Computer Science Department, Faculty of Exact Sciences, Mascara - Algeria
Mathematics and Digital Sciences doctoral school (MADIS), Lille - France

Interactive Analysis of Spiking Neural Networks Simulation Traces

Doctoral Thesis
Computer Science

Hammouda Elbez

20 June 2022

Supervisors

Director
Pr. Pierre Boulet
Université de Lille, France

Co-director
Pr. Kamel Benhaoua
Université de Mascara, Algérie

Guest

Dr. Philippe Devienne
CNRS, France

Jury

President
Pr. Sidi Mohammed Benslimane
Ecole Supèrieure en Informatique de Sidi Bel Abbès, Algérie

Referees
Dr. Gilles Sassatelli
CNRS, France

Pr. Sidi Mohammed Benslimane
Ecole Supèrieure en Informatique de Sidi Bel Abbès, Algérie

Examiners
Dr. Ihsen Alouani
Université Polytechnique Hauts-de-France, France

Pr. Fatima Debbat
Université de Mascara, Algérie

Université de Lille

Université de Mascara

Ecole Doctorale MADIS, Lille - France
Département d'Informatique, Faculté des sciences exactes, Mascara - Algérie

Visualisation interactive de traces de simulation de réseaux neurones matériels à impulsions

Thèse de doctorat
Informatique

Hammouda Elbez

20 Juin 2022

Superviseurs

Directeur
Pr. Pierre Boulet
Université de Lille, France

Co-directeur
Pr. Kamel Benhaoua
Université de Mascara, Algérie

Invité

Dr. Philippe Devienne
CNRS, France

Jurés

Président
Pr. Sidi Mohammed Benslimane
Ecole Supèrieure en Informatique de Sidi Bel Abbès, Algérie

Rapporteurs
Dr. Gilles Sassatelli
CNRS, France

Pr. Sidi Mohammed Benslimane
Ecole Supèrieure en Informatique de Sidi Bel Abbès, Algérie

Examinateurs
Dr. Ihsen Alouani
Université Polytechnique Hauts-de-France, France

Pr. Fatima Debbat
Université de Mascara, Algérie

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

IRCICA

Acknowledgements

This manuscript is a result of a five-year Ph.D. thesis journey between two universities (the University of Mascara in Algeria & University of Lille in France). First, I would like to thank the Algerian and French ministry of higher education and scientific research for the Algerian-French cooperation Profas B+ scholarship that allowed me to conduct this thesis between both universities. Second, I have to thank CRIStAL and the IRCICA institute. For three years, I worked in one of their offices after arriving in France, using their computation resources and facilities, which helped me advance my scientific research.

Next, I am extremely grateful to my supervisors, Prof. Mohamed Kamel Benhaoua and Prof. Pierre Boulet, for their invaluable advice, continuous support, and patience during my Ph.D. study. Their immense knowledge and great experience have encouraged me throughout my academic research and daily life. Moreover, I would also like to express gratitude to Dr. Philippe Devienne for his kind help and support that have made my study and life in France a fantastic time from the first day. I want to thank the thesis committee members: Pr. Sidi Mohammed Benslimane, Dr. Gilles Sassatelli, Dr. Ihsen Alouani, and Pr. Fatima Debbat, for taking the time to read and evaluate this manuscript and all the advice and refreshing ideas that I appreciate and will help improve this manuscript.

My appreciation goes out to my parents, Naima Benyammi and Omar Elbez, for their encouragement and support throughout my studies and for always believing in me. Furthermore, a special thank you to my wife and life partner, Amal Dabouz, for her support during the thesis and during difficult times. Finally, thank you to all colleagues and friends from the Emeraude team and IRCICA for their kindness and friendliness.

To all whom I have not mentioned, sincerely, Thank you

– Hammouda Elbez

Abstract

Neuromorphic architectures are promising approaches to significantly reduce the energy consumption for tomorrow's computers and the post-Moore era. The brain function is the inspiration behind this architecture, consisting of spiking artificial neural networks. Due to low energy consumption, deploying such architectures is useful in many applications, especially those with energy-limited constraints. Furthermore, using this architecture, we can process a large quantity of data and provide the computation power needed for machine learning tasks.

Neuromorphic architectures consist of spiking artificial neural networks inspired by the brain functionalities with many open questions. This situation impacts the implementation of artificial spiking neural networks and their performance compared to conventional neural networks. Moreover, in SNN, many questions are still debatable, like how the learning is happening and what learning rule is the most suitable, memory location in such networks and how it works, and how the network encodes the information using spikes. Such neuroscience-related questions prevent spiking neural networks from performing like the conventional ones. Therefore, to better understand the different phenomena in SNN, we need to analyze the internal network activity during the simulation. The network activity contains the spikes, neurons, and synapses states activity. When simulating a large network that takes time and resources to finish, we generate a large simulation trace that is challenging to analyze due to its size and spatio-temporal aspect, which we can study at several scales.

This manuscript aims to study the visual analysis of spiking neural networks by visualizing the collected trace from a simulation. The primary objective is to better understand the different network phenomena and improve the network using visual analysis. The first contribution is the study of the visualization techniques in SNN simulators. This study from the technical and visualization aspect of the simulators shed light on the diversity of the used technologies. Furthermore, this study also shows the similarity of the visualization techniques provided by the simulators. At the end of this study, we concluded that we need more dedicated tools to analyze than what simulators provide for visual analysis.

Next, we developed VS2N (Visualization tool for Spiking Neural Networks). A web-based tool for post-mortem interactive dynamic visualization and analysis of spiking neural networks. The novelty of VS2N compared to the existing visual analysis tools can be summarized in four points: modular nature, simulator-independent, scalability, and dynamic analytics. In addition, VS2N provides the possibility to walk in time with the evolution of the network during activity, which is not possible using the existing tools. This feature is significant when the network evolution is over hours of activity, which is the case in spiking neural networks.

Finally, we proposed a novel approach to compress a spiking neural network based on the visual analysis conducted on SNN. This dynamic compression approach concerns the synapses in the network by providing two formulas to calculate the dynamic threshold, which changes based on the compression status, instead of having a static threshold which is the case in the existing works. This approach can maintain or improve the network accuracy compared to the non-compressed network while compressing up to 80%.

الملخص

تعتبر البنى العصبية من الطرق الواعدة للحد بشكل كبير من استهلاك الطاقة لأجهزة الكمبيوتر المستقبلية وعصر مابعد Moore. وظيفة الدماغ هي مصدر الإلهام وراء هذه البنى، التي تتكون من شبكات عصبية اصطناعية. نظرا لاستهلاكها المنخفض للطاقة، فإن استعمال مثل هذه البنى مفيد في العديد من التطبيقات، خاصة تلك التي تستلزم استهلاكا محدودا للطاقة. علاوة على ذلك، بفضل هذه البنية، يمكننا معالجة كمية كبيرة من البيانات وتوفير القوة الحاسوبية اللازمة لمهام التعلم الآلي.

تتكون البنى العصبية من شبكات عصبية مسننة (Spiking Neural Networks) مستوحاة من وظائف الدماغ ذو العديد من الأسئلة بدون جواب. يؤثر هذا الموقف على فعالية هذه الشبكات وأدائها مقارنة بالشبكات العصبية التقليدية. علاوة على ذلك، فيما يخص SNN، لا تزال العديد من الأسئلة قابلة للنقاش، مثل كيفية التعلم وما هي قاعدة التعلم الأنسب، وموقع الذاكرة في هذه الشبكات وكيف تعمل، وكيف تقوم الشبكة بتمثيل المعلومات باستخدام النبضات. تحول مثل هذه الأسئلة المتعلقة بعلم الأعصاب بين الشبكات العصبية المسننة و بين ادائها بشكل افضل مثل الشبكات التقليدية. لذلك لفهم الظواهر المختلفة في SNN بشكل أفضل، نحتاج إلى تحليل النشاط الداخلي للشبكة أثناء المحاكاة. يحتوي نشاط الشبكة على نشاط خلايا العصبية ونقاط الاشتباك العصبى. عند محاكاة شبكة كبيرة ستستغرق وقتا اكبر و تتطلب مواردا معتبرة لذلك، بعدها نتحصل على ملفات تتبع النشاط الداخلي ذات حجم كبير يصعب تحليلها نظرا لحجمها وجانبها المكاني والزماني، والذي يمكننا دراسته على عدة مستويات.

تهدف هذه الاطروحة إلى دراسة التحليل المرئي للشبكات العصبية المسننة SNN من خلال ملفات تتبع النشاط الداخلي التي تم جمعها من المحاكاة. الهدف الأساسي هو فهم الظواهر الداخلية للشبكة بشكل أفضل وتحسين الشبكة باستخدام التحليل المرئي. المساهمة الأولى هي دراسة تقنيات التصور في برامج محاكات SNN. تلقي هذه الدراسة من الجانب الفني وتقنيات التصور

لبرامج المحاكاة الضوء على تنوع التقنيات المستخدمة. علاوة على ذلك، تشابه تقنيات التصور المقدمة. في نهاية هذه الدراسة، خلصنا إلى أننا بحاجة إلى أدوات متخصصة في التحليل احسن من الاعتماد على ما توفره برامج المحاكاة لتحليل البيانات. بعد ذلك، قمنا بالتطوير VS2N (Visualization tool for Spiking Neural Networks). أداة قائمة على الويب للتصور الديناميكي التفاعلي لتحليل الشبكات العصبية المسننة. يمكن تلخيص حداثة VS2N مقارنة بأدوات التحليل المرئي الموجودة في أربع نقاط: الطبيعة المعيارية، والمحاكاة المستقلة، وقابلية التوسع، والتحليلات الديناميكية. بالإضافة إلى ذلك، يوفر VS2N إمكانية التنقل عبر الزمن مع تطور الشبكة أثناء النشاط، وهو أمر غير ممكن باستخدام الأدوات الحالية. هذه الميزة مهمة عندما يكون نشاط الشبكة على مدى عدة ساعات، وهذا هو الحال في الشبكات العصبية المسننة.

