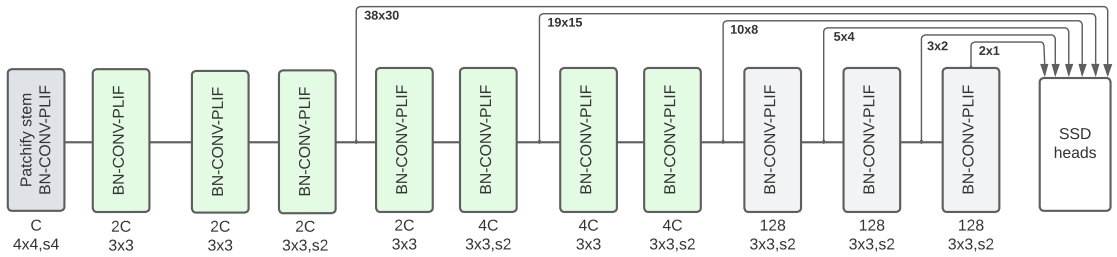


Figure 5.8: ST-VGG+SSD architecture.

5.3 Temporal continuity for object detection

We presented in Chapter 4 the first SNNs able to perform object detection on the complex real-world event dataset Prophesee GEN1, achieving 0.19mAP, which is still an important difference with conventional networks that can reach 0.40mAP. Our SNNs were trained on individuals samples of 100ms, each one associated to one or multiple ground truth bounding boxes, but without any temporal continuity between samples. As a result, we had to reset the membrane potentials to zero between each sample, preventing long-term memory that could tremendously improve the performance and preventing the use of our networks in a real time setting.

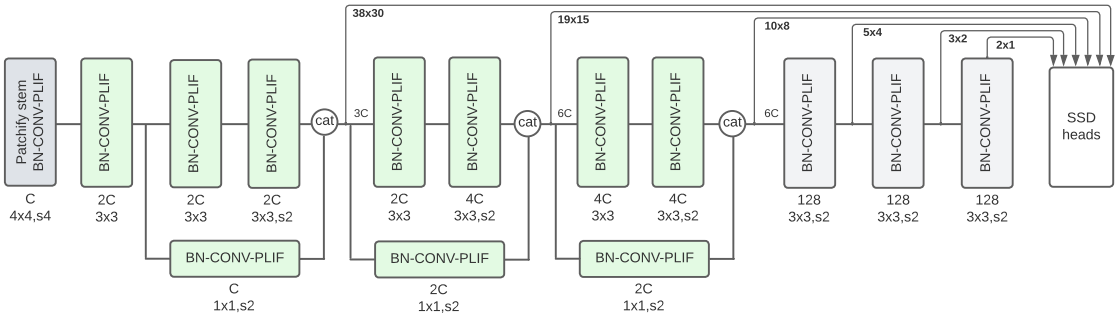


Figure 5.9: ResCat-SNN+SSD architecture.

In this section, we now consider a training without temporal reset, which requires a rethinking of the way data are input to the SNNs and to implement a truncated Backpropagation Through Time learning algorithm for SNN.

5.3.1 Truncated Backpropagation Through Time

Principle of operation

As stated in Section 2.3.3, SNNs are recurrent neural networks and as such, can be trained using Backpropagation Through Time (BPTT): an unrolled network is created by making copies of the original for each timestep, on which standard backpropagation can then be applied. An important problem with BPTT is that the higher the number of timesteps, the deeper the unrolled network will be, which can make the training difficult (vanishing gradient problem) or computationally expensive.

To train networks on long sequences, a modified version of the BPTT learning method has been proposed, called truncated BPTT. The main idea is illustrated in Fig. 5.10.

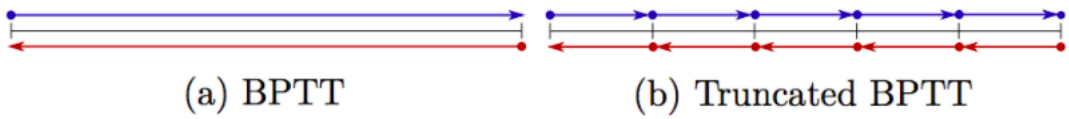


Figure 5.10: (a) Backpropagation Through Time: the network is unrolled over the entire duration of the sample (blue arrow), and gradients are computed over the whole sample (red arrow). (b) Truncated Backpropagation Through Time: the sample is now divided in a fixed number of parts, and gradients are now computed over each single part.

In classical BPTT, we first forward through the entire sample and then compute the gradients during the backward pass over the whole sample, which can be computationally and memory intensive when the duration (i.e. the number of timesteps) of the data is high. Truncated BPTT, the forward and the backward pass are applied on a truncated part of the sample, reducing the computational and memory needs, while increasing the number of updates. However, truncation favors short-term dependencies, as the weights updates depend only on nearby timesteps.

Additionally, more complex variants of Truncated BPTT exist. By defining $k1$, the number of timesteps used in each forward pass and $k2$, the number of previous timesteps used during the backward pass, exotic TBPTT methods with different $k1$, $k2$ values can be used. Because of their difficult implementation, we will only use the classical version where $k1 = k2$ in our works.

Implementation of truncated BPTT in SNNs

In modern deep learning frameworks, truncated BPTT with $k1 = k2$ can easily be implemented. Indeed, in PyTorch and TensorFlow, a computational graph of operations is constructed during the forward pass on which gradients will be computed during the backward pass. Truncated BPTT can therefore be implemented by triggering the backward pass every $k1$ timesteps, which will compute the gradients over the computational graph. To implement $k1 = k2$, the computational graph must be emptied after each backward pass (i.e. every $k2$ timesteps) in order not to influence the subsequent gradient calculations.

In SNNs, the membrane potentials serve as a memory between timesteps and thus represent the only tensors that influence the gradient computations of subsequent timesteps. Truncated BPTT is therefore implemented in SNNs by simply detaching, or removing, these tensors from the operation graph after a backward pass. In the case where Parametric LIF neurons are used, the learnable weight of PLIF neurons must also be detached. We provide in Listing 5.2 our detach method, called after each backward pass.

```
def detach(self):
    last_seq = None
    for m in self.modules():
        if isinstance(m, neuron.ParametricLIF):
            m.v.detach_()
            m.w.detach_()
        elif isinstance(m, nn.Sequential):
            last_seq = m
```

Listing 5.2: Method to detach all the PametricLIF neurons' membrane potentials v and learnable time constant w from the computational graph.

5.3.2 Continuous samples and batch construction

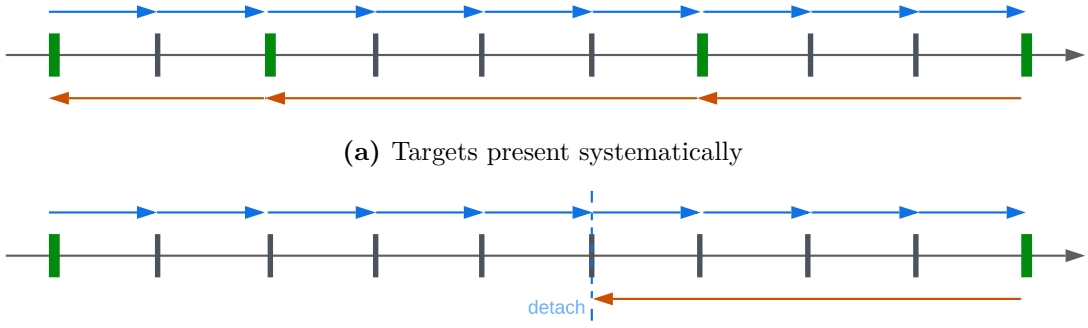
To remove the need for the temporal reset of our neurons' membrane potentials, we need to rethink the way we input data from the Prophesee GEN1 object detection dataset into our SNNs. The dataset is composed of over a thousand 60-second clips of event data, but ground truth bounding boxes are only available at best every 250ms.

Sample construction

Previously, we constructed our samples by taking the 100ms of event data preceding the times when ground truth bounding boxes were present, without consideration for the continuity inside the same 60-second clip. From now on, we will instead tackle the dataset in a continuous manner: each sample now corresponds to a single

60-second clip, divided in $T = \frac{60}{\Delta t}$ timesteps. To be coherent with previous works on this dataset (Perot et al., 2020), we consider $\Delta t = 50$ms, therefore each sample is composed of 1200 timesteps. Each timestep is composed of 50ms of event data and ground truth bounding boxes present in the 50ms interval, and if there are none, the timestep simply doesn't have any ground truth target.

To train our SNNs on such a large number of timesteps, we use the above-mentioned Truncated Backpropagation Through Time to compute a loss and trigger a backward pass at every timestep *at most*, that would be equivalent to $k1 = k2 = 1$. We use the term *at most* because an important point of this construction is that a timestep not having ground truth targets does not mean that no objects are present, as ground truth bounding boxes are only available at best every 250ms. Thus, we must not train the network to predict that there are no objects when there are no targets, as it may not be the case. This behavior is implemented by computing a loss and doing a backward pass only when ground truth targets are present. When no target is present, a normal forward pass is executed to update the potentials, but no loss is calculated, as illustrated in Fig. 5.11a. But with this approach, if no ground truth are present for an high number of timesteps, the subsequent backward pass will have high computational and memory cost. To avoid this situation, we define a maximum number of timesteps $k2_{max}$ included in the computational graph, detaching the tensors when this number is exceeded. This behavior is illustrated in Fig. 5.11b.



(a) Targets present systematically

(b) Absence of targets for a certain number of timesteps

Figure 5.11: TBPTT applied to object detection. (a) The forward pass (blue arrow) is executed at every timestep, while a backward pass is initiated only when ground truth targets (green) are present. (b) If ground truth targets are absent for a fixed number of timesteps (here 5 timesteps), we detach the tensors anyway to avoid exploding computational and memory costs during the next backward pass.

In the end, the variant of truncated Backpropagation Through Time that we use is equivalent to setting $k1 = 1$ and $k2 \in \{1, \dots, k2_{max}\}$.

Batch construction

Due to the nature of our samples and training, our networks will do at most T backward passes for a single sample, one for each timestep. This poses a concern for the generalization of our training, as the training will occur for a long period of timesteps on a single clip. Since we cannot shuffle the timesteps inside a sample as that would destroy the continuity, the only way to introduce variety during training is to construct batches where n samples will be iterated sequentially together.

In the Prophesee GEN1 dataset, samples are labeled at a certain frequency, either 1, 2 or 4Hz (i.e. every 250, 500 or 1000ms), but not on synchronized timestamps. One could be inclined to resynchronize all samples together, such that ground truth bounding boxes occur consistently on the same timesteps. The problem with this approach is that the neural networks will now learn to output predictions on these particular timesteps, using the intermediate timesteps without ground truth labels to construct a single meaningful prediction. This is not the behavior we expect to train, since we want the network to learn to predict at each timestep.

For this reason, we choose to not resynchronize the samples, which raises another problem. When considering a batch size of size n, ground truth bounding boxes can occur at any timestep and be part of only some samples of the batch. Therefore, we need to adapt our truncated backpropagation so that the training goes well on a batch. At every forward pass, we check if any of the samples has ground truth labels, and if it is the case, we compute a loss **only on these samples**, and trigger a backward pass that will work in the same way as described in the previous section. By computing a loss only on some samples, the network will not learn to predict that there are no objects on the other samples, because, again, this might not be the case. This behavior is illustrated in Fig. 5.12.

Due to the limitation of deep learning frameworks, the whole computational graph is unfortunately reset after a backward pass: the samples that did not have ground truth labels during a backward pass lose their history in the computational graph, which will therefore not influence the next backward pass that concerns these samples. An implementation capable of keeping different computational graph of different depth for each sample would for sure provide a better training.

Finally, we notice that the batch size n now has an influence on the number of backward passes executed. Indeed, a small batch size would generate backward passes which may be several timesteps apart, and often run into the maximum number of timesteps kept in memory $k2_{max}$. On the other hand, a large batch size would trigger backward passes on almost every timestep, since statistically the ground truth labels can be present at any timestep. But one could argue that

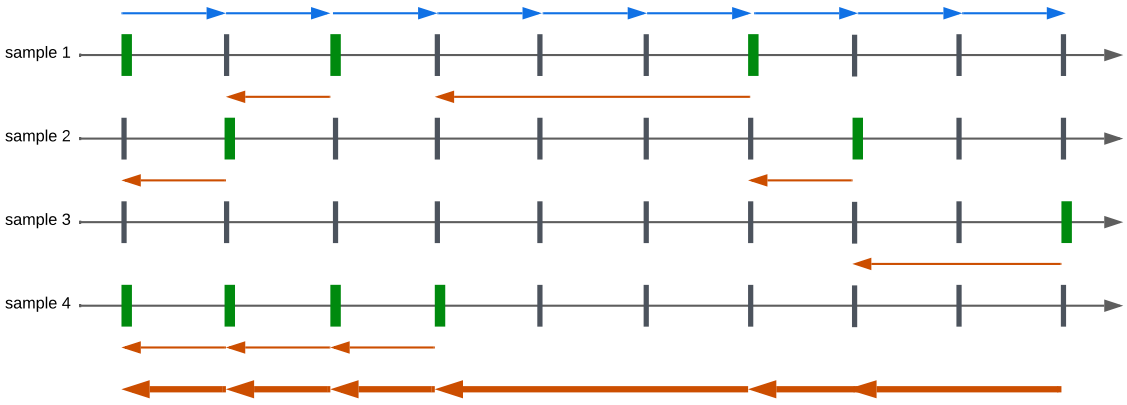


Figure 5.12: TBPTT with a batch size of 4. A backward pass (bold red arrows) is initiated when ground truth bounding boxes are present in any sample, but only on those samples (red arrows). The whole computational graph is reset after a backward pass, even if only one sample was concerned.

a very high batch size leads to a great representativity, in the sense that many samples will have ground truths at each timestep.

We are not able to quantify precisely how the batch size n impacts our training with TBPTT, but our best results were obtained with a batch size of 32, while batch sizes of 16 and 64 led to worse results. A batch size of 32 leads to about one backward pass every 3 timesteps, with a large disparity between the number of samples involved.

5.4 Results

In this section, we will now evaluate our new SNNs architectures on the automotive event datasets studied in this thesis, both for classification and object detection tasks.

5.4.1 Classification on event data

We evaluate our new SNN backbones on the Prophesee automotive event datasets: N-CARS and GEN1 Classif, already presented in Section 3.4.2. For both datasets, we used samples of 100ms encoded as binary voxel cubes of 5 timesteps and 2 micro time bins. The samples were resized to 64×64 pixels using nearest-neighbor interpolation to keep the input events binary. The networks were trained over 30 epochs, with a learning rate of $5e^{-3}$, as it was done in Section 3.4.2.

We compared the results achieved by our SNNs on Prophesee N-CARS with state-of-the-art models in Table 5.3.

Table 5.3: Comparison with state-of-the-art models on Prophesee N-CARS

Methods	**Representation**	**Network**	**N-CARS acc ↑**
HATS (2018)	TimeSurface	N/A	0.902
Gabor-SNN (2018)	Spike	SNN	0.789
HybridSNN (2021)	VoxelGrid	SNN	0.77
HybridSNN (2021)	VoxelGrid	SNN-CNN	0.906
YOLE (2019)	VoxelGrid	CNN	0.927
Asynet (2020)	VoxelGrid	CNN	**0.944**
EvS-S (2021)	Graph	GNN	0.931
Ch.3 VGG-11	VoxelCube	SNN	**0.933**
Ch.3 MobileNet-64	VoxelCube	SNN	0.922
Ch.3 DenseNet121-24	VoxelCube	SNN	0.904
16-ST-VGG	VoxelCube	SNN	0.917
32-ST-VGG	VoxelCube	SNN	**0.941**
64-ST-VGG	VoxelCube	SNN	0.925
ResCat-SNN-16	VoxelCube	SNN	0.927
ResCat-SNN-32	VoxelCube	SNN	0.935
ResCat-SNN-64	VoxelCube	SNN	**0.940**

Our ST-VGGs and ResCat-SNNs reach excellent accuracies on the Prophesee N-CARS datasets, over 91% for all models. The accuracy results are however not linear, as a network too big seems to result in a slightly lower performance, surely a consequence of overfitting. The ResCat-SNNs manage to reach higher accuracies than the ST-VGGs thanks to their concatenation-based residual connections and added parameters. Nonetheless, our best accuracy was obtained with a 32-ST-VGG, reaching the new state-of-the-art result for SNNs of 94.1%.

Table 5.4 provides extensive results of all the spiking neural networks considered in this thesis on both automotive classification datasets.

We notice that our proposed ST-VGGs and ResCat-SNNs reach state-of-the-art accuracies on Prophesee N-CARS, with an activity usually inferior to 30%. These SNNs reach better accuracies than our Ch. 3 SNNs while being around 10 times smaller. However, the results are significantly lower on the Prophesee GEN1 Classif dataset, which may indicate an implementation problem on this dataset as our trainings have difficulties to converge. It could also comes from the lower parameters count, or from the immediate downsampling in the first layer, which was not done until the third layer in our Ch. 3 SNNs.

Table 5.4: Comparison between our spiking models on the automotive event classification datasets Prophesee N-CARS and Prophesee GEN1 Classif.

Models	#Params	ACCs/ts	N-CARS		GEN1 Classif.	
			Acc ↑	Act ↓	Acc ↑	Act ↓
Ch.3 VGG-11	9.23M	0.61G	**0.933**	12.04%	0.969	14.69%
Ch.3 MobileNet-64	18.81M	4.20G	0.922	17.14%	0.966	30.60%
Ch.3 DenseNet121	3.93M	2.25G	0.904	33.59%	**0.975**	27.26%
16-ST-VGG	125k	6.01M	0.917	23.36%	0.897	24.30%
32-ST-VGG	499k	23.27M	**0.941**	22.64%	**0.920**	29.03%
64-ST-VGG	1.994M	93.06M	0.925	41.61%	0.906	34.85%
ResCat-SNN-16	216k	7.78M	0.927	28.73%	0.893	30.18%
ResCat-SNN-32	862k	30.59M	0.935	26.62%	**0.917**	33.25%
ResCat-SNN-64	3.442M	121.31M	**0.940**	36.05%	0.901	41.33%

5.4.2 Object Detection on Prophesee GEN1

We will now evaluate our proposed ST-VGG + SSD and ResCat-SNN + SSD models trained without temporal reset as explained in Section 5.3.1. We used a learning rate of $5e^{-3}$, a batch size of 32, and trained our networks over 50 epochs using a cosine annealing learning rate scheduler that gradually decrease the learning rate towards 0. We used the Adam optimizer with a weight decay of $1e^{-2}$. All convolutions were initialized using the Kaiming uniform method, and all batch normalization layers were initialized with a weight of 1 and a bias of 0. The Parametric LIF neurons all had an initial membrane time constant τ of 2, a membrane threshold of 1 and the Sigmoid function as the surrogate function. Norm of the gradient values were clipped to a maximum of 1 to avoid exploding gradients. All trainings were done in the SpikingJelly framework using a 48GB NVIDIA A40 GPU.

As presented in Section 5.3.1, each sample represents a 60-second clip and is composed of 1200 timesteps of 50ms, represented by a binary voxel. To avoid the addition of an additional hyperparameter, we did not use our voxel cube representation to add more temporal precision to each of these 50ms timesteps, but we will briefly study their contribution in Section 5.5.3. The results reached by our SNNs are presented Table 5.5.

Thanks to our training approach that does not require temporal resets, our new spiking models are able to reach higher mAP with a much lower number of parameters and ACCs per timestep. We reach 0.203 COCO mAP with our 64-ST-VGG+SSD model that has only 2.8M parameters and requires 1.73G ACCs. A

Table 5.5: Comparison with state-of-the-art models on Prophesee GEN1. Chapter 4 SNNs operate on 5 timesteps, while our ST-VGGs and ResCat-SNNs operate on a single timestep.

Methods	Type	#Params	ACCs/ts	Activity ↓	mAP ↑
Asynet (2020)	CNN	133M	-	-	0.15
MatrixLSTM (2020)	CNN	65M	-	-	0.31
RED (2020)	CNN	24M	-	-	**0.40**
Ch.4 VGG-11+SDD	SNN	12.64M	11.07G	22.22%	0.174
Ch.4 MobileNet-64+SSD	SNN	24.26M	4.34G	29.44%	0.147
Ch.4 DenseNet121+SSD	SNN	8.20M	2.33G	37.20%	**0.189**
16-ST-VGG+SSD	SNN	642k	125M	37.66%	0.135
32-ST-VGG+SSD	SNN	1.14M	453M	36.24%	0.184
64-ST-VGG+SSD	SNN	2.88M	1.73G	38.87%	**0.203**
ResCat-SNN-16+SSD	SNN	795k	165M	33.71%	0.147
ResCat-SNN-32+SSD	SNN	1.43M	596M	35.59%	0.192
ResCat-SNN-64+SSD	SNN	4.57M	2.27G	45.14%	**0.203**

network as small as 642k parameters is able to reach 13.5 test mAP, a performance comparable with a CNN of 133M parameters. The concatenation-based residual connections also improve the results, as all our ResCat-SNN+SSD reach higher or similar accuracies than their ST-VGGs counterpart. Our 64-ST-VGG+SSD and ResCat-SNN-64+SSD both reach **20.3 test mAP**, setting new state-of-the-art results for SNNs.

5.5 Discussion

In this section, we briefly measure the influence of two of this chapter contributions (patchify stem, training without temporal reset) and of our voxel cube representation on the object detection task, using our 32-ST-VGG+SSD on the Prophesee GEN1 object detection dataset.

5.5.1 Influence of the patchify stem

We train a 32-ST-VGG+SSD with a more traditional stem, i.e. the one we used in Chapter 4: a 3×3 convolution with a stride of 2, followed by a max pooling of size 3 and a stride of 2. The result of this training is depicted in Table 5.6.

As we can see, using a patchify stem or a traditional one provides a little performance boost, accompanied by a lowered number of ACCs and the absence of

Table 5.6: Comparison between a 32-ST-VGG+SSD using a patchify or a classical stem.

Model	Stem	#Params	ACCs	Activity ↓	mAP ↑
32-ST-VGG+SSD	Patchify stem	1.14M	453M	36.24%	**0.184**
32-ST-VGG+SSD	Conv + Pooling	1.14M	459M	34.70%	0.171

max pooling which are nice to haves when considering an embedded implementation. While the network using a conventional stem has a lower activity, both networks remain in the same order of magnitude. Therefore, we can only recommend the use of the patchify stem in SNN models for object detection.

5.5.2 Importance of the training without temporal reset

To measure the importance of our new approach for the training of SNNs without temporal reset for object detection, we train a 32-ST-VGG+SSD model with the method used in Chapter 4. As a reminder, it involved the construction of a sample by using the 100ms preceding a ground truth bounding box, and training the network over 5 timesteps without any temporal continuity between the samples. Consequently, the SNN required a reset of its membrane potentials before the start of a prediction on a new sample, which is incompatible with a real-time operation. The network without temporal reset was trained using the hyperparameters presented in the previous section, over 50 epochs.