أخيرا، اقترحنا طريقة جديدة لضغط الشبكات العصبية المسننة بناءً على التحليل المرئي الذي تم إجراؤه على الشبكة. تعتمد هذه الطريقة على التقليص الديناميكي لنقاط الاشتباك العصبى في الشبكة من خلال اقتراح معادلتين لحساب العتبة الديناميكية للتقليص، والتي تتغير بناءً على حالة الشبكة، بدلاً من وجود عتبة ثابتة كما هو الحال في الأعمال السابقة. يمكن لهذا الأسلوب الحفاظ على دقة الشبكة أو تحسينها مقارنة بالشبكة غير المضغوطة بنسبة ضغط تصل إلى ٨٠٪ .

Résumé

Les architectures neuromorphiques sont des approches prometteuses pour réduire significativement la consommation énergétique des ordinateurs de demain et de l'ère post-Moore. La fonction cérébrale est l'inspiration derrière cette architecture, consistant en des réseaux de neurones artificiels à impulsion. Du fait de leur faible consommation énergétique, le déploiement de telles architectures est utile dans de nombreuses applications, notamment celles ayant des contraintes énergétiques limitées. De plus, grâce à cette architecture, nous pouvons traiter une grande quantité de données et fournir la puissance de calcul nécessaire aux tâches d'apprentissage automatique.

Les architectures neuromorphiques consistent en des réseaux de neurones impulsionnels (SNN) inspirés des fonctionnalités cérébrales avec de nombreuses questions ouvertes. Cette situation impacte la mise en place de réseaux de neurones à impulsions et leurs performances par rapport aux réseaux de neurones classiques. De plus, dans les SNN, de nombreuses questions sont encore ouvertes, par exemple: comment se passe l'apprentissage et quelle règle d'apprentissage est la plus appropriée, l'emplacement de la mémoire dans ces réseaux et comment cela fonctionne, et comment le réseau représente les informations à l'aide d'impulsions. De telles questions liées aux neurosciences empêchent les SNN de fonctionner comme les réseaux conventionnels. Par conséquent, pour mieux comprendre les différents phénomènes dans les SNN, nous devons analyser l'activité interne du réseau lors de la simulation. L'activité du réseau consiste en celle des impulsions, des neurones et des synapses. Lors de la simulation, nous générons une trace de simulation difficile à analyser en raison de sa taille et de son aspect spatio-temporel, que nous pouvons étudier à plusieurs échelles.

Ce manuscrit vise à étudier l'analyse visuelle des réseaux de neurones impulsionnels en visualisant la trace collectée à partir d'une simulation. L'objectif principal est de mieux comprendre les différents phénomènes du réseau et d'améliorer le réseau à l'aide d'analyses visuelles. La première contribution est l'étude des techniques de visualisation dans les simulateurs SNN. Cette étude du point de vue technique et visualisation des simulateurs a mis en lumière la diversité des technologies utilisées.

Par ailleurs, cette étude montre également la similarité des techniques de visualisation fournies par les simulateurs. À la fin de cette étude, nous avons conclu qu'il nous faut des outils dédiés pour analyser la trace fournie par les simulateurs.

Ensuite, nous avons développé VS2N. Un outil basé sur technologie Web pour la visualisation et l'analyse dynamiques interactives post-mortem des réseaux de neurones à impulsions. La nouveauté de VS2N par rapport aux outils d'analyse visuelle existants peut être résumée en quatre points : sa nature modulaire, son indépendance par rapport au simulateur, son scalabilité et ses capacités en analyse dynamique. De plus, VS2N offre la possibilité de fonctionner sur le réseau en cours de simulation, ce qui n'est pas possible avec les outils existants. Cette caractéristique est importante à noter surtout pour les longues simulations, ce qui est souvent le cas pour les réseaux de neurones à impulsions.

Enfin, nous avons proposé une nouvelle approche pour compresser un réseau de neurones à impulsions basée sur l'analyse visuelle menée sur les SNN. Cette approche de compression dynamique concerne les synapses du réseau en proposant deux formules pour calculer le seuil dynamique, qui évolue en fonction de l'état de compression précédente, au lieu d'avoir un seuil statique comme c'est le cas dans les travaux existants. Cette approche peut maintenir ou améliorer la performance du réseau par rapport au réseau non compressé tout en compressant jusqu'à 80%.

Contents

Chapter 1

Introduction and motivations

The machine learning field emerged as one of the most attractive fields nowadays, which provides algorithms to solve different tasks, such as image classification, segmentation, and prediction. What makes the machine learning field this interesting is the ability to associate it with other areas (agronomy, medicine, transportation, etc.) and apply those algorithms to provide solutions that expand what we can achieve using classic approaches. Neural networks are machine learning components with convolutional neural networks and deep learning technologies. This technology allows the processing of complex tasks, and we usually consider it the first resort in different fields when using machine learning.

However, with the increased amount of data collected everywhere, the ability to process and handle this massive quantity of data becomes a challenge over time. Moreover, with the advancement in machine learning and neural networks, we have more powerful models which require more computation power and energy to run correctly. This need represents a challenge due to the Von Neumann architecture limitation, and the end of Moore's Law. As a result, the hardware progress is slowing down and cannot keep up with the way big data and machine learning algorithms are growing. To overcome the limitation of the existing architecture and fulfill the need for more computation power in the machine learning field, we need an architecture that can go beyond Von Neumann's computing. Research highlights two candidates based on the existing works: Quantum computing and Neuromorphic architectures; the latter interests us in this work.

Neuromorphic architectures are promising approaches to significantly reducing the energy consumption for tomorrow's computers. The brain function is the inspiration behind this architecture and consists of Spiking Neural Networks (SNN). Due to their low energy consumption and the use of spikes for communication, deploying such architectures can be helpful in many applications, especially those with energy-limited constraints. With the low energy consumption and the nature of

neuromorphic architectures, we can process huge quantities of data and provide the computation power needed for machine learning tasks. Due to the brain's complex activity, the purpose of neuromorphic computing is not to perfectly imitate the brain's functions. Instead, we extract the knowledge we have on the internal activity of the brain and try to represent it on a computing system.

Neuromorphic architectures consist of spiking neural networks. Those networks are inspired by the brain functionalities, the most powerful machine with many open questions. This situation reflects the implementation of spiking neural networks and their performance compared to conventional neural networks. Spiking Neural Networks can be suitable for many domains (like multimedia-related tasks). In contrast to conventional neural networks, SNNs consider time during activity and process natural signals while being robust to noise, which can be helpful such in a sound recognition task [Wu et al., 2018], texture retrieval of images [Yang et al., 2017], or image classification [Falez et al., 2019]. Moreover, in SNN, many questions are still debatable, like how the learning is happening and what learning rule is the most suitable, memory location in such networks and how it works, and how the network encodes the information in the spikes. Such neuroscience-related questions prevent spiking neural networks from performing like the conventional ones. Therefore, much of today's research focuses on providing answers and improving this technology, either in biology, electronics, or computer science. One of the main decisions we need to make in the process of making neuromorphic hardware is implementing neurons on hardware. There are two leading technologies to choose from, which are: CMOS [Sourikopoulos et al., 2017], which is a mature technology used in most of the existing hardware, and memristors [Strukov et al., 2008], which is relatively new compared to CMOS and has the potential to reduce the hardware components required for neuron implementation. In recent years, we saw many hardware implementations starting from 2010. However, most of them are still for research purposes only [Schemmel et al., 2010, Benjamin et al., 2014, Furber et al., 2014, Merolla et al., 2014, Davies et al., 2018].

1.1 Motivations

A neural network is considered a black box due to its structure and the internal activity. Therefore, it is hard to identify the learned function from a neural network or to guarantee that a network learns a defined function. Moreover, since the usual

way (the unique way in most cases) to evaluate a neural network is by measuring the accuracy, the network is exposed to potential issues such as overfitting, discrimination issues, or any dysfunction due to biased data. In addition, due to the network complexity, it's time-consuming to debug a network in case of a bad performance and pinpoint the issue, which can be the input data, the network structure, the used parameters, or a single layer in the network.

Moreover, with the increase in complexity of neural networks comes an increase in computation power and energy demand to satisfy the requirements of the recent neural networks and deep learning. The Von Neumann architecture often penalizes such hardware-dependent networks due to their limitation. They usually depend on GPU or super calculators to provide the computation power, which raises the issue of energy consumption and carbon footprint emissions [Dhar, 2020, Schwartz et al., 2020], especially with the huge increase in network complexity in recent years. In the NLP (Natural Language Processing) domain, GPT-2 was presented in 2019 and has 1.5 billion parameters, followed by GPT-3 in 2020, which has 175 billion parameters (100x bigger than GPT-2), and still suffer from the same biases while require much more power. However, the bio-inspired technology (SNN) represents a promising replacement to reduce energy consumption and provide computation power despite the lack of a more profound understanding of those networks that affect its performance compared to the conventional neural network.

Spiking neural network provides ultra-low energy consumption via a spike-based communication inspired by biology. SNN is a promising way to provide computation power with low energy consumption for the next generation of neural networks. However, due to the bioinspired nature of SNN, many questions remain with no precise answer, the questions which may concern the proper learning mechanism, neurons type, or network architecture. In addition, all the neuroscience-related questions involving the brain impact the SNN performance. One of the reasons why SNN are underperforming is how the learning is done in the network, usually by using local learning rules such as STDP [Markram et al., 1997] instead of gradient-based global learning rules, since we cannot apply it on spikes. Some works proposed the use of the surrogate gradient descent [Neftci et al., 2019], which allows the use of global learning rule to train SNN using yet a considerable amount of energy. Others tried to exploit the SNN benefits by converting a trained ANN to SNN [Chen et al., 2018, Rueckauer et al., 2017, Xing et al., 2019] to keep the same performance at a low-energy consumption. However, bioinspired effective training methods for SNN are still lacking, despite the existing efforts involving local rules.

In spiking neural networks, we don't have a way or a technique to analyze the network activity, making it harder to improve and understand. This limitation also impacts the interpretation of results, this technology's usability, and the neuromorphic architecture. The motivation for this work is to present a solution for this limitation by adopting a different way of looking into neural networks using visual analysis, to improve this type of networks. With the help of human perception and prior knowledge about neural networks, we can explore and analyze the data we collect from simulations to understand the different phenomena happening inside the network. Using this approach, we can analyze the network differently, on-the-fly, and without waiting until the end of the simulation, which may take many hours (even days) to finish.

Visual analysis of spiking neural networks implies going through several interactive visualizations. It requires studying the existing visualization techniques for a similar type of data and a certain degree of interactivity. We can generate massive execution traces during the simulation of large neural networks (a few hundred thousand neurons, a few tens of millions of synapses) that last for many days. The traces contain all the network activity during the simulation. Analyzing such data is a challenge due to its size and the Spatio-temporal aspect that we can study at several scales.

1.2 Outline

In this manuscript, we present in chapter 2 a background on the neural networks, such as biological neuron presentation, neural network generations, neuron models, and learning in neural networks. Besides that, a brief introduction on visualization in general, human perception, and visualization in neural networks are included. Chapter 3 discusses the visualization techniques used in different spiking neural network simulators, comparing the visualizations they offer using multiple criteria and a technical review. Then, chapter 4 presents VS2N, a web-based visual analysis tool for spiking neural networks, the different modules, and use-cases where VS2N was used to understand or improve spiking neural networks. Next, in chapter 5 we present a novel progressive pruning approach and dynamic weights reinforcement, applied on single layer spiking neural networks. This approach allows the network to remove more than half of the unnecessary synapses while learning without losing the network accuracy, which is not the case in most pruning-related works. This approach is a

result of an observation using VS2N. Finally, chapter 6 concludes this manuscript by presenting a summary of the work presented in the different chapters and the perspectives on visualization in SNN and spiking neural networks in general.

Chapter 2

Background

To work on visual analysis of spiking neural networks, we need prior knowledge in a couple of fields, such as neuroscience: to understand how our brain works. Computer science: to implement and simulate neural networks with different components. Electronics: to better understand the process of making neuromorphic architectures and their hardware-related challenges. It is also important to learn about visualization, human perception, and how to extract information and insights from data. This chapter introduces neural networks and the different generations, neuron models, and learning rules. The second part of this chapter covers the visualization part, such as human perception, psychological principles of visual display, and visualization used in neural networks.

2.1 Biological neuron

A neuron is a cell in the nervous system that represents the elementary component of a neural network. It is capable of communicating and processing large amounts of information. There are almost 100 billion interconnected neurons in the human nervous system. However, we can distinguish different neurons according to their shape, behavior, and position in the brain.

A neuron generally does a simple operation that calculates the output based on the information received as input. Having many of those simple units working together and creating a network makes it possible to perform different tasks. It is impressive what a couple of neurons can achieve. A C. elegans is a tiny roundworm present on the soil. It is one of the simplest and easiest organisms to handle in the laboratory. With 1 mm in length and 0.2 mm in diameter, C. elegans has 959 cells (in adults). From those cells, we can find almost a third (302) of them are

neurons representing the brain (see Figure 2.1). By using those neurons, the tiny animal can move, feed, and reproduce while consuming ultra-low energy [White et al., 1986] [Frézal and Félix, 2015].

Figure 2.1: C. elegans nervous system

2.1.1 Neuron components

In Figure 2.2, we can break down a neuron into three principal parts, which are:

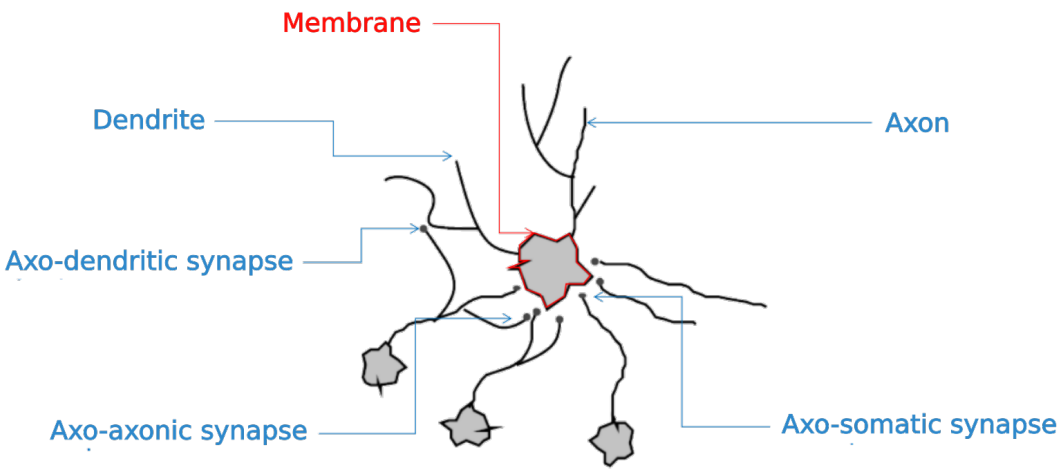

Figure 2.2: Structure of neuron

- The cell body (membrane) contains the core of the neuron and the other molecules essential to the cell's life, which has a few micron diameter.

- Dendrite is a thin tubular extension located around the neuron. It has a length of a few tens of microns. Dendrite is the element that captures and transmits the information from outside into the neuron cell.
- Axon is the information output medium from a neuron. It is longer than the dendrite and connects with the dendrite of other neurons. The size of an axon can vary from a few millimeters to several meters.
- Synapse represents the region of interaction between two neurons (see Figure 2.3) for the transmission of information (between the axon of one neuron and the dendrite of another), and there are several types of a synapse, such as axo-dendritic, axo-somatic, and axo-axonic. The density of a synapse directly affects the influence of a transmitter neuron on receptor neurons.

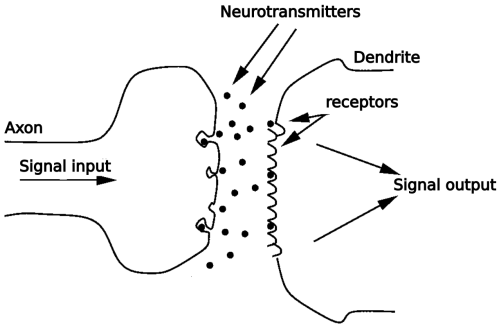

Figure 2.3: A synapse

2.1.2 Neuron functionality

A neuron is considered a polarized entity (information is transmitted in only one direction: from dendrites to axons). Therefore, when data arrives at a neuron, a summation will occur at the cell body level of all this information. Then, according to an activation function, the neuron will emit a signal along the axon, representing the result of the operation at the cell body level. When this signal arrives at the synaptic endings (pre-synaptic), synaptic vesicles will merge with the cell membrane, releasing neurotransmitters. These neurotransmitters Figure 2.3 represent chemical mediators that allow the information to go from one neuron to another because of the impossibility of having an electrical signal which passes through the synapse (in the case of a chemical synapse). In addition, there are receptors for neurotransmitters

on the dendritic membrane at the post-synaptic level. Therefore, the excitability of this neuron will change depending on the type of these neurotransmitters and the receptors. Thus propagate the information or not.

Synaptic connections are capable of adjusting their functioning according to their repeated activation or not, which will facilitate or not the passage of information, and this plasticity is considered the origin of learning mechanisms.

2.2 Artificial neural network

An artificial neural network is a group of artificial neurons working together, inspired by the human brain, which consists of about 100 billion neurons linked together by about one million billion synapses. This type of artificial network offers the possibility of modeling the information processing and learning mechanism in the human brain. Furthermore, this type of network is considered attractive, interesting, robust, and fault-tolerant solutions that provide true parallelism.

2.2.1 Artificial neuron

An artificial (formal) neuron is the mathematical translation of a biological neuron by decomposing each part to a mathematical equivalent, which performs the same task. Like a biological neuron, an artificial neuron transmits information in one direction from multiple entries to only one exit. An artificial neuron is composed of several parts, which are:

- Synaptic weights: this is the equivalent of synapses in biological neurons. Each synapse has its value, which will be used afterward and represents the strength of the connection between two neurons.

- Transfer function: also called activation function, which is the equivalent of the cell body, which has the role of simulating its operation and making the decision according to the result of this function to transmit the signal received or not.