The comparison between the two training methods on Prophesee GEN1 can be found in Table 5.7.

Table 5.7: Comparison between a 32-ST-VGG+SSD training with and without temporal reset. The network trained without temporal reset operates on a single timestep while the one trained with temporal reset operates on 5 timesteps.

Model	Training approach	#Params	ACCs/ts	Activity ↓	mAP ↑
32-ST-VGG+SSD	w/o temporal reset	1.14M	453M	36.24%	**0.184**
32-ST-VGG+SSD	w/ temporal reset	1.14M	453M	28.97%	0.123

The same network trained with our old training approach reaches 0.123mAP, 35% lower than our 32-ST-VGG+SSD trained without temporal reset. This important performance gap proves that preserving the temporal continuity between samples enables a better training of SNNs. This also suggests that the larger networks used in Chapter 4, which reached 0.18mAP with the old training approach, would also benefit from this new training approach, perhaps reaching even higher than 0.203mAP, but unfortunately we did not have the time nor the computing power to investigate it.

5.5.3 Influence of the number of time bins

In order to avoid adding an additional hyperparameter to our trainings, we did not use our voxel cube representation to train our object detection models. But while our previous trainings in Chapter 4 were performed on 5 timesteps of 20ms each, our trainings without temporal reset now occur on 1200 timesteps of 50ms each for computational cost reasons. While our results have improved, we have however lost temporal information in each timestep, which could be somewhat retrieved using our voxel cube representation.

To verify this claim, we trained the same 32-ST-VGG+SSD with 1, 2 and 4 time bins on the Prophesee GEN1 dataset. The results are presented in Table 5.8.

Table 5.8: Comparison between a 32-ST-VGG+SSD trained with 1, 2 or 4 time bins.

Model	**# time bins**	**mAP ↑**
32-ST-VGG+SSD	1	0.184
32-ST-VGG+SSD	2	0.186
32-ST-VGG+SSD	4	**0.192**

As a matter of fact, our voxel cube representation provides better performance, as the network is able to exploit more temporal information with multiple time bins. Using 4 time bins allows the representation of 12.5ms of events in different channels, which is closer to the temporal precision used in Chapter 4 (5 timesteps of 20ms each with 2 time bins, i.e. a temporal precision of 10ms). This is another demonstration that when the number of timesteps is constrained, we can use our voxel cube representation to obtain better results, with a totally negligible cost in terms of parameters and operations since only the number of input channels of the first layer is impacted.

5.6 Conclusion

This chapter represents the culmination of our years of research in SNNs design and training. We have addressed the reasons why our proposed BN-CONV order provides better performance, and developed new powerful lightweight SNN architectures entitled ST-VGG and ResCat-SNN. These architectures and their variants for object detection, ST-VGG+SDD and ResCat-SNN+SSD, reach even higher results than our previous SNNs, while being smaller and easier to embed.

In particular, a 500k parameters 32-ST-VGG reach 94.13% on Prophesee N-CARS, a new state-of-the-art result for SNNs. Thanks to our new training approach

without temporal reset, we were able to reach 20.3mAP on the complex real-world Prophesee GEN1 object detection dataset with a network of only two million parameters, a size much smaller than the networks usually considered.

These results prove once more that spiking neural networks trained directly on event data are now able to tackle real-world problems with good performance, with very few parameters and operations needed. Now that the performance criteria is met, there is still a lot of work to be done to embed these networks and verify their promise of low power consumption and low latency. As a first step towards this end goal, we will study in the next chapter a method to analytically estimate the energy consumption of an SNN, in order to provide new insights into the energy efficiency of SNNs.

6

Towards efficient embedded SNNs

Contents

As stated multiple times throughout this manuscript, spiking neural networks are considered as a low-power alternative to classical neural networks due to their sparse and binary operating function. But few works have proven these claims to be true on real-world machine learning applications. The focus of this thesis has been to develop the SNNs training methods and architectures in order to tackle more complex real-world automotive tasks, all in order to implement in the future these algorithms on embedded systems and leverage their energy efficiency.

To finally be able to characterize to what extent SNNs are more energy efficient than ANNs, we propose in this chapter an analytical estimation of the energy consumption of SNNs. This was done by designing an hardware-independent analytical metric relying not only on the synaptic operations, but also on the memory accesses and addressing mechanisms, often neglected as they are unfavorable to SNNs. This work is taken from the paper entitled *"An Analytical Estimation of Spiking Neural Networks Energy Efficiency"* (Lemaire, Cordone, et al., 2022), presented at the International Conference on Neural Information Processing in 2022.

This chapter is organized as follows: first, we discuss the comparisons of energy efficiency between SNNs and ANNs available in the literature. Second, we introduce our different metrics, leading to our energy consumption metric for SNNs and ANNs. Third, we apply our metric on networks designed to solve three different real-world classification tasks, showing that our SNNs are 6 to 8 times more efficient than their ANN counterparts while providing comparable performance. Finally, we conclude this chapter with guidelines on the design of efficient embedded SNNs and on the remaining research areas to tackle.

6.1 Energy efficiency of SNNs in the literature

To estimate the energy efficiency of SNNs, several comparisons between SNNs and ANNs have already been proposed in the literature. However, such comparisons are hardly generalizable since they focus on specific applications or hardware targets. Besides, the considered applications are often toy examples not representative of real-world artificial intelligence tasks. Therefore, another approach is to produce metrics to evaluate the energy consumption of both types of networks, based on their respective synaptic operations and activities.

In this section, we will review these two types of comparisons between SNNs and ANNs available in the literature, first those based on measurements and then those based on metrics.

6.1.1 Comparisons based on measurements

In the literature, a few comparisons between SNNs and ANNs have been made based on hardware measurements. Barchid, Mennesson, Eshraghian, et al. (2022) showed competitive results for SNNs, highlighting the influence of the spike encoding method on the accuracy and computational efficiency of the SNN. They compared the spiking and classical networks through a Resnet-18 like architecture on two classification frame-based datasets. They found that SNNs reached higher or

equivalent accuracy and energy efficiency. Khacef, Abderrahmane, and Miramond (2018) showed that an SNN that an SNN could reach twice the power and resource efficiency of an ANN, with an MLP on MNIST dataset targeting ASIC. However, those encouraging results are still very specific and thus hardly generalizable.

In a more holistic approach, Lemaire, Miramond, et al. (2022) performed an exploration of the design space (including encoding, learning method, level of parallelism) and showed that the benefits of SNNs depended on the considered case, which makes it difficult to establish general rules. In (Rueckauer, Bybee, et al., 2022), the authors demonstrated that SNNs running on a specialized hardware (Intel Loihi) resulted in better energy efficiency than equivalent ANNs on conventional hardware (CPU / GPU) for small topologies, but observed the opposite using larger CNNs. Once more, the conclusions depended on the studied case and could not be generalized. Albeit encouraging, those results are not sufficient to draw general conclusions regarding the savings offered by the spike paradigm, since they depend on the selected application, network hyperparameters and hardware targets.

Therefore, another approach consists in comparing the two types of neural networks through estimation metrics, taking a step back to produce more general conclusions.

6.1.2 Comparisons based on metrics

Most energy consumption metrics are based on the number of synaptic operations: accumulations (ACC) in the SNN and multiplication-accumulations (MAC) in the ANN. Those models have limitations: energy consumption is assimilated to the energy consumption of synaptic operations (Kundu et al., 2021), thus other factors (such as neuron addressing in multiplexed architecture or memory accesses) are often neglected. Moreover, the models usually do not take into account some specific mechanisms, like membrane potential leakage, reset and biases integration. In (Barchid, Mennesson, Eshraghian, et al., 2022), the authors proposed a metric based on synaptic operations only, and found great energy consumption savings for the SNN (up to 126× more efficient than the ANN baseline). In (Lemaire, Miramond, et al., 2022) the authors demonstrated that such simplistic metrics were not always coherent with actual energy consumption of circuits on FPGA. When taking memory into account, Deng et al. (2020) found equivalent energy consumption for SNNs and ANNs using various topologies on CIFAR-10. Additionally, Davidson and S. B. Furber (2021) measured a theoretical maximum spike rate of 1.72 to guarantee energy savings in the SNN based on a detailed metric, accounting for synaptic operations, memory accesses and activation broadcast. Those energy consumption models are

enlightening, but still fail to settle whether event-based processing is sufficient to increase energy efficiency. That is mostly because those metrics are too hardware specific, or do not take all significant sources of energy consumption into account.

In the present work, we propose a metric intended to be independent from low-level implementation choices, based on three types of operations encountered in SNNs and ANNs: synaptic operations, memory accesses and addressing operations for convolutions.

6.2 Proposed metrics

In this section, we will define a number of metrics enabling the count of accumulations operations (noted ACC) and of multiply-and-accumulate operations (noted MAC). These different metrics will serve as the basis of our final energy consumption metric, relying on the number of ACC and MAC.

6.2.1 Operational cost

We will now define the operational cost of SNNs and ANNs as the number of ACC and MAC resulting from the synaptic operations, handling separately convolutional layers and fully-connected layers.

Convolutional layers

For a convolution layer, the number of filters is defined by C_{out} and their size are noted $C_{\text{in}} \times H_{\text{kernel}} \times W_{\text{kernel}}$, where C, H and W stands for channel, height and width. The input and output of the layer are composed of a set of feature maps, with shapes $(C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}})$ and $(C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}})$ respectively. In the following we consider the padding mode "*same*" and 2D stride S_h, S_w. The equations describing the number of MAC and ACC operations in ANNs for convolution layers are summarised in Eq. 6.1.

$$\begin{aligned} \text{MAC}^{\text{ANN}}_{\text{OpsConv}} &= C_{\text{in}} \times H_{\text{kernel}} \times W_{\text{kernel}} \times C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}} \\ \text{ACC}^{\text{ANN}}_{\text{OpsConv}} &= C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}} \end{aligned} \tag{6.1}$$

In ANNs, the integration of dense input matrices requires a MAC operation for each element of the convolution kernels. That is described in the first row of Eq. 6.1. Additionally, the integration of synaptic biases as ACC operations, since it does not require multiplication with input activation. There is one bias per output neuron as shown in the second row of Eq. 6.1.

On the other hand, for an SNN the input activations are sparse binary matrices. Eq. 6.2 characterizes the number of MAC and ACC operations in the convolutional layers of SNNs, with the number of timesteps noted T.

$$\begin{aligned}
\text{MAC}^{\text{SNN}}_{\text{OpsConv}} &= T \times C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}} \\
\text{ACC}^{\text{SNN}}_{\text{OpsConv}} &= \theta_{\text{in}} \times \lceil \frac{H_{\text{kernel}}}{S_h} \rceil \times \lceil \frac{W_{\text{kernel}}}{S_w} \rceil \times C_{\text{out}} \\
&\quad + T \times C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}} \\
&\quad + \theta_{\text{out}}
\end{aligned} \tag{6.2}$$

The number of operations of the layer L depends on its number of input and output spikes, noted θ_{in} and θ_{out}. Since spikes are binary, they are integrated via ACC operations in contrast to ANNs. Each input spike causes one ACC operation per element of each filter, as shown in the first term of the second row of Eq. 6.2. The second term accounts for the bias added to each membrane potential at each timestep. The third term accounts for the membrane potential reset in the case neurons use *soft reset*, in which the threshold value is subtracted from the membrane potentials whenever an output spike is generated. This term would be equal to 0 if the neurons use *hard reset*, i.e the membrane potentials are reset to the value 0 whenever a spike is emitted. Additionally, SNNs may involve a membrane potential leakage (*i.e.* LIF neurons), which is modeled by additional MAC operations for each output neuron, and is repeated at each timestep. This is depicted in the first row of Eq. 6.2.