- Output element: which represents the equivalent of axons in the biological neuron, it's used to transmit signals to the recipient neuron(s).

The artificial neuron receives and adds signals before transmitting them to the transfer function. Then, depending on its value and the result of the transfer function (for example: greater than a defined threshold), the neuron decides to produce an activation or not.

2.2.2 Neural network generations

We can classify the evolution of neural networks that started from the beginning of the 20th century into three generations:

The first generation

The first generation of this type of neural network saw the proposal of simple linear models of neural networks.

Formal model of McCulloch & Pitts

It's the first model McCulloch & Pitts proposed in 1943 [McCulloch and Pitts, 1943] to try and represent the neuron mathematically. This model (Figure 2.4) contains multiple entries $x_{1..n}$ with weights $w_{1..n}$, after summation of all the entries multiplied by their weights, we compare this value to a fixed threshold. If this value is more significant than the threshold, the neuron will send a signal; otherwise, nothing happens.

Rosenblatt Perceptron

It's a model of a neural network with a single layer and a learning rule. This model, presented by Rosenblatt in 1957 [Rosenblatt, 1958], is considered the first artificial system capable of learning by experience. The learning consists of calculating the

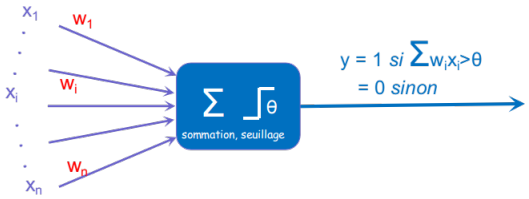

Figure 2.4: Formal model of McCulloch & Pitts

value of a using the same rule presented in Hebb's law (equation 2.6) and deduces the value of x. To calculate the new value of the weight $w_i(t+1)$, we use the formula 2.1 which consists of the current weight value of the neuron $w_i(t)$, the expected value x_d, the error estimation $x_d - x$,the input value e_i, and a positive constant μ.

$$w_i(t+1) = w_i(t) + \mu((x_d - x)e_i) \tag{2.1}$$

Adaline (Adaptive Linear Neuron) of Widrow

Widrow created this single-layer model of neural network in 1960 [Widrow and Hoff, 1960], and it is similar to perceptron. However, the learning rule is different. The learning in the perceptron model is based on examples and expected results, and Adaline is based on the error rate of each iteration. The learning steps are the same as the perceptron model. The only difference is the formula for calculating the new weight w_i (equation 2.2), where μ represents a positive constant, x_d the expected value, e_i the input value, and x which represents the actual output. This type of neuron converges to the least-squares error of $(x_d - x)^2$.

$$w_i = w_i + \mu(x_d - x)e_i \tag{2.2}$$

Kohonen Self-Organizing Maps (SOM)

Introduced by Kohonen in the 1980s [Kohonen, 1982], this map is deployed to represent a set of data. Each neuron represents a particular data group based on the common points that bring them together. This card uses unsupervised learning.

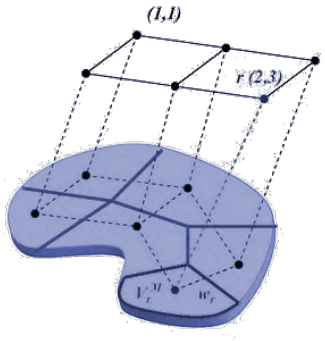

Figure 2.5: Kohonen Self-Organizing Maps

Hopfield Network

Proposed by Hopfield in 1982 [Hopfield, 1982], this network model (Figure 2.6) consists of N neurons with binary states all interconnected and the total input of a neuron i equal to the sum of all connections multiplied by its weight.

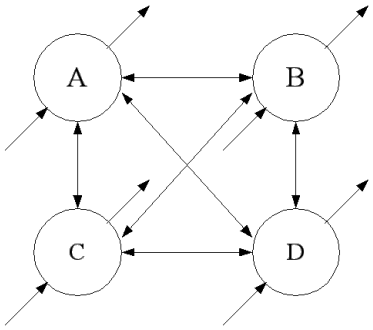

Figure 2.6: Hopfield Network

Multi-layer Perceptron

David Rumelhart presented it in 1986 [Rumelhart et al., 1986]. The notion of having several layers of neurons makes it possible to make non-linear associations and overcome the limitation of a single layer and linear problems. This model is based

on the retro propagation of the error gradient, and it is similar to the perceptron. The only difference is the way we calculate the synaptic weights.

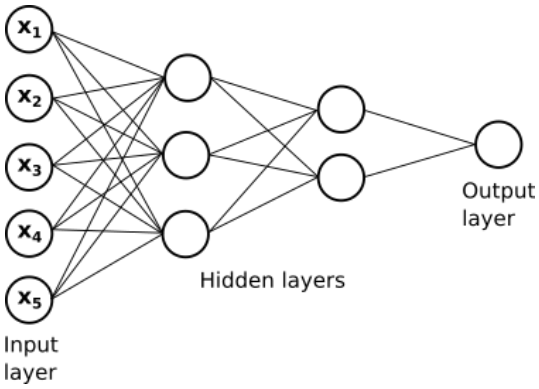

Figure 2.7: Multi-layer Perceptron

The second generation

A convolutional neural network (CNN) represents the second generation of neural networks. This type of neural network was introduced in the late 80s [LeCun et al., 1989] and is known for its power and efficiency, especially in recommendation systems and multimedia fields [LeCun et al., 1998, Avilov et al., 2020, van den Oord et al., 2013]. This type of network represents a set of successive processing over several layers. The group of outputs from a processing layer makes it possible to reconstitute an intermediate image which serves as input to the next layer.

The convolution is the operation used to calculate if a characteristic is present in a whole image, using defined filters.

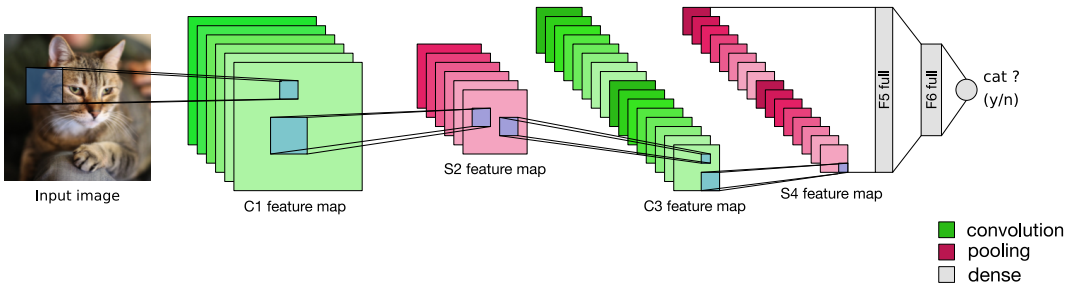

Figure 2.8: Convolution

CNN components

The network is composed of layers. Each layer has a specific function. The advantage of having the same type of data at input and output offers the possibility to stack several layers in the network. Thus, we can choose a suitable model according to its efficiency and needs. The different layers are:

Convolutional layer

The first step is looking for the existing features in the whole image by using the defined filters. It is a principal component of a convolutional neural network (CNN). The operation behind it is called a convolution, from which convolutional neural networks get their names. To calculate the correspondence between a feature and a subpart of the image, we multiply each feature pixel by the image pixel. Then, we sum the results and divide them by the total number of feature pixels. An example is shown in Figure 2.9.

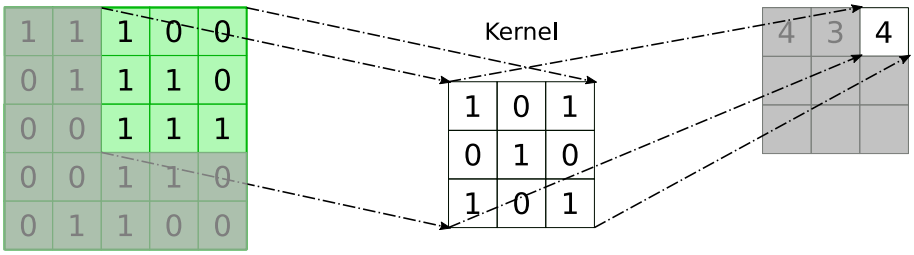

Figure 2.9: Convolution

Pooling layer

This layer is used to reduce the dimension of the received data while preserving important information. We can achieve this by dragging a small window step by step on all the image parts and choosing the maximum value (in the case of Max pooling, but there are other types of pooling). This technique can reduce the needed operations and the complexity for the following layers.

Figure 2.10: Max Pooling

Rectified Linear Units (ReLUs)

A unit performs a simple operation on the input data by turning the negative values from the input to zeros and keeping the positive values unchanged, which helps the network produce better results.

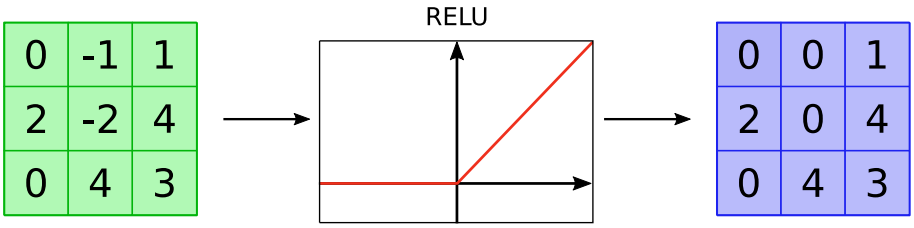

Figure 2.11: Rectified Linear Unit

Fully connected layer

Called also dense layer, it is usually added at the end of the network layers as a voting mechanism for the network output decision. Each neuron from the previous layer is connected to every neuron from this layer, and the output will be an array of values (votes). One of the characteristics of this type of layer is the possibility to stack many fully connected layers to improve the network performance since the input and output of this layer can be the same.