Fully-connected layers

The same reasoning is applied to fully-connected (FC) layers. For a given layer L the number of input and output neurons is noted N_{in} and N_{out} respectively. The equations for the number of MAC and ACC operations attributable to synaptic operations in FC layers are summarized in Eq. 6.3.

$$\begin{aligned}
\text{MAC}^{\text{ANN}}_{\text{OpsFC}} &= N_{\text{in}} \times N_{\text{out}} \\
\text{ACC}^{\text{ANN}}_{\text{OpsFC}} &= N_{\text{out}} \\
\text{MAC}^{\text{SNN}}_{\text{OpsFC}} &= N_{\text{out}} \times T \\
\text{ACC}^{\text{SNN}}_{\text{OpsFC}} &= \theta_{\text{in}} \times N_{\text{in}} \times N_{\text{out}} + T \times N_{\text{out}}
\end{aligned} \tag{6.3}$$

6.2.2 Memory accesses cost

Data flowing from and to the memory is an important sink of energy, especially when considering SNNs that have more memory accesses than ANNs due to the membrane potentials and the sequential operation over multiple timesteps. In this subsection, we will attempt to quantify the number of memory accesses of ANNs and SNNs.

In order to do that, multiple assumptions have to be made. Without these assumptions, results could vastly vary between different unconstrained hardware implementations. Each layer of the ANN is assumed to have its own local (non-shared) memory. As a result, activations need to be kept in memory (*i.e.* I/O buffers) for all layers. The data flow between layers of the SNN is assumed to be sparse and asynchronous. Therefore, messages of incoming spikes must be buffered in a FIFO queue for each layer. Additionally, the SNN must keep the membrane potentials for all layers between timesteps. In both cases, we assume that all the memory is akin to local SRAM, including weights in order to support a reconfigurable architecture. Additionally, there is no local caching in a register bank. Only registers for the operands and a local accumulator are present and are excluded from this evaluation. All data is assumed to be represented with the same number of bits, including the messages describing a spike.

Once these assumptions have been made, we can describe each read or write operation for each layer of the two types of neural networks. Equations are provided for a single input for the ANN and a single timestep for the SNN, as matter of simplicity of notations. Each equation of the SNN must therefore be multiplied by the number of timesteps T.

Inputs Read operations The number of read operations attributed to the inputs of a layer of an ANN is computed in Eq. 6.4. For a convolutional layer, each output position matches with read operations from all input channels and all positions for which the kernel applies. For an FC layer, the number of read operations for the input data is equal to the number of inputs N_{in}.

$$\begin{aligned} \text{InRead}_{\text{Conv}}^{\text{ANN}} &= C_{\text{in}} \times C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}} \times W_{\text{kernel}} \times H_{\text{kernel}} \\ \text{InRead}_{\text{FC}}^{\text{ANN}} &= N_{\text{in}} \end{aligned} \tag{6.4}$$

For the SNN, the read operations in the queue directly depends on the number of incoming spikes θ_{in} and must be measured during inference:

$$\text{InRead}^{\text{SNN}} = \theta_{\text{in}} \tag{6.5}$$

Parameters Read operations Additional read operations occur to access the parameters of the network. For an ANN, their estimation is depicted in Eq. 6.6. In a non-spiking convolutional layer, each output position matches with read operations for all the weights in all filters associated to all input channels. The biases generate additional reads for all output positions and all filters. For an FC layer, every weight generates a read operation, corresponding to the product between all output neurons N_{out} and all inputs neurons N_{in}. The biases cause additional reads for all output neurons.

$$\begin{aligned} \text{ParamRead}_{\text{Conv}}^{\text{ANN}} &= (C_{\text{in}} \times W_{\text{kernel}} \times H_{\text{kernel}} + 1) \times C_{\text{out}} \times W_{\text{out}} \times H_{\text{out}} \\ \text{ParamRead}_{\text{FC}}^{\text{ANN}} &= (\theta_{\text{in}} + 1) \times N_{\text{out}} \end{aligned} \tag{6.6}$$

In a spiking convolutional layer, all received spikes θ_{in} will trigger a read for all output filters and all associated output positions (i.e. of the dimensions of the kernel). Biases for all output positions and filters must still be read. For an FC layer, the number of read operations for parameters is similar to an ANN except that weights are only read for all input spikes θ_{in}. Biases must still be read for all output neurons N_{out}.

$$\begin{aligned} \text{ParamRead}_{\text{Conv}}^{\text{SNN}} &= \theta_{\text{in}} \times C_{\text{out}} \times W_{\text{kernel}} \times H_{\text{kernel}} \\ &\quad + C_{\text{out}} \times W_{\text{out}} \times H_{\text{out}} \\ \text{ParamRead}_{\text{FC}}^{\text{SNN}} &= N_{\text{in}} \times N_{\text{out}} + N_{\text{out}} \end{aligned} \tag{6.7}$$

Membrane potentials Read operations There is no membrane potentials to update in an ANN so there is no associated read operations. But spiking neural networks need to access membrane potentials of each layer to determine if neurons will spike.

In a spiking convolutional layer, the membrane potentials corresponding to all output positions affected by each input (i.e. of the dimensions of the kernel) in all filters must be read in order to update them. Biases need to be applied separately and therefore generate an additional read operation at each timestep for all output positions and all filters. For FC layers, the potentials of all output neurons are read for each input. Biases are applied separately and therefore generate an additional read operation at each timestep for all output neurons.

$$\begin{aligned} \text{PotRead}_{\text{Conv}} &= \theta_{\text{in}} \times C_{\text{out}} \times W_{\text{kernel}} \times H_{\text{kernel}} \\ &\quad + C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}} \\ \text{PotRead}_{\text{FC}} &= (\theta_{\text{in}} + 1) \times N_{\text{out}} \end{aligned} \tag{6.8}$$

Outputs Write operations In a non-spiking convolutional layer, each output position in all filters requires a write operation. For an FC layer, each output neuron requires a write operation. In both cases, the output is assumed to be fully computed in the local accumulator, including bias, before being written to RAM.

$$\begin{aligned} \text{OutWrite}_{\text{Conv}}^{\text{ANN}} &= C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}} \\ \text{OutWrite}_{\text{FC}}^{\text{ANN}} &= N_{\text{out}} \end{aligned} \tag{6.9}$$

For an SNN, the write operations to the queue directly depends on the number of generated spikes θ_{out} and must be measured during inference:

$$\text{OutWrite}^{\text{SNN}} = \theta_{\text{out}} \tag{6.10}$$

Membrane potentials Write operations Again, ANNs have no membrane potentials therefore no associated write operations.

In an spiking convolutional layer, the membrane potentials corresponding to all output positions affected by each input (i.e. of the dimensions of the kernel) in all filters must be written to in order to update them. Additionally, the biases must also be written separately to the potentials for all output positions and all filters at each new timestep. For an FC layer, the potentials of all output neurons must be written to for each input. Additionally, the biases must also be written to the potentials for all output neurons separately at each new timestep.

$$\begin{aligned} \text{PotWrite}_{\text{Conv}} &= \theta_{\text{in}} \times C_{\text{out}} \times W_{\text{kernel}} \times H_{\text{kernel}} \\ &\quad + C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}} \\ \text{PotWrite}_{\text{FC}} &= \theta_{\text{in}} \times N_{\text{out}} + N_{\text{out}} \end{aligned} \tag{6.11}$$

6.2.3 Addressing cost

In this subsection, we evaluate the cost of addressing in ANNs and SNNs. The first uses dense processing, in which all input synapses are stimulated at the same time. On the other hand, the second uses sparse processing, in which synapses are sparsely stimulated across time. Let us begin with convolution layers. In order to simplify the following matter, we consider convolutions with a "*same*" padding (input and output feature maps of same sizes) and a stride of S. In ANNs, a kernel scans all its possible positions (depending on padding, stride...) on the input sample and generates a dense output feature map. In such dense convolutions, computation is performed sequentially and addresses can be computed by incrementing only an index (by 1

or S) assuming the memory is contiguous and ordered the same way it is processed. Thus, one index runs through the input, one index runs through the output and one index runs through the weights. In SNNs, sparse convolutions are performed asynchronously upon reception of input spikes, thus the kernel positions (*i.e.* output neuron addresses) must be calculated each time a spike is received. In a sparse representation, computation is performed non-sequentially with no prior knowledge of which output position is affected by an incoming spike. Computing the initial output position requires two multiplications. Thereafter, the computation of the remaining positions are computed by incrementing an index assuming the memory is contiguous and ordered as for ANNs. There is only one index running through the kernel weights. The cost of addressing in number ACC and MAC operations in spiking and formal convolution layer can then be summarized in Equations 6.12:

$$\begin{aligned} \mathrm{ACC}^{\mathrm{ANN}}_{\mathrm{AddrConv}} &= C_{\mathrm{in}} \times H_{\mathrm{in}} \times W_{\mathrm{in}} + C_{\mathrm{out}} \times H_{\mathrm{out}} \times W_{\mathrm{out}} \\ &\quad + C_{\mathrm{out}} \times H_{\mathrm{kernel}} \times W_{\mathrm{kernel}} \\ \mathrm{MAC}^{\mathrm{SNN}}_{\mathrm{AddrConv}} &= \theta_{\mathrm{in}} \times 2 \\ \mathrm{ACC}^{\mathrm{SNN}}_{\mathrm{AddrConv}} &= \theta_{\mathrm{in}} \times C_{\mathrm{out}} \times H_{\mathrm{kernel}} \times W_{\mathrm{kernel}} \end{aligned} \tag{6.12}$$

The same reasoning is applied to fully-connected layers. In ANNs, one index runs through the input, and another index runs through the output. In SNNs, one single index runs through the output upon receiving an input spikes. This yields Eq. 6.13.

$$\begin{aligned} \mathrm{ACC}^{\mathrm{ANN}}_{\mathrm{AddrFC}} &= N_{\mathrm{in}} + N_{\mathrm{out}} \\ \mathrm{ACC}^{\mathrm{SNN}}_{\mathrm{AddrFC}} &= \theta_{\mathrm{in}} \times N_{\mathrm{out}} \end{aligned} \tag{6.13}$$

Where N_{in} and N_{out} are resepctively the number of input and output neurons.

6.2.4 Energy consumption metric

In this section, we combine the equations obtained for computation, memory accesses and addressing in a global Energy evaluation metric. For this purpose, we multiply the energy cost of each operation by its number of occurrences, according to the metrics presented in the preceding sections. Our model can be summarized as follows:

$$E = E_{\mathrm{mem}} + E_{\mathrm{ops}} + E_{\mathrm{addr}} \tag{6.14}$$

Where E_{mem} is the energy consumption of memory accesses, E_{ops} that of synaptic operations, and E_{addr} that of addressing mechanisms.