The third generation

The third generation of neural networks is the Spiking Neural Network (SNN), representing an electronic brain model. This neural network considers time in its operations and can process a significant quantity of data using a relatively small number of spikes.

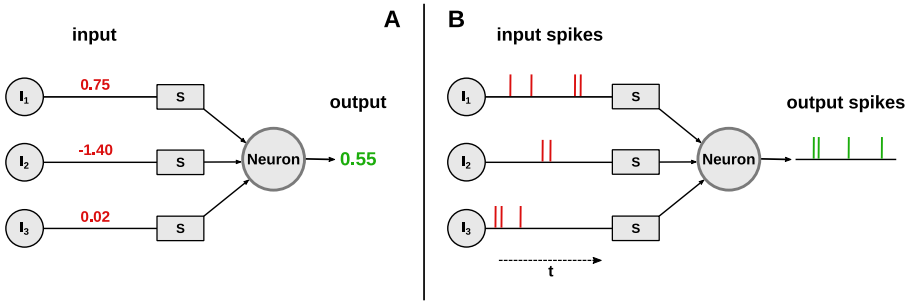

Figure 2.12: CNN (A) and SNN (B) input and output format

2.2.3 Hyperparameters

Hyperparameters represent the different parameters and variables that we need to set appropriately for the network to perform as expected. Unfortunately, these parameters are usually numerous, and it takes time to define them without experimenting with different combinations. Therefore, those parameters are set manually or by testing. In addition, some of them are learned by the network during training (the weights), for example:

- The number of layers and neurons to use.
- For each convolutional layer, we have the kernel size, padding, and stride value.
- For each pooling layer, we have the window size to use and the stride value.
- For each additional fully connected layer, the number of neurons.

Nowadays, we tend to have deep neural networks being used in different domains, and those networks have many hyperparameters, which takes a lot of time and energy

to be tuned. Moreover, the number of parameters changes from one layer to another, and the global number of the hyperparameters grows while going more profound in the network (see Table 2.1).

Model	Number of layers	Parameters	Trainable parameters
VGG16 [Simonyan and Zisserman, 2014]	16	138M	14.7M
VGG19 [Simonyan and Zisserman, 2014]	19	144M	20M
ResNet50 [Zagoruyko and Komodakis, 2016]	50	25.6M	25.5M
ResNet152 [Zagoruyko and Komodakis, 2016]	152	60.4M	60.2M
MobileNetV2 [Sandler et al., 2018]	53	2.25M	2.22M
GPT-3 [Brown et al., 2020]	96	175B	–

Table 2.1: Hyperparameters example in artificial neural networks

2.2.4 Training a neural network

Training a neural network is the critical phase that makes the network useful. We train a neural network by updating the weights of the synapses that we usually initialize randomly. Training a neural network may differ from one architecture to another. However, the goal is always to minimize a loss function.

Backpropagation

Backpropagation represents the class of algorithms widely used to train feed-Forward neural networks [Kelley, 1960]. This training consists of finding the proper values of

the existing parameters to minimize an error function. The process of going down through the loss function is called gradient descent.

2.2.5 Neural network architectures

The interconnection between the different neurons in a neural network offers a diversity of possible architectures and topologies [Miikkulainen, 2010]. The choice of a suitable architecture depends on the application. Moreover, we can classify those architectures into three general categories which are:

- Feed-Forward Neural Networks: This represents the commonly used architecture nowadays applications. It is created by using layers, and each layer contains neurons. There is no connection between neurons in the same layer or between this network topology's current and previous layers. Instead, the connections are between the current layer and the following layer neurons. The first layer is the input layer, the last one represents the output layer, and between them, we find the hidden layers. If there is more than one hidden layer, the network is called "deep".

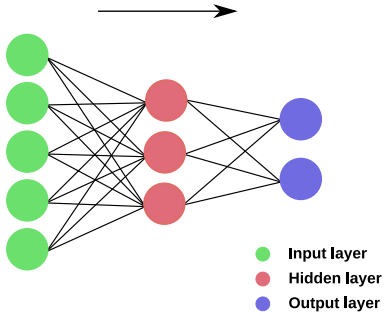

Figure 2.13: Feed-Forward Neural Network

- Recurrent Neural Networks: This network topology contains direct connections and connections oriented to the previous layers and additional information for the network. This architecture is hard to train sometimes due to its complex dynamics. Recurrent neural networks are used to model sequential data, and due to their hidden state, they are capable of remembering information for a long time. Moreover, different models were derived from recurrent neural

network such as reservoir computing [Tanaka et al., 2019], which is used for time-series analysis.

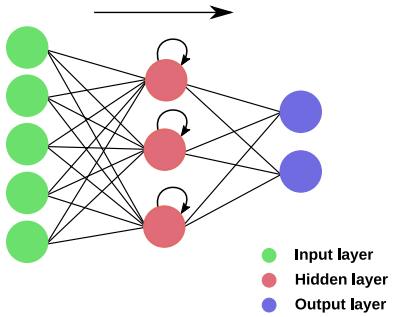

Figure 2.14: Recurrent Neural Network

- Symmetrically Connected Neural Networks: these are similar to recurrent neural networks. The only difference is that the connections between neurons, in this case, are symmetrical by having the same weight in both directions. This category is more restricted than RNN in their application due to the energy function they use. Having a symmetrically neural network with hidden units is called "Boltzmann machine," and without hidden units is called "Hopfield network".

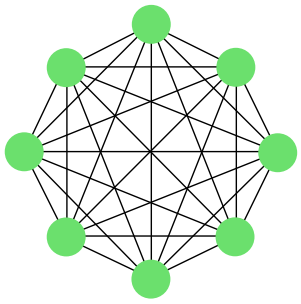

Figure 2.15: Symmetrically Connected Neural Network

2.3 Spiking neural network

The third generation of the neural network is inspired by the brain's functionality, making it possible to achieve some of the brain's properties on future computers,

such as parallelism, energy efficiency, and robustness to noise.

2.3.1 Neuron models

In SNN, neurons use spikes as a means of communication. They take spikes as input and the neuron will send a spike if its membrane potential value crosses a specific threshold. Different neuron models were proposed, we can mention as an example: Hodgkin-Huxley Model (HH) [Hodgkin and Huxley, 1952], Integrate and Fire Model (IF) [Abbott, 1999], Izhikevich Model [Izhikevich, 2003], Spike Response Model (SRM) [Gerstner et al., 1993], FitzHugh-Nagumo Model [FitzHugh, 1961], and Morris-lecar Model [Morris and Lecar, 1981]

Among these models, the most used are:

- **Hodgkin-Huxley Model (HH)**: the first spike neuron model, an electric model based on ions, contains Sodium, potassium, and leakage current.

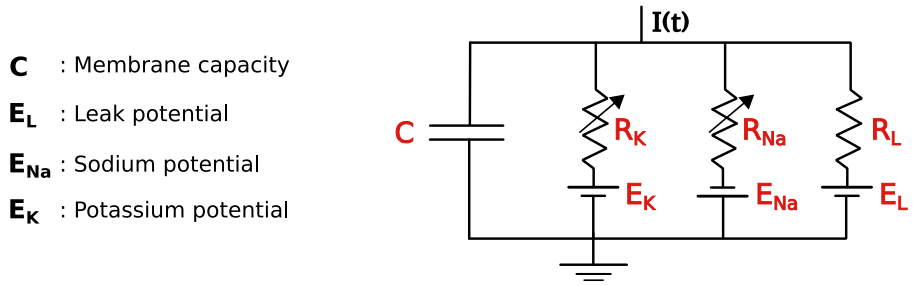

Figure 2.16: Hodgkin-Huxley model

- **Integrate and Fire (IF)**: this model is derived from the Hodgkin-Huxley model but with less complexity and computational cost. This model has several variations, such as Leaky Integrate and Fire (LIF) [Abbott, 1999], one of the most used models because of its efficiency in spiking neural networks and large-scale network simulations.

$$\tau_n \frac{dv}{dt} = -v(t) + RI_{syn}(t) \tag{2.3}$$

In equation 2.3, v represents the membrane potential, R the membrane resistance. τ_n is a time constant ($\tau_n = RC$), and $I_{syn}(t)$ represents the global input current.

- **Izhikevich model**: it's a model that combines the computational efficiency of the Integrate and Fire (IF) model and the biological plausibility of the Hodgkin-Huxley (HH) model. This model can simulate many spiking neurons in real-time. The advantages of this model are the simplicity of calculation and the possibility to produce different spikes models and bursts.

$$\begin{aligned} &\frac{dV(t)}{dt} = 0.04V(t)^2 + 5V(t) + 140 - u(t) + I(t) \\ &\frac{du(t)}{dt} = a.(b.V(t) - u(t)) \\ &if\ V(t) \geq 30mV,\ then \begin{cases} V(t) \leftarrow c \\ u(t) \leftarrow u(t) + d \end{cases} \end{aligned} \tag{2.4}$$

In equation 2.4, $V(t)$ represents the membrane potential, $u(t)$ is the membrane recovery variable. $I(t)$ represents the input current, a is the timescale of $u(t)$, and b is the sensitivity of $u(t)$. c and d represent the reset value of the membrane potential after a spike and $u(t)$, respectively.