E_{mem} is detailed in Equation 6.15, with E_{ReadRAM} and E_{WriteRAM} the energy for a single read and a single write operation in RAM, respectively. In our future computations, we will assume that $E_{\text{ReadRAM}} = E_{\text{WriteRAM}}$ for simplicity purpose.

$$\begin{aligned} E_{\text{mem}}^{\text{ANN}} &= (\text{InRead}^{\text{ANN}} + \text{ParamRead}^{\text{ANN}}) \times E_{\text{ReadRAM}} \\ &\quad + \text{OutWrite}^{\text{ANN}} \times E_{\text{WriteRAM}} \\ E_{\text{mem}}^{\text{SNN}} &= (\text{InRead}^{\text{SNN}} + \text{ParamRead}^{\text{SNN}} + \text{PotRead}) \times E_{\text{ReadRAM}} \\ &\quad + (\text{OutWrite}^{\text{SNN}} + \text{PotWrite}) \times E_{\text{WriteRAM}} \end{aligned} \tag{6.15}$$

The energy consumption of the synaptic operations E_{ops} and that of addressing operations E_{addr} are expressed by Equations 6.16 and 6.17, with E_{ADD} and E_{MUL} are the energy cost of single additions and multiplications respectively:

$$\begin{aligned} E_{\text{ops}}^{\text{ANN}} &= (E_{\text{ADD}} + E_{\text{MUL}}) \times \text{MAC}_{\text{ops}}^{\text{ANN}} + E_{\text{ADD}} \times \text{ACC}_{\text{ops}}^{\text{ANN}} \\ E_{\text{ops}}^{\text{SNN}} &= (E_{\text{ADD}} + E_{\text{MUL}}) \times \text{MAC}_{\text{ops}}^{\text{SNN}} + E_{\text{ADD}} \times \text{ACC}_{\text{ops}}^{\text{SNN}} \end{aligned} \tag{6.16}$$

$$\begin{aligned} E_{\text{addr}}^{\text{ANN}} &= (E_{\text{ADD}} + E_{\text{MUL}}) \times \text{MAC}_{\text{addr}}^{\text{ANN}} + E_{\text{ADD}} \times \text{ACC}_{\text{addr}}^{\text{ANN}} \\ E_{\text{addr}}^{\text{SNN}} &= (E_{\text{ADD}} + E_{\text{MUL}}) \times \text{MAC}_{\text{addr}}^{\text{SNN}} + E_{\text{ADD}} \times \text{ACC}_{\text{addr}}^{\text{SNN}} \end{aligned} \tag{6.17}$$

We draw the energy consumption of single operations (addition, multiplication and memory accesses) from the literature (Jouppi et al., 2021), for 45nm CMOS technology, other values for other technology could be used in the future. For addition and multiplication with 32-bit integers, we use respectively $0.1pJ$ and $3.1pJ$. For SRAM memory accesses, we compute a linear interpolation function based on 3 particular values (taken from Jouppi et al., 2021): 8 kB ($10pJ$), 32 kB ($20pJ$) and 1 MB ($100pJ$). This function enables to compute the energy cost of a memory access knowing the memory size (*i.e.* knowing the network hyperparameters).

6.3 Experiments and results

With our energy consumption metric now defined, we will now apply it on three different datasets, each representative of a certain type of data to judge if SNNs are more energy efficient than comparable ANNs on these real-world machine learning tasks. To provide a fair comparison with ANNs, we will place the BN layers after the convolutions even in our SNNs, even if we have shown in the previous chapter that the BN-CONV order could further increases the performance of our SNNs.

6.3.1 Datasets and models

Static frame-based data: CIFAR-10

The CIFAR-10 dataset is made of 60000 32x32 RGB images representing 10 classes. For the SNN, CIFAR-10 samples are repeated as input over $T = 4$ timesteps, using direct encoding (Y. Kim et al., 2022), repeating the input as many times as the number of timesteps.

For this task, we use a VGG-16 architecture described in (Simonyan and Zisserman, 2015). We dropped the max pooling layers by using a stride of 2 in their preceding convolution and we added batch normalization layers after each convolution. The ANN uses regular ReLU activation layers while the SNN uses IF neurons.

Dynamic frame-based data: Google Speech Commands V2

The Google Speech Commands V2 dataset (GSC) is composed of audio signals sampled at 16 kHz of 1-second recordings of 35 spoken keywords. We performed data augmentation by randomly changing the speed of the raw audio signals, padding or truncating the samples when necessary.

The raw spectograms are preprocessed to obtain images that can be fed to CNNs. We used 10 MFC Coefficients, FFT of size 1024, a window size of 640 with a hop of 320, and a padding of 320 on both sides. This results in a 48x10 image interpreted as 1D vector of size 48 with 10 channels. For the SNN, we again used direct encoding, but without the need of repeating the input as we divided this temporal data in $T = 2$ timesteps, each of size 24.

To tackle this classification task, we designed a 4-layers CNN with the following topology: 48c3 - 48c3 - 96c3 - 35c1. Each convolutional layer has a stride of 1 and was followed by a batch normalization layer.

Event-based data: Prophesee N-CARS

The Prophesee N-CARS dataset (Sironi et al., 2018) is a classification dataset composed of 24k samples of length 100ms captured with a Prophesee GEN1 event camera mounted behind the windshield of a moving car. The samples represent either a car or background. We resized all the samples to a size of 64×64 pixels using nearest neighbor interpolation to keep our inputs binary. For SNNs, we divided each sample in $T = 5$ timesteps, while all the events were summed into a single frame for the CNN.

We designed Tiny VGG-11, a variant of the classical VGG-11 architecture that uses 4 times fewer channels in each convolution layers, reducing the number of parameters and calculations. We dropped the max pooling layers by using a stride of 2 in their preceding convolution and we added batch normalization layers after each convolution. Finally, we replaced the final 3 fully-connected layers by our output layer described in the following Section 6.3.2.

6.3.2 Organization of the output layer

Both ANN and SNN use a traditional final fully-connected layer for static frame-based data. On the other hand, our models for dynamic frame-based and event-based data use a specific final classification layer as the feature maps are not sufficiently reduced to be flattened before the final fully-connected layer.

Indeed, we followed the approach used in Chapter 3. The output layer of our SNNs is simply composed of a 1×1 convolution outputting $num\_classes$ channels, a batch normalization layer and a final layer of LIF neurons. The final predictions are then obtained by summing all output spikes first in the spatial dimension and time dimension. We therefore obtain a tensor with a spatial size of 1×1 with $num\_classes$ channels, which is equivalent to the output of conventional fully-connected layers. The 1D convolution in the final layer enables to avoid the use of *e.g.* average pooling to reduce the spatial dimension as it would be incompatible with spikes computations. We used the same approach for the equivalent ANNs but without summation along the time axis since it does not exist.

6.3.3 Results

We trained our ANNs using PyTorch, and our SNNs using SpikingJelly (Fang, Y. Chen, et al., 2020) with surrogate gradient learning. The models were trained over 50 epochs for GSC and N-CARS, and 300 for CIFAR-10. All presented results represent the average over 5 runs. The performance of our networks were

measured by their classification accuracy. We also measured the spike rate of our SNNs, corresponding to the average number of spikes per synapse in the network. Since computations are only performed when there is a spike, this has a direct impact on the SNN energy consumption within our metric. These results are summarized in Table 6.1.

Table 6.1: Accuracy and spike rate comparisons between our proposed SNNs and CNNs on CIFAR-10, Google Speech Commands V2, and Prophesee N-CARS.

Dataset	Network	Network type	#Params	Acc.	Spike Rate
CIFAR-10	VGG-16	ANN (ReLU)	15.2M	0.951	–
		SNN (IF)		0.884	0.10
GSC	4-layers CNN	ANN (ReLU)	29k	0.936	–
		SNN (LIF)		0.918	0.14
N-CARS	Tiny VGG-11	ANN (ReLU)	356k	0.934	–
		SNN (LIF)		0.935	0.08

For dynamic and event data, SNNs are able to reach equivalent or close accuracies to their ANN counterparts, a result rarely shown experimentally before on these datasets. On the CIFAR-10 dataset, our SNN reaches lower accuracy than the ANN. This is coherent with state of the art results, but should be improved in future works. Still, all of our SNNs reach these performance while having a very low spike rate, on average each neuron of a model spikes less than 0.14 times per inference for the three datasets.

Using the metrics proposed in Section 6.2, we were able to precisely estimate the energy consumption of our models. For the memory accesses, we consider that the weights, biases and potentials are stored using 32-bits. In our model, I/O buffers between SNN layers are FIFOs able to store 1000 32-bit elements. This assumption has been validated through hardware simulation using the SPLEAT architecture (Abderrahmane et al., 2022). On the other hand, the full feature maps are stored in SRAMs between ANN layers. The results are illustrated in Fig. 6.1 and detailed in Table 6.2.

As expected, the SNNs synaptic operations cost much less energy than those of the ANNs since they are mainly composed of ACC operations. On the other hand, addressing energy cost is a little higher in SNNs, as more computations are needed to process sparse inputs.

But the total energy consumption is dominated by the cost of memory accesses, which is yet unduly neglected in most metrics of the literature. While SNNs have additional memory accesses for updating the neuron potentials, it requires fewer

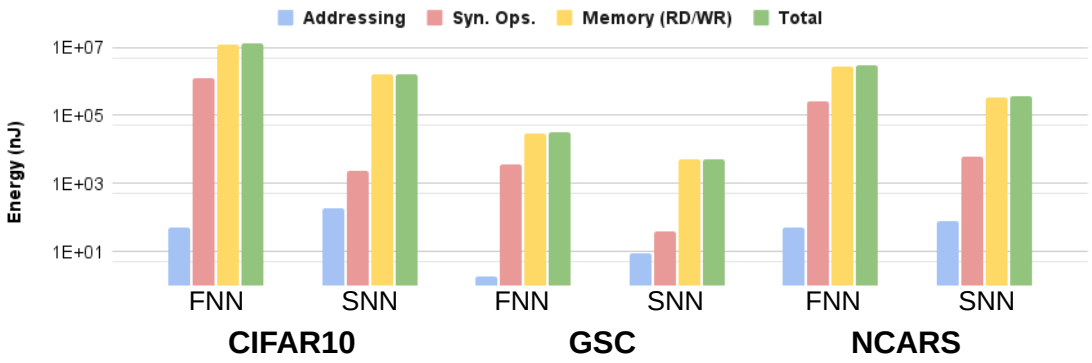


Figure 6.1: Estimation of energy consumption for 45nm CMOS technology.

Table 6.2: Energy consumption estimations of our SNNs and their equivalent FNNs on the 3 datasets for 45nm CMOS technology.

		CIFAR10		GSC		NCARS	
		SNN	**FNN**	**SNN**	**FNN**	**SNN**	**FNN**
Memory Accesses (nJ)	**Potentials**	1.12E+6	–	3.50E+3	–	2.69E+5	–
	Weights	4.60E+5	5.59E+6	1.73E+3	1.46E+4	8.10E+4	1.11E+6
	Bias	2.79E+2	6.97E+1	6.00E+0	3.00E+0	3.71E+1	7.43E+0
	In/Out	7.65E+2	6.09E+6	3.90E+1	1.50E+4	2.88E+2	1.54E+6
	Total	1.58E+6	1.17E+7	5.27E+3	2.96E+4	3.50E+5	2.64E+6
Synaptic Op. (nJ)		2.41E+3	1.29E+06	4.08E+1	3.53E+3	6.15E+3	2.67E+5
Addressing (nJ)		1.97E+2	4.94E+1	9.28E+0	1.93E+0	5.05E+1	7.65E+1
Total (nJ)		1.58E+6	1.24E+7	5.32E+3	3.32E+4	3.57E+5	2.86E+6
$E^{\mathbf{FNN}}/E^{\mathbf{SNN}}$		**8.19**		**6.22**		**8.17**	

memory accesses for the weights and the I/O. Moreover, the size of I/O buffers are often much smaller in SNNs than in ANNs, since the first only requires FIFOs of 1000 elements whereas the second requires storing the full feature maps.