- **Morris-Lecar model**: this model describes the three currents like the HH (Hodgkin-Huxley) model, but with just two dynamic variables, this model is useful for fast-modeling spiking neurons.

$$\begin{aligned} C\frac{dV}{dt} &= I - I_{Cav}(v) - I_{Kv}(w, V) - I_L(V) \\ \frac{dw}{dt} &= -[w - w_\infty(V)]/\tau_w(V) \end{aligned} \tag{2.5}$$

In equation 2.5, I_{Cav} represents calcium current, I_{Kv} represents potassium current, and the leak current is represented by I_L. w represents the activation of I_{Kv}.

2.3.2 Information coding in SNN

In SNN, spikes are used as a means of communication. The processing of the spikes by the brain is one of the questions that are still open; how information is represented depends on the specific region of the brain, and sometimes, more than just one encoding type is present in the same area [Thorpe et al., 2001][Brette, 2015]. Different information encoding exists, such as:

- **Frequency coding**: There are two forms of this encoding: *Spike count rate*: which depends on the number of spikes in a given period. *Time-dependent firing rate*: depends on the average number of spikes in a small interval $\triangle$t (Figure 2.17).

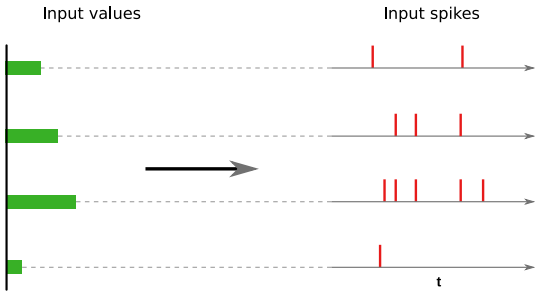

Figure 2.17: Frequency coding

- **Temporal coding**: This type of information encoding depends on the spike timing, which carries information (Figure 2.18).

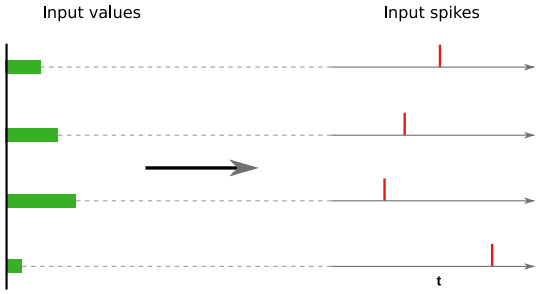

Figure 2.18: Temporal coding

2.3.3 Learning in spiking neural networks

Since spiking neural networks are bio-inspired, the learning in this type of network is also inspired by brain activity. However, spikes (discrete events) are not differentiable, which prevents us from using backpropagation as it is in spiking neural networks. Instead, local learning rules are preferred due to their energy efficiency and performance in unsupervised learning tasks, called "Hebbian rules." However, in recent years, the surrogate gradient descent was proposed, which allows the use of backpropagation and global learning rules to train SNN [Neftci et al., 2019].

Hebb's rule

Donald Hebb proposed Hebb's rule in 1949 [Hebb, 1949], which indicates that "neurons that fire together, wire together," which means that the connection weights are updated based on the neural activity of the neurons in both ends. This rule is the foundation of the bio-inspired learning rule in artificial neural networks, mimicking how our brains work and neurons interact. This rule is composed of two parts:

- First, we calculate a (equation 2.6), which equals the sum of input values multiplied by their weights ($e_{1..n}$) minus the threshold S. Based on the calculated value of a we can conclude the value of x, using the following condition:
  - if $(a > 0) : x = +1$
  - else : $x = -1$

$$a = \sum_{i=1}^{n} (w_i * e_i) - S \tag{2.6}$$

- Second, we calculate the new weight value based on 2.7. w_{ij} is the old weight value, x_i and x_j represent the value x calculated for the pre-synaptic neuron i and post-synaptic neuron j, and the positive constant value μ.

$$w_{ij}(t+1) = w_{ij}(t) + \mu * x_i * x_j \tag{2.7}$$

Spiking Time Dependent Plasticity (STDP)

STDP is a Hebbian learning rule [Markram et al., 1997]. It is based on the activity of both presynaptic and postsynaptic neurons. The order of the activation is essential for the weight update. Thus, the concept of weight update over time is called LTP (long term potentiation) and LTD (long term depression). Different types of STDP rules were presented, which share the same idea (Hebb's rule) but are different when it comes to the number of parameters and how it works, for example:

- Pair-based STDP [Babadi, Baktash and F. Abbott, L., 2016]: the synaptic weight modification depends on the pre-synaptic and post-synaptic neuron activity (equation 2.8).
- The triplet STDP [Pfister and Gerstner, 2006]: the synaptic weight update in this model depends on the triplet pre-post-pre synaptic or post-pre-post neuron activity.
- Reward-modulated STDP [Legenstein et al., 2008]: the synaptic weight update, in this case, combines the unsupervised STDP with a reinforcement signal used for modulating the synaptic weights, acting as a reward signal.

$$\Delta w = \sum_{pre=1}^{n} \sum_{post=1}^{m} W(x)(t_i^{post} - t_j^{pre}) \tag{2.8}$$

2.3.4 Winner takes all

WTA is a technique used during the training of spiking neural networks [Lazzaro et al., 1989]. It prevents all the other neurons with lower activity to fire when the first one with the highest activity fires. This prevention can happen between all the neurons from one layer or between neurons that cover the same region in the input data. To apply the WTA in a simulation or on hardware, we add inhibition neurons that send negative current once they receive the first spike, which prevents the rest of the neurons from spiking. Thus, WTA helps the network neurons learn different patterns and avoid having identical neurons during and after the training.

Homeostasis

Homeostasis is a regulation process of the neuron's activity used during training with a Hebbian learning rule. This process contributes to preventing long-term perturbations of the neuron activity and helps to keep the average neuron's activity in the network stable. For example, we can allow different neurons to fire by preventing any neuron from firing excessively by using homeostasis. Furthermore, in training a neural network, using homeostasis enables the neurons to learn different patterns without being penalized by a dominant neuron [Marder and Goaillard, 2006]. In artificial neural networks, homeostasis can be provided by increasing or decreasing the neuron threshold based on the neuron activity, which produces the same effect as biology.

2.4 Spiking neural networks simulation

Simulation is a must-have in producing an efficient SNN for neuromorphic hardware. It mainly consists of implementing a specific neuron type, learning mechanism, and network topology and observing the network performance. Thus, there are two main possibilities to simulate a spiking neural network: using simulators that support SNN or by using one of the existing neuromorphic hardware. Using the software approach is enough since we can simulate every network component most of the time. Still, if the goal is to check any hardware-related properties of the network, then neuromorphic hardware may be more suitable and realistic.

During simulation, we introduce a dataset to train the network, and this dataset can be any data (image, audio, video, etc.) as presented in Figure 2.19. Thus, one of the crucial steps before training the network is to prepare the dataset, perform any needed pre-processing, and remove any missing data, which is very common on real-life data.

2.4.1 Software-based simulators

Software-based simulators exist in different programming languages, and we can find two main categories: event-driven simulators and clock-driven simulators. Event-

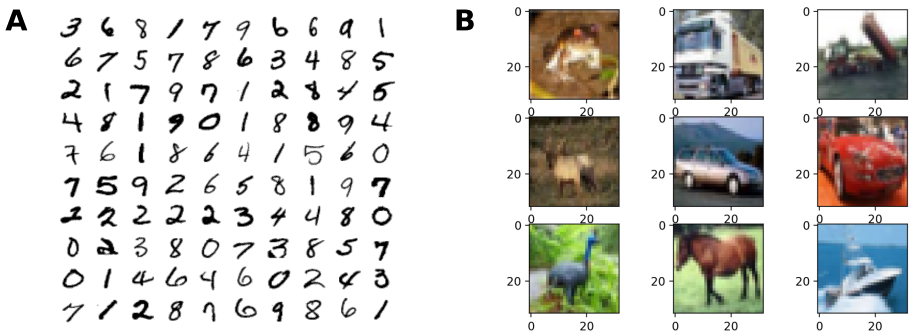

Figure 2.19: MNIST (A) and Cifar-10 (B) dataset

driven simulation is more adapted to how neurons work since the calculation happens only when activity happens at a neuron. This approach can help reduce energy consumption and time by avoiding unnecessary computation. Event-driven simulation usually doesn't depend on the hardware it is running on. We can deploy it on multiple nodes. However, we also need to provide a synchronization mechanism for the timing of the events. The networks with a learning rule that requires the exact spike time perform better in this simulator. We can take as example of event-driven simulators N2S3 [Boulet et al., 2017] and NEURON [Hines and Carnevale, 1997]. The clock-driven simulation does the synchronous update for all the neurons at every step. Such an approach is easier to implement, especially on GPUs, to parallelize the learning process, which helps speed up the simulation. Such an approach takes advantage of the remarkable advancement in GPU technology. However, the choice of the update step for a clock-driven simulation impacts the network performance. If the selected step is big, it may lead to a loss in precision or a higher computation cost if the selected step is small. As an example of simulators we have Nengo [Bekolay et al., 2014] and Brian [Goodman and Brette, 2008].

In this manuscript, we use an event-driven simulator (N2S3) for shallow networks and a clock-driven simulator (Nengo) for multi-layer networks.

2.4.2 Hardware-based simulators

The hardware-based simulation uses neuromorphic hardware, which is suitable for this type of neural network. Usually, when working with neuromorphic hardware (like SpiNNaker [Furber et al., 2014] or Loihi [Davies et al., 2018]), we use libraries

explicitly made to communicate with the hardware, such as PyNN [Davison et al., 2009], a simulator-independent language for building neuronal networks models. Another lower-level approach compared to the previous one using FPGA's based boards (like the ZedBoard using the Xilinx programmable SoC). Using this approach, we can speed up the simulation and collect hardware-related information about the network performance and energy consumption.

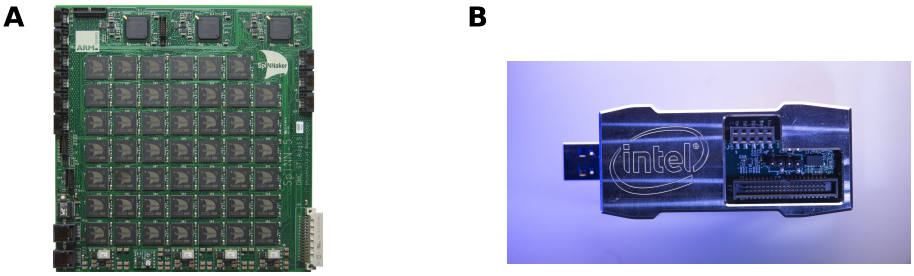

Figure 2.20: SpiNNaker (A) and Loihi (B)

2.5 Visualization

Vision is the dominant and powerful sense among the five channels we use to sense the world, and it provides more information than all the other senses combined. Decades ago, the potential of representing quantitative information in a way that our eye can easily intercept was recognized by early pioneers in data visualization. Representing numbers as text slows down our ability to process information since it has to be processed one at a time, and we quickly become lost once we try to analyze it due to our short memory. However, if we could accurately visualize this information, we can understand, explore, and examine it in a previously impossible way. In 1786, William Playfair (1759-1823) published his first early graph (Figure 2.21), which was a demonstration of what we can achieve by visualization, and opened the door to new ways of exploring the meanings in data that enables the perception of trends, patterns, and exceptions.

By presenting quantitative data in a visual form, we extend the capacity of our memory by providing in front of our eyes a quantity of data that usually cannot be held in our minds altogether. As a result, it makes it easier to understand trends and exceptions and present them to others. While visual perception significantly

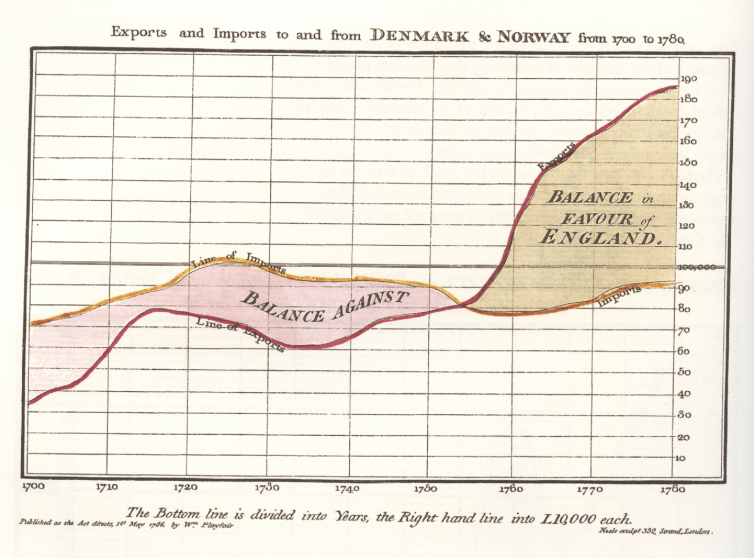

Figure 2.21: Playfair's first graph in 1786

impacts how information is perceived, the correct representation of data is essential by choosing the proper graphs and the right quantity of data to show.

We analyze data to get a precise, accurate, and better understanding, which will help us make better decisions. One of the ways of going from analysis to conclusion is by asking the right questions. We can divide the questions into two categories: descriptive and predictive. We use descriptive questions to understand what is happening and what is causing this to happen; those questions help us understand what we see. Based on the answers of the first category, we can ask the predictive questions, which concern what we want to happen and what are the actions that can lead to such an outcome. Such questions can help us shape the result we are looking for when doing data analysis. [Few, 2009]

To create visualizations for analysis purposes, we can use the analytical design principles presented by Edward Tufte in his book *Beautiful Evidence* [Tufte, 2006]. Analytical design principles come from analytical thinking to help reflect on evidence. In his book, Edward Tufte uses one figurative map created by Charles Minard in 1869 (Figure 2.22) to describe and present the six principles. The figurative map represents the successive losses in men of the French army in the Russian campaign 1812-1813. Minard utilizes six variables in different visual encodings: the direction of the army's movement, its two-dimensional location (latitude and longitude), the

size of the army, and temperature during the withdrawal from Moscow. The six principals of analytical design are:

- **Comparisons:** show comparisons, contrasts, and differences. If the purpose of a visualization is to assist thinking, it should make it possible to compare. For example, in Figure 2.22, we can see Minard compares the size of the army in the different stages of the invasion, which gives us an idea of the losses in men.

- **Causality, mechanism, structure, explanation:** it is essential to show causality in visualization for analysis purposes. We can accomplish this by simply organizing data in a way that may provoke thoughts about cause and effect. For instance, in Figure 2.22, having the temperature, the date, and the name of the different locations added to the map helps explain the change in the size of the army during the march (Berezina river, the low temperatures, etc.).

- **Multivariate analysis:** the world we seek to understand is deeply multivariate, yet we usually use 2D-dimensional representation. For visual analysis, it is always better to use more than two variables or dimensions, similar to what we have in Figure 2.22 which uses six variables in different visual encodings.

- **Integration of Evidence:** it is essential to integrate words, numbers, images, and diagrams into a visualization. This integration is beneficial for exploratory data analysis. We can provide this using layers and filtering techniques to guarantee a clean look and bring other information into the scene. This integration can also be seen in Figure 2.22.

- **Documentation:** we need to provide a complete description of the representation, a clear title, indicate the authors, document the data sources, show full measurement scales, and point out relevant issues. We simplify the analysis process by providing that information, just like Minard did in the figurative map.

- **Content Counts Most of All:** analytical presentations ultimately stand or fall depending on their content's quality, relevance, and integrity. Therefore, we need to provide any additional information that can reinforce the presentation's quality or integrity.

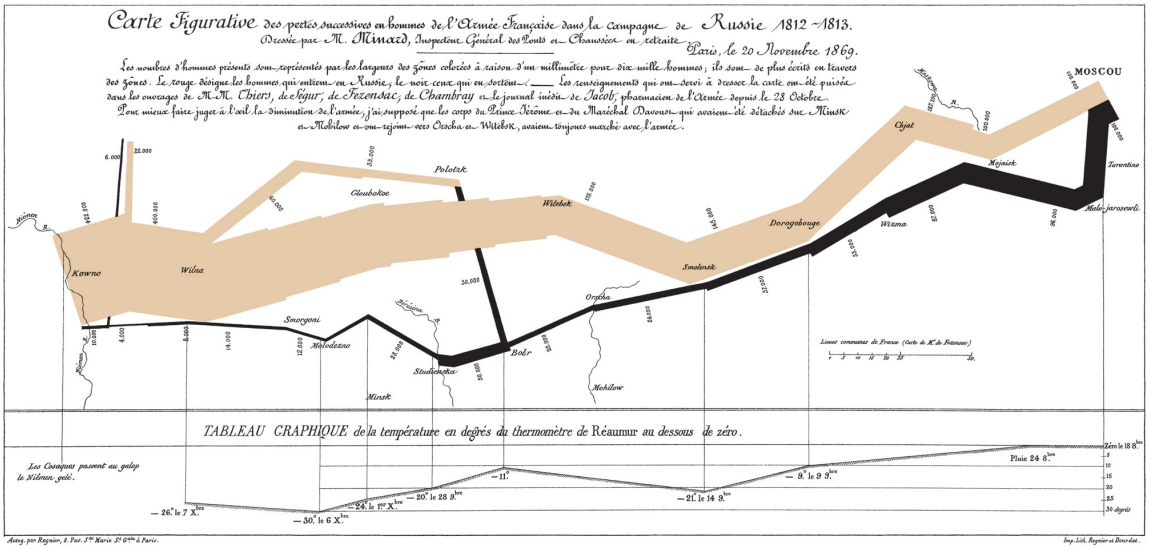

Figure 2.22: Figurative map of Napoleon's march on Moscow in 1812

2.5.1 Data science

Machine learning in general and neural networks, in particular, are considered data-driven models. Data preparation is crucial before exploiting it for training. The quality of this step has a significant role in defining the neural network's performance at the end. It is commonly considered that 80% of our time working with neural networks is spent preparing data, especially when working with real-world data. Due to the massive quantity of data that we can acquire nowadays in different fields, the incomplete and unstructured nature can be a huge challenge to explore it using neural networks; that's where data science comes in. Data science uses scientific methods, processes, algorithms, and systems to extract insights and knowledge from data, structured or unstructured. It helps understand the data existing correlations and spot any missing information. As Carly Fiorina, former CEO of Hewlett-Packard, once said: "The goal is to turn data into information, and information into insight." [1]

[1] "Information: the currency of the digital age" Oracle OpenWorld, San Francisco, 2004