In the end, the total energy consumption of our SNNs is between 6.22 and 8.19 times lower than their ANN counterparts, a promising result for the development of embedded SNNs on specialized hardware.

6.4 Conclusion

Spiking neural networks are often presented as low-power alternatives to classical ANNs. By proposing a generic and accurate energy estimation metric that is independent from low-level implementation choices and hardware targets, we show

that SNNs are roughly 8 times more energy-efficient than ANNs on three different classification tasks, with comparable accuracies.

Furthermore, our analytical estimation also provides guidelines for the design of efficient spiking neural networks and/or of neuromorphic hardware. As memory accesses represent the majority of the energy consumption of an SNN, it is on this essential point that both SNNs developers and manufacturers of neuromorphic hardware should focus.

6.4.1 Number of timesteps

As all memory accesses have to be repeated every timestep, the most important parameter for SNNs energy consumption is the number of timesteps T. As mentioned several times in this manuscript, this is why we do not consider SNNs built by conversion as a viable approach, as they require a large number of timesteps to replicate good ANNs performances. The direct training of SNNs, enabled e.g. by the surrogate gradient learning approach, is for us much more promising and flexible. We have shown that we can directly train SNNs on a few number of timesteps on temporal data, leading to low energy consumption estimations. Regarding frame-based data, achieving state-of-the-art performance with a low number of timesteps is still an active research area, but the direct encoding enabled once again by the surrogate gradient learning method is promising, as long as we have the possibility on the specialized hardware to do MAC operations on the first layer of the network.

6.4.2 Sparsity

The memory accesses in SNNs depend often on the activity of each layer, proving that sparse spiking neural networks are important to achieve efficient SNNs. But today, the sparsity in SNNs is not really controlled and is rather a side effect measured at the end of a successful training. Decreasing the activity of an SNN (i.e. its spike rate) yet allows substantial energy gains, and we can only encourage researchers to study efficient ways of increasing SNNs sparsity while not penalizing the already difficult learning process.

In this thesis, we have shown that processing event data, which is highly sparse by nature, allows to preserve a low activity throughout the network. We also hypothesize that the use of higher weight decay values will result in higher sparsity, as the learned weights will have smaller values and will reach the potentials to emit spikes less often. We can also mention the addition of the number of spikes emitted directly in the network training loss, as proposed in (Zimmer et al., 2019), which also seems to be an effective way to decrease the network activity.

6.4.3 Quantization

Finally, an important aspect of the embedding of SNNs that we have not tackled in this thesis is quantization. In SNNs, the activations don't require quantization as they are already binary, but the weights and the membrane potentials can be quantized. Quantizing weights and potentials has a direct impact on the energy consumption, as it would require less memory accesses than using full precision weights. Quantized values also require less energy for addition and multiplication. We know as a fact that some of the SNNs presented in this thesis can be quantized on 16-bit values with a nearly lossless performance, but a lot of research still has to be conducted to guarantee the performance of SNNs when targeting 8-bit or lower representation.

7 Conclusion and future works

Contents

7.1 Conclusion

With the multiplication of artificial intelligence algorithms embedded in cars, it becomes essential that they have a low energy consumption. To avoid accidents and potentially save lives, they need to be high-performance while having a low latency. In this thesis, we have studied how an end-to-end biologically-inspired approach processing event data with spiking neural networks can be used to design embedded automotive intelligence artificial algorithms that are high-performance, fast and energy-efficient.

Spiking neural networks, usually presented as low-power alternatives to classical neural networks, were however unable to compete with them in terms of performance outside of toy examples. As their particular operation requires input data in the form of spikes, the common approach to process visual information was to encode classical images in spikes. Instead, we focused in this thesis on the processing of

event data directly with spiking neural networks, as events can be interpreted as spikes and thus represent an ideal input for SNNs.

The objective of this thesis have been to increase the performance of spiking neural networks for event data processing, with the ambition to take SNNs to the next level. Using the latest advances in the SNNs and event-based processing fields, we have develop new training methods and new architectures for SNNs that have enabled the tackling of real-world automotive computer vision tasks such as object detection, previously unattainable by SNNs.

While the major focus of this thesis has been on the software side, this improved performance of SNNs is only useful if we are able to implement them on embedded systems and have a lower energy consumption than equivalent classical neural networks. As a first step towards this end goal, we have provided an hardware-independent analytical estimation of the energy consumption of any SNN, showing that SNNs are now able to reach comparable or better results than equivalent ANNs on three different machine learning while consuming 6 to 8 times less energy.

7.2 Future works

Our results, in terms of performance and of energy consumption, lay the groundwork for a multitude of future works that could ultimately lead to the use of SNNs in embedded automotive applications.

7.2.1 Short-term perspectives: further improving the performance

The results obtained in Chapter 5 prove that SNNs are able to tackle object detection problems, but there is still a significant performance gap with ANNs. Based on our works, we can propose several leads that we believe will further improve our results:

- Training object detection models with TBPTT with a shorter time resolution, e.g. 20ms. We used 50ms for computational reasons, and since we use binary voxel grids / voxel cubes, a lot of information can be lost on long timesteps. Furthermore, since bounding boxes ground truths are attached to a timestep whatever the moment they occur in the interval could lead to slightly misplaced ground truths that could impair the training, and this would also be solved by the use of shorter timesteps.

- Training bigger object detection models with TBPTT. Even though bigger networks would be more difficult to embed, the neural networks field has consistently used very large networks to improve performance on difficult tasks, to then design reduced networks for embedded systems.

- Improve the TBPTT training algorithm. An already stated limitation of our TBPTT training approach is that we have to detach the whole computational graph even though we trained only on certain samples of a batch. This leads to the forgetting of past states of samples that did not participate in the backward pass. We believe that improving the TBPTT training algorithm to only detach the past states on samples concerned by a backward pass would dramatically improve the results achieved on object detection.

- Develop new SNNs architectures with residual connections. Residual connections are essential to the performance of classical neural networks. We proposed in this thesis to use concatenation-based residual connections, providing performance boosts but also increasing the parameters count and operational complexity. The several addition-based residual connections in SNNs proposed in the literature have not yet been proven possible on neuromorphic architectures, making their actual relevance uncertain.

- Tackle the object detection event dataset Prophesee 1MPX, which is larger in terms of resolution, classes (7) and ground truth bounding boxes (100 times more) than the Prophesee GEN1 one. With ground truth bounding boxes at best every 16.66 ms (60Hz), the techniques implemented to learn on Prophesee GEN1 with TBPTT will be even more effective.

7.2.2 Middle-term perspectives: other tasks and embedding on neuromorphic hardware

In this thesis, we have focused on the object detection task, which represents a good example of a complex automotive embedded application. But everything that has been proposed in this manuscript to increase the performance of SNNs on event data can be applied to other computer vision tasks, such as depth estimation, segmentation, optical flow estimation, or even to other types of data as we did in Chapter 6 by tackling a problem of keyword spotting with SNNs.

And once the performance of SNNs is deemed sufficient for real-world applications, it is time to embed them into specialized or neuromorphic hardware.

A probably unavoidable step towards the embedding of SNNs is quantization, mostly absent from our thesis work. While SNNs operate on binary spikes, they still rely on floating point weights and potentials, and as we've seen in Chapter 6, the number of bits on which they are encoded has a direct influence on the energy consumption of the network, but also on the latency and on the size of the network considered. While several post-quantization schemes for SNNs are already available in the literature, we think that developing new Quantization-Aware Training (QAT) methods for SNNs will bring better performance when considering the encoding of floating point values on a low number of bits. This will ensure that the performance achieved during training will be met in real-life conditions.

In the same vein, several software toolchains have to be developed to port SNNs trained in modern deep learning frameworks into neuromorphic hardware. Each hardware having its specificities, it will necessarily constrain the operations used, the size of the networks considered and the resulting performance. The sooner these toolchains are available, the sooner SNNs could be developed around these now defined constraints. This is important as it will lead to SNNs results reproducible on embedded hardware and not only in software simulation.

7.2.3 Long-term perspectives: automotive applications

Finally, the long-term perspectives depend mainly on the development and availability of cheap neuromorphic hardware and event cameras. Once the deployment of powerful SNNs on neuromorphic hardware becomes a reality, a multitude of intelligent low-power applications could emerge, particularly in cars.

Indeed, we already illustrated the object detection task with event cameras, that could be used in cars to detect surrounding dangers faster than traditional sensors, and in more difficult conditions. As it needs to run continuously at a high rate, the energy gains would be substantial. Similarly, SNNs processing event data could then be used for other perception tasks: lanes detection (lane keeping assist), distance estimation (auto-pilot in traffic jams), blind spot dangers detection.

Another interesting use of event sensors could be in-cabin: EU legislations will soon require an in-cabin camera to monitor the driver attention installed in all new vehicles, and for privacy reasons some customers would be more comfortable being filmed by an event camera rather than a traditional camera. An in-cabin monitoring algorithm would be used to detect fatigue, inattention or even violent conduct ; and must run continuously and at a high rate: the possible energy gains could be, once again, significant.

More generally, high-performance SNNs embedded on neuromorphic hardware could become the go-to solution for embedded applications, enabling the introduction of intelligent algorithms in a variety of everyday products at a low energy cost. The next few years will show whether or not spiking neural networks can fulfill their potential.

References

[1] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. en. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133 (cit. on p. 7).

[2] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain”. In: *Psychological Review* 65 (1958), pp. 386–408 (cit. on p. 7).

[3] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. en. In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366 (cit. on p. 7).

[4] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536 (cit. on p. 7).

[5] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. en. July 1986 (cit. on p. 7).

[6] Yann Le Cun. “Modèles connexionnistes de l’apprentissage”. These de doctorat. Paris 6, Jan. 1987 (cit. on p. 7).

[7] D. H. Hubel and T. N. Wiesel. “Receptive fields of single neurones in the cat’s striate cortex”. In: *The Journal of Physiology* 148.3 (Oct. 1959), pp. 574–591 (cit. on p. 7).

[8] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. en. In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202 (cit. on p. 7).

[9] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324 (cit. on p. 7).

[10] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. en. In: *Proceedings of the 32nd International Conference on Machine Learning*. June 2015, pp. 448–456 (cit. on p. 8).

[11] Kaiming He et al. “Deep Residual Learning for Image Recognition”. en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA, June 2016, pp. 770–778 (cit. on p. 8).

[12] Christian Szegedy et al. “Inception-v4, inception-ResNet and the impact of residual connections on learning”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI’17. San Francisco, California, USA, Feb. 2017, pp. 4278–4284 (cit. on p. 8).

[13] Mingxing Tan and Quoc Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. en. In: *Proceedings of the 36th International Conference on Machine Learning*. May 2019, pp. 6105–6114 (cit. on p. 8).

[14] A. L. Hodgkin and A. F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of Physiology* 117.4 (Aug. 1952), pp. 500–544 (cit. on p. 9).

[15] E.M. Izhikevich. “Which model to use for cortical spiking neurons?” In: *IEEE Transactions on Neural Networks* 15.5 (Sept. 2004), pp. 1063–1070 (cit. on p. 9).

[16] L. F. Abbott. “Lapicque’s introduction of the integrate-and-fire model neuron (1907)”. eng. In: *Brain Research Bulletin* 50.5-6 (Dec. 1999), pp. 303–304 (cit. on p. 10).

[17] Xiang Cheng et al. “LISNN: Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition”. en. In: vol. 2. July 2020, pp. 1519–1525 (cit. on p. 11).

[18] Daniel Auge et al. “A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks”. en. In: *Neural Processing Letters* 53.6 (Dec. 2021), pp. 4693–4710 (cit. on p. 11).

[19] E. D. Adrian and Y. Zotterman. “The impulses produced by sensory nerve endings: Part 3. Impulses set up by Touch and Pressure”. eng. In: *The Journal of Physiology* 61.4 (Aug. 1926), pp. 465–483 (cit. on p. 12).

[20] S. Thorpe, D. Fize, and C. Marlot. “Speed of processing in the human visual system”. eng. In: *Nature* 381.6582 (June 1996), pp. 520–522 (cit. on p. 12).

[21] Roland S. Johansson and Ingvars Birznieks. “First spikes in ensembles of human tactile afferents code complex spatial fingertip events”. en. In: *Nature Neuroscience* 7.2 (Feb. 2004), pp. 170–177 (cit. on p. 12).

[22] Simon Thorpe and Jacques Gautrais. “Rank Order Coding”. en. In: *Computational Neuroscience: Trends in Research, 1998*. Ed. by James M. Bower. Boston, MA, 1998, pp. 113–118 (cit. on p. 12).

[23] Bodo Rueckauer and Shih-Chii Liu. “Conversion of analog to spiking neural networks using sparse temporal coding”. In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. May 2018, pp. 1–5 (cit. on p. 12).

[24] Seongsik Park et al. “T2FSNN: deep spiking neural networks with time-to-first-spike coding”. In: *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*. DAC ’20. Virtual Event, USA, July 2020, pp. 1–6 (cit. on p. 12).

[25] Saeed Reza Kheradpisheh and Timothée Masquelier. “Temporal Backpropagation for Spiking Neural Networks with One Spike per Neuron”. In: *International Journal of Neural Systems* 30.06 (June 2020), p. 2050027 (cit. on p. 12).

[26] Wenzhe Guo et al. “Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems”. In: *Frontiers in Neuroscience* 15 (2021) (cit. on p. 12).

[27] Edgar Lemaire. “Modélisation et exploration d’architectures neuromorphiques pour les systèmes embarqués haute-performance”. These de doctorat. Université Côte d’Azur, Mar. 2022 (cit. on pp. 12, 13).

[28] Youngeun Kim et al. “Rate Coding Or Direct Coding: Which One Is Better For Accurate, Robust, And Energy-Efficient Spiking Neural Networks?” In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2022, pp. 71–75 (cit. on pp. 13, 113).

[29] Minhao Yang et al. “A 0.5V 55µW 64×2-channel binaural silicon cochlea for event-driven stereo-audio sensing”. In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. Jan. 2016, pp. 388–389 (cit. on p. 13).

[30] 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)*. July 2015, pp. 1–8 (cit. on p. 14).

[31] Abhronil Sengupta et al. “Going Deeper in Spiking Neural Networks: VGG and Residual Architectures”. In: *Frontiers in Neuroscience* 13 (2019) (cit. on p. 14).

[32] Amirhossein Tavanaei et al. “Deep learning in spiking neural networks”. en. In: *Neural Networks* 111 (Mar. 2019), pp. 47–63 (cit. on p. 14).

[33] Jianhao Ding et al. “Optimal ANN-SNN Conversion for Fast and Accurate Inference in Deep Spiking Neural Networks”. In: *Int. Joint Conf. on Artif. Intell.* 2021, pp. 2328–2336 (cit. on p. 14).

[34] S. Song, K. D. Miller, and L. F. Abbott. “Competitive Hebbian learning through spike-timing-dependent synaptic plasticity”. eng. In: *Nature Neuroscience* 3.9 (Sept. 2000), pp. 919–926 (cit. on pp. 14, 15).

[35] Taro Toyoizumi et al. “Spike-timing Dependent Plasticity and Mutual Information Maximization for a Spiking Neuron Model”. In: *Advances in Neural Inf. Process. Syst.* Vol. 17. 2004 (cit. on p. 14).

[36] Gopalakrishnan Srinivasan et al. “Training Deep Spiking Neural Networks for Energy-Efficient Neuromorphic Computing”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2020, pp. 8549–8553 (cit. on p. 15).

[37] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, et al. “STDP-based spiking deep convolutional neural networks for object recognition”. en. In: *Neural Networks* 99 (Mar. 2018), pp. 56–67 (cit. on p. 15).

[38] Milad Mozafari et al. “Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks”. en. In: *Pattern Recognition* 94 (Oct. 2019), pp. 87–95 (cit. on p. 15).

[39] Sander M. Bohte, Joost N. Kok, and Han La Poutré. “Error-backpropagation in temporally encoded networks of spiking neurons”. In: *Neurocomputing* 48.1 (Oct. 1, 2002), pp. 17–37 (cit. on pp. 15, 16).

[40] Eric Hunsberger and Chris Eliasmith. *Spiking Deep Networks with LIF Neurons*. Oct. 29, 2015. arXiv: `1510.08829[cs]` (cit. on pp. 16, 20).

[41] Emre O. Neftci, Charles Augustine, et al. “Event-Driven Random Back-Propagation: Enabling Neuromorphic Deep Learning Machines”. In: *Frontiers in Neuroscience* 11 (2017) (cit. on p. 16).

[42] Romain Zimmer et al. *Technical report: supervised training of convolutional spiking neural networks with PyTorch*. Nov. 2019 (cit. on pp. 17, 18, 39, 43, 44, 53, 117).

[43] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. “Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks”. In: *IEEE Signal Processing Magazine* 36.6 (Nov. 2019), pp. 51–63 (cit. on p. 16).

[44] Sander M. Bohte. “Error-Backpropagation in Networks of Fractionally Predictive Spiking Neurons”. en. In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. by Timo Honkela et al. Lecture Notes in Computer Science. Berlin, Heidelberg, 2011, pp. 60–68 (cit. on p. 17).

[45] Friedemann Zenke and Surya Ganguli. “SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks”. In: *Neural Computation* 30.6 (June 2018), pp. 1514–1541 (cit. on p. 18).

[46] Sumit Bam Shrestha and Garrick Orchard. “SLAYER: Spike Layer Error Reassignment in Time”. In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018 (cit. on pp. 18, 19, 34, 42, 43, 45).

[47] Nicholas T. Carnevale and Michael L. Hines. *The NEURON Book*. Cambridge, 2006 (cit. on p. 18).

[48] Marc-Oliver Gewaltig and Markus Diesmann. “NEST (NEural Simulation Tool)”. In: *Scholarpedia* 2.4 (2007), p. 1430 (cit. on p. 18).

[49] Marcel Stimberg, Romain Brette, and Dan FM Goodman. “Brian 2, an intuitive and efficient neural simulator”. In: *eLife* 8 (Aug. 2019). Ed. by Frances K Skinner, e47314 (cit. on p. 18).

[50] Andrew Davison et al. “PyNN: a common interface for neuronal network simulators”. In: *Frontiers in Neuroinformatics* 2 (2009) (cit. on p. 19).

[51] Hananel Hazan et al. “BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python”. In: *Frontiers in Neuroinformatics* 12 (2018), p. 89 (cit. on p. 19).

[52] Intel. *LAVA, a Software Framework for Neuromorphic Computing*. `https://github.com/lava-nc/lava`. 2022 (cit. on p. 19).

[53] Christian Pehle and Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks*. Version 0.0.7. Jan. 2021 (cit. on p. 20).

[54] Daniel Rasmussen. *NengoDL: Combining deep learning and neuromorphic modelling methods*. Mar. 2019 (cit. on p. 20).

[55] Trevor Bekolay et al. “Nengo: a Python tool for building large-scale functional brain models”. In: *Frontiers in Neuroinformatics* 7 (2014) (cit. on p. 20).

[56] Wei Fang, Yanqi Chen, et al. *SpikingJelly*. `https://github.com/fangwei123456/spikingjelly`. 2020 (cit. on pp. 20, 73, 114).

[57] Prophesee. *What is Event-Based Vision | Metavision by Prophesee*. Jan. 2021 (cit. on p. 21).

[58] Daniel Neil. “Deep Neural Networks and Hardware Systems for Event-driven Data”. en. Doctoral Thesis. ETH Zurich, 2017 (cit. on p. 22).

[59] K.A. Boahen. “A burst-mode word-serial address-event link-I: transmitter design”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 51.7 (July 2004), pp. 1269–1280 (cit. on p. 22).

[60] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. “A 128$\times$ 128 120 dB 15 µs Latency Asynchronous Temporal Contrast Vision Sensor”. In: *IEEE Journal of Solid-State Circuits* 43.2 (Feb. 2008), pp. 566–576 (cit. on p. 22).

[61] 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 (Jan. 2011), pp. 259–275 (cit. on p. 22).

[62] Thomas Finateu et al. “5.10 A 1280×720 Back-Illuminated Stacked Temporal Contrast Event-Based Vision Sensor with 4.86µm Pixels, 1.066GEPS Readout, Programmable Event-Rate Controller and Compressive Data-Formatting Pipeline”. In: *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. Feb. 2020, pp. 112–114 (cit. on p. 23).

[63] Yunjae Suh et al. “A 1280×960 Dynamic Vision Sensor with a 4.95-µm Pixel Pitch and Motion Artifact Minimization”. In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. Oct. 2020, pp. 1–5 (cit. on p. 23).

[64] Shoushun Chen and Menghan Guo. “Live Demonstration: CeleX-V: A 1M Pixel Multi-Mode Event-Based Sensor”. en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Long Beach, CA, USA, June 2019, pp. 1682–1683 (cit. on p. 23).

[65] Gemma Taverni et al. “Front and Back Illuminated Dynamic and Active Pixel Vision Sensors Comparison”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 65.5 (May 2018), pp. 677–681 (cit. on p. 23).

[66] Guillermo Gallego, Tobi Delbrück, et al. “Event-Based Vision: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.1 (Jan. 2022), pp. 154–180 (cit. on pp. 23, 25).

[67] Amos Sironi et al. “HATS: Histograms of Averaged Time Surfaces for Robust Event-Based Object Classification”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2018, pp. 1731–1740 (cit. on pp. 23, 49, 50, 73, 96, 114).

[68] Ana I. Maqueda et al. “Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2018, pp. 5419–5427 (cit. on pp. 23, 26).

[69] 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 (July 2017), pp. 1346–1359 (cit. on p. 24).

[70] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. “A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation”. In: *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (June 2018) (cit. on pp. 24, 25).

[71] Yusuke Sekikawa, Kosuke Hara, and Hideo Saito. “EventNet: Asynchronous Recursive Event Processing”. en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA, June 2019, pp. 3882–3891 (cit. on p. 24).

[72] Tobias Brosch, Stephan Tschechne, and Heiko Neumann. “On event-based optical flow detection”. In: *Frontiers in Neuroscience* 9 (2015) (cit. on p. 25).

[73] David Weikersdorfer and Jörg Conradt. “Event-based particle filtering for robot self-localization”. In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Dec. 2012, pp. 866–870 (cit. on p. 25).

[74] Bodo Rueckauer, Iulia-Alexandra Lungu, et al. “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification”. In: *Frontiers in Neuroscience* 11 (2017) (cit. on p. 25).

[75] Hongmin Li, Guoqi Li, and Luping Shi. “Classification of Spatiotemporal Events Based on Random Forest”. en. In: *Advances in Brain Inspired Cognitive Systems*. Ed. by Cheng-Lin Liu et al. Lecture Notes in Computer Science. Cham, 2016, pp. 138–148 (cit. on p. 26).

[76] Anthony Fleury, Michel Vacher, and Norbert Noury. “SVM-Based Multi-Modal Classification of Activities of Daily Living in Health Smart Homes: Sensors, Algorithms and First Experimental Results”. In: *IEEE Transactions on Information Technology in Biomedicine* 14.2 (Mar. 2010), pp. 274 –283 (cit. on p. 26).

[77] Alex Zhu et al. “EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras”. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018 (cit. on p. 26).

[78] Alex Zihao Zhu et al. “Unsupervised Event-Based Learning of Optical Flow, Depth, and Egomotion”. en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA, June 2019, pp. 989–997 (cit. on p. 26).

[79] Henri Rebecq et al. “Events-To-Video: Bringing Modern Computer Vision to Event Cameras”. en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA, June 2019, pp. 3852–3861 (cit. on p. 26).

[80] 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 (Feb. 2018), pp. 106–122 (cit. on p. 27).

[81] Alexander Neckar et al. “Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model”. In: *Proceedings of the IEEE* 107.1 (Jan. 2019), pp. 144–164 (cit. on p. 27).

[82] Steve Furber et al. “Overview of the SpiNNaker System Architecture”. In: *Computers, IEEE Transactions on* 62 (Dec. 2013), pp. 2454–2467 (cit. on p. 27).

[83] 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 (Oct. 2015), pp. 1537–1557 (cit. on p. 27).

[84] Mike Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* 38.1 (Jan. 2018), pp. 82–99 (cit. on p. 27).

[85] Peter Blouw et al. *Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware*. Mar. 2019 (cit. on p. 27).

[86] Garrick Orchard et al. “Efficient Neuromorphic Signal Processing with Loihi 2”. In: *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. Oct. 2021, pp. 254–259 (cit. on p. 27).

[87] Brainchip. *Akida 1000 Ref SoC*. `https://brainchip.com/akida-neural-processor-soc/`. 2020 (cit. on p. 28).

[88] Brainchip. *BrainChip Demonstrates Smart Automotive*. Oct. 2021 (cit. on p. 28).

[89] Nassim Abderrahmane et al. “SPLEAT: SPiking Low-power Event-based ArchiTecture for in-orbit processing of satellite imagery”. In: *IEEE WCCI Congress*. Padua, Italy, July 2022 (cit. on pp. 28, 45, 115).

[90] Prophesee. *Event Preprocessing Tutorial - event cube — Metavision Intelligence Docs 3.0.2* (cit. on p. 33).

[91] Loïc Cordone, Benoit Miramond, and Phillipe Thierion. “Object Detection with Spiking Neural Networks for Automotive Event Data”. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. July 2022 (cit. on pp. 33, 47, 70).

[92] Weihua He et al. “Comparing SNNs and RNNs on neuromorphic vision datasets: Similarities and differences”. en. In: *Neural Networks* 132 (Dec. 2020), pp. 108–120 (cit. on p. 34).

[93] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. “Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)”. In: *Frontiers in Neuroscience* 14 (2020) (cit. on pp. 34, 43, 45).

[94] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, et al. “Incorporating Learnable Membrane Time Constant To Enhance Learning of Spiking Neural Networks”. en. In: 2021, pp. 2661–2671 (cit. on pp. 34, 42, 43, 45, 47, 55).

[95] Yannan Xing, Gaetano Di Caterina, and John Soraghan. “A New Spiking Convolutional Recurrent Neural Network (SCRNN) With Applications to Event-Based Hand Gesture Recognition”. In: *Frontiers in Neuroscience* 14 (2020) (cit. on pp. 35, 43, 45).

[96] Hanle Zheng et al. “Going Deeper With Directly-Trained Larger Spiking Neural Networks”. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (May 2021), pp. 11062–11070 (cit. on p. 36).

[97] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, et al. “Deep Residual Learning in Spiking Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021, pp. 21056–21069 (cit. on pp. 36, 37, 48).

[98] Byunggook Na et al. “AutoSNN: Towards Energy-Efficient Spiking Neural Networks”. en. In: *Proceedings of the 39th International Conference on Machine Learning*. June 2022, pp. 16253–16269 (cit. on p. 36).

[99] Loïc Cordone, Benoit Miramond, and Sonia Ferrante. “Learning from Event Cameras with Sparse Spiking Convolutional Neural Networks”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. July 2021, pp. 1–8 (cit. on pp. 37, 47).

[100] Nico Messikommer et al. “Event-Based Asynchronous Sparse Convolutional Networks”. en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Lecture Notes in Computer Science. Cham, 2020, pp. 415–431 (cit. on pp. 38, 50, 74, 96, 98).

[101] Christopher Choy, JunYoung Gwak, and Silvio Savarese. “4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3075–3084 (cit. on pp. 38, 40).

[102] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. “3D Semantic Segmentation with Submanifold Sparse Convolutional Networks”. In: *CVPR* (2018) (cit. on p. 38).

[103] *Benchmark - Minkowski Engine* (cit. on p. 38).

[104] Arnon Amir et al. “A Low Power, Fully Event-Based Gesture Recognition System”. en. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, July 2017, pp. 7388–7397 (cit. on p. 41).

[105] Rong Xiao et al. “An Event-Driven Categorization Model for AER Image Sensors Using Multispike Encoding and Learning”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.9 (Sept. 2020), pp. 3649–3657 (cit. on p. 47).

[106] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations*. 2015 (cit. on pp. 48, 113).

[107] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Apr. 2017 (cit. on p. 48).

[108] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 2261–2269 (cit. on p. 49).

[109] Pierre de Tournemire et al. *A Large Scale Event-based Detection Dataset for Automotive*. Jan. 2020 (cit. on p. 49).

[110] Loïc Cordone. *Object Detection with Spiking Neural Networks*. `https://github.com/loiccordone/object-detection-with-spiking-neural-networks`. 2022 (cit. on p. 49).

[111] Alexander Kugele et al. “Hybrid SNN-ANN: Energy-Efficient Classification and Object Detection for Event-Based Vision”. In: *Pattern Recognition: 43rd DAGM German Conference, DAGM GCPR 2021, Bonn, Germany, September 28 – October 1, 2021, Proceedings*. Berlin, Heidelberg, Sept. 2021, pp. 297–312 (cit. on pp. 50, 68, 69, 96).

[112] Marco Cannici et al. “Asynchronous Convolutional Networks for Object Detection in Neuromorphic Cameras”. In: 2019, pp. 0–8 (cit. on pp. 50, 96).

[113] Yijin Li et al. “Graph-Based Asynchronous Event Processing for Rapid Object Recognition”. en. In: 2021, pp. 934–943 (cit. on pp. 50, 96).

[114] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587 (cit. on pp. 60, 61).

[115] Jasper Uijlings et al. “Selective Search for Object Recognition”. In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171 (cit. on p. 60).

[116] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’15. Montreal, Canada, 2015, pp. 91–99 (cit. on p. 61).

[117] Jifeng Dai et al. “R-FCN: Object Detection via Region-Based Fully Convolutional Networks”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. Barcelona, Spain, 2016, pp. 379–387 (cit. on p. 61).

[118] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788 (cit. on p. 62).

[119] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. en. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Lecture Notes in Computer Science. Cham, 2016, pp. 21–37 (cit. on pp. 63, 70).

[120] Tsung-Yi Lin, Priya Goyal, et al. “Focal Loss for Dense Object Detection”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017, pp. 2999–3007 (cit. on pp. 63–65, 70).

[121] Tsung-Yi Lin, Piotr Dollár, et al. “Feature Pyramid Networks for Object Detection”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 936–944 (cit. on p. 64).

[122] Syed Sahil Abbas Zaidi et al. “A survey of modern deep learning based object detection models”. en. In: *Digital Signal Processing* 126 (June 2022), p. 103514 (cit. on p. 65).

[123] Seijoon Kim et al. “Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection”. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.07 (Apr. 2020), pp. 11270–11277 (cit. on pp. 66, 67).

[124] Mark Everingham et al. *The PASCAL Visual Object Classes (VOC) challenge*. 2010 (cit. on p. 66).

[125] Tsung-Yi Lin, Michael Maire, et al. “Microsoft COCO: Common Objects in Context”. en. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Lecture Notes in Computer Science. Cham, 2014, pp. 740–755 (cit. on p. 66).

[126] Sami Barchid, José Mennesson, and Chaabane Djéraba. “Deep Spiking Convolutional Neural Network for Single Object Localization Based On Deep Continuous Local Learning”. In: *2021 International Conference on Content-Based Multimedia Indexing (CBMI)*. June 2021, pp. 1–5 (cit. on pp. 67, 68).

[127] Etienne Perot et al. “Learning to Detect Objects with a 1 Megapixel Event Camera”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 16639–16652 (cit. on pp. 74–76, 93, 98).

[128] Marco Cannici et al. “A Differentiable Recurrent Surface for Asynchronous Event-Based Data”. en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Vol. 12365. Cham, 2020, pp. 136–152 (cit. on pp. 74, 98).

[129] Alwyn Mathew. *Gradient flow check in Pytorch* (cit. on p. 81).

[130] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017 (cit. on p. 86).

[131] Asher Trockman and J. Zico Kolter. *Patches Are All You Need?* Jan. 2022 (cit. on p. 87).

[132] Zhuang Liu et al. “A ConvNet for the 2020s”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022) (cit. on p. 87).

[133] Edgar Lemaire, Loïc Cordone, et al. “An Analytical Estimation of Spiking Neural Networks Energy Efficiency”. In: *29th International Conference on Neural Information Processing (ICONIP 2022)*. Sept. 2022 (cit. on p. 104).

[134] Sami Barchid, José Mennesson, Jason Eshraghian, et al. *Spiking Neural Networks for Frame-based and Event-based Single Object Localization*. June 13, 2022 (cit. on pp. 104, 105).

[135] Lyes Khacef, Nassim Abderrahmane, and Benoit Miramond. “Confronting machine-learning with neuroscience for neuromorphic architectures design”. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. July 2018, pp. 1–8 (cit. on p. 105).

[136] Edgar Lemaire, Benoit Miramond, et al. “Synaptic Activity and Hardware Footprint of Spiking Neural Networks in Digital Neuromorphic Systems”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 37.4 (Aug. 2022), p. 26 (cit. on p. 105).

[137] Bodo Rueckauer, Connor Bybee, et al. “NxTF: An API and Compiler for Deep Spiking Neural Networks on Intel Loihi”. In: *ACM Journal on Emerging Technologies in Computing Systems* 18.3 (Jan. 29, 2022), 48:1–48:22 (cit. on p. 105).

[138] Souvik Kundu et al. “Spike-Thrift: Towards Energy-Efficient Deep Spiking Neural Networks by Limiting Spiking Activity via Attention-Guided Compression”. In: *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Waikoloa, HI, USA, Jan. 2021, pp. 3952–3961 (cit. on p. 105).

[139] Lei Deng et al. “Rethinking the performance comparison between SNNS and ANNS”. In: *Neural Networks* 121 (Jan. 1, 2020), pp. 294–307 (cit. on p. 105).

[140] Simon Davidson and Steve B. Furber. “Comparison of Artificial and Spiking Neural Networks on Digital Hardware”. In: *Frontiers in Neuroscience* 15 (2021) (cit. on p. 105).

[141] Norman P. Jouppi et al. “Ten Lessons From Three Generations Shaped Google’s TPUv4i : Industrial Product”. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. June 2021, pp. 1–14 (cit. on p. 112).