2.5.2 Psychological principles for visual display

The way we present a visualization impacts how the recipient processes it. For example, this recipient can be an audience in a conference or a scientist working with data. We can apply the following psychological principles to guarantee a certain degree of clarity while creating those visualizations. Stephen M. Kosslyn presented those principles in his book *Graph Design for the Eye and Mind* [Kosslyn, 2006]. We can adapt the principles to any visual display, and in particular for large scale data visualization:

- **Principale of relevance:** communication is most effective when neither too much nor too little information is presented. Therefore, defining the exact clear message before designing a visualization will be essential to select what information to include.
- **Principale of appropriate knowledge:** communication requires prior knowledge of pertinent concepts, jargon, and symbols. Hence, visual analysis is easier to apply if the visualization is built on top of appropriate information known to the recipient, and it's easier to understand a new idea if we present it as growing out of a familiar one.
- **Principale of salience:** attention is drawn to significant perceptible differences, and our focus goes first to the most prominent features of visualization. It can be a particular color, the difference in contrast, or a shape. That's because the superior colliculus (for vision) in our brain is functional immediately after birth, whereas the other parts of the brain that requires shifting voluntarily the attention become active a couple of months after birth. This attention needs to be oriented to the essential information.
- **Principale of discriminability:** two properties must differ by a large sufficient proportion, or they will not be distinguished. This principle applies to the size, thickness, brightness, and density of dots. A particular case of discriminability occurs when characters must be large enough or differ in color, contrast, or weight to stand out from the rest and therefore be noticed. Otherwise, it will be much more challenging to see.
- **Principale of perceptual organization:** people automatically group elements into units, which they then attend to and remember. We pay attention to patterns registered by the same input channel, and these channels can be the object orientation or any changes in light.
- **Principale of compatibility:** a message is easiest to understand if its form is compatible with its meaning. The appearance of a pattern should be consistent

with what it symbolizes, and this can mean different aspects, such as appearance-meaning correspondence, cultural conventions, or perceptual distortion.

- **Principale of informative changes:** people expect changes in properties to carry information. Usually, there is no new information if there is no change in the actual pattern or visible object. However, suppose there is any additional information. In that case, we expect to see a visible change on display, which can be a change in color, light, object position, adding or deleting a line, etc. The same thing when we see a change in display. We expect this change to convey a piece of information. If not, this will be considered as a distraction instead.
- **Principale of capacity limitations:** people have a limited capacity to maintain and process information and will not understand a message if too much information must be retained or processed. One of the significant aspects of this principle is the short-term memory limits since we can only hold in mind about four units simultaneously. Therefore, it will be hard to understand a display if it requires too much information in mind. Processing limits is another aspect of this principle. It will be harder to understand if a display requires too much effort to be processed and understood. For example, a visual display may require mental transformation, comparison, or averaging operations, which requires additional effort to understand.

2.5.3 Visualization in neural networks

Due to the immense amount of data available nowadays in different domains and the increasing complexity of neural networks, understanding how the network reacts to specific inputs and follows the training process is not a trivial task. Data visualization is one possibility to understand better how the network works. By using the right visualizations, we can expand our knowledge of the network, which takes us a step further into having explainable neural networks.

In convolutional neural networks (CNN), visualization has proven to be very helpful in understanding the network performance and outputs. In the multimedia field, neural networks achieve state-of-the-art performance in most of the tasks like segmentation, classification, and object detection [Noh et al., 2015],[Zeiler and Fergus, 2014], the possibility to visualize components of the network helped validate the network performance and explain the decisions. Visualization in neural networks can also include neurons, synapses, input activity, and learning. However, one of the actual challenges of neural networks is the need for a lot of data to train the

network and usually a more complex architecture for better performance. This situation makes it challenging to perform data analysis since the massive quantity of data requires extra attention to the selected visualizations and how much data we present at once to avoid overwhelming the user with unnecessary data that will make the analysis process harder [Ali et al., 2016].

2.6 Conclusion

In this chapter, we presented a background on neural networks, such as the different generations, the network components, the existing learning rules, and neurons. We also introduced spiking neural networks and the various information codings. In the second part, we introduced the visualization aspect of this work, a brief introduction on visualization in general, data science, and visualization in a neural network.

In the rest of this manuscript, we present our contributions toward better spiking neural networks using visual analysis. First, Chapter 3 presents a group of SNN simulators and what they provide as visualization techniques. Moreover, in this chapter, we compare the simulators in terms of their features and the quality of the visualizations. This comparison concludes that we need a better way to visually analyze the network activity. Next, in chapter 4, we present VS2N, a tool for dynamic over time analysis of spiking neural networks. Using VS2N, we can analyze, understand, validate, and propose improvements for SNNs. Since the evolution of the network takes place over time, it is essential to consider the time aspect during the analysis, which is supported in VS2N by the ability to move in time, unlike other existing tools for visual analysis. Finally, chapter 5 presents a dynamic approach to compress a spiking neural network during training, which was proposed based on the visual analysis conducted using VS2N. The compression is applied on synapses, depending on the synaptic weights and a dynamic threshold. Using this approach, we can compress the network up to 80% and preserve similar performance to a non-compressed network.

Chapter 3

Visualization techniques in SNN simulators

Neural networks are the first thing that comes to mind when discussing artificial intelligence because of their significant impact in different fields. Neural networks have evolved since their first appearance. It was able to solve linear problems only (first generation) than solving non-linear problems (second generation) using deep neural networks, to finally the third generation, which is bio-inspired. However, the near end of Moore's law and the limitation of the Von Neumann architecture prevent us from having suitable hardware for neural networks, which results in immense energy consumption and carbon footprint. On the other hand, neuromorphic architectures are a promising way to overcome those limitations. Spiking neural networks on neuromorphic hardware can reduce energy consumption due to the spike-based communication, which is non-trivial to implement on a binary-based architecture (Von Neumann).

Neuromorphic architectures implement spiking neural networks (SNNs), which model at a precise level how our brains work. However, one of the main problems that prevent us from getting the best of SNNs and neuromorphic architectures in terms of performance is the lack of a clear and complete understanding of SNN behavior, especially what makes learning efficient and how to detect any issues during the activity. One way to answer that is by visual analysis of the activity in spiking neural networks during the simulation. This chapter presents some of the simulators used to train SNN and compares the visualization techniques proposed by them for analysis purposes. This comparison aims to determine if the current visualizations offered by the simulators are enough to analyze the network or not [Elbez et al., 2018].

3.1 SNN Simulators

Simulation is a crucial step in every experiment. Before implementing any architecture, we have to pass by the simulation to conduct the required tests. Many simulators have been created to help researchers to test their hypotheses. In this chapter, we are interested in simulators that offer a visualization for this type of network: Neuron, Brian, Nengo, Neuronify, Simbrain, N2S3, and NEST. We also mention another category of simulators that appeared in recent years, which are based on PyTorch simulator [Paszke et al., 2017].

Standard differential equations essentially define neural network learning, but due to the discrete nature of spikes, designing an efficient simulation of spiking neural networks is a non-trivial problem. There are two families of simulation algorithms used for SNN: event-based simulation and clock-based one. Synchronous or clock-driven simulation simultaneously updates all the neurons at every tick of a clock, which is easier to implement, especially on GPUs for efficient execution of data-parallel learning algorithms. On the other hand, event-driven simulation behaves more like hardware, in which conceptually concurrent components are activated by incoming events (or spikes) [Brette et al., 2007].

Event-driven simulation is particularly suitable for untethered devices such as neurons and synapses since the nodes go into sleep to preserve energy during the absence of activity. Energy-aware simulation needs information about active hardware units and event counters to establish the energy usage of each spike and each component of the neural network. The event-driven execution model is independent of the hardware architecture which is running on. So, event-driven simulators can naturally run on a grid of computers, with the caveat of synchronization issues in the event timings management. Furthermore, as the learning mechanisms of spiking neural networks usually incorporate spike time, the choice of the clock period for a clock-based simulation may lead either to imprecision or to a higher computational cost.

Another category of simulators has emerged in recent years, a category based on the PyTorch simulator. PyTorch is a clock-based simulator created mainly for conventional neural networks simulation using backpropagation and deep learning. It was adapted in academia as a first choice due to the flexibility and the ability to implement custom components since it is based on Python. A couple of simulators were created on top of PyTorch by adding support for SNN to take advantage of what

PyTorch offers when working with SNNs. Applying the backpropagation requires differentiable activation functions. However, spikes are "all-or-none" events, which cause discontinuities, and it is impossible to use on SNN at this state. One of the tricks to make the backpropagation algorithm work with SNN is using a sigmoid function on the spikes to calculate derivation and the gradient, called surrogate gradient learning [Neftci et al., 2019].

3.1.1 NEURON

NEURON is one of the oldest simulators. It was developed in 1997 by Michael Hines et al. at Yale, and Duke university [Hines and Carnevale, 1997]. NEURON offers the possibility to model individuals or networks of neurons. The primary scripting language of NEURON is HOC (High Order Calculator) programming language [Kernighan, 1984]. However, a Python interface is also available. NEURON can be used with the possibility of loading programs from a file or written in a shell. In addition, it supports parallelization using the MPI protocol [Forum, 1994].

We can resume the visualization that NEURON offers for the users in various graphs as shown in Figure 3.1, graphs such as membrane potential graph and dendrite voltage graph, and a reconstruction of the cell shape. Each visualization is presented in a separate window. The interface did not change with the new versions of the simulator.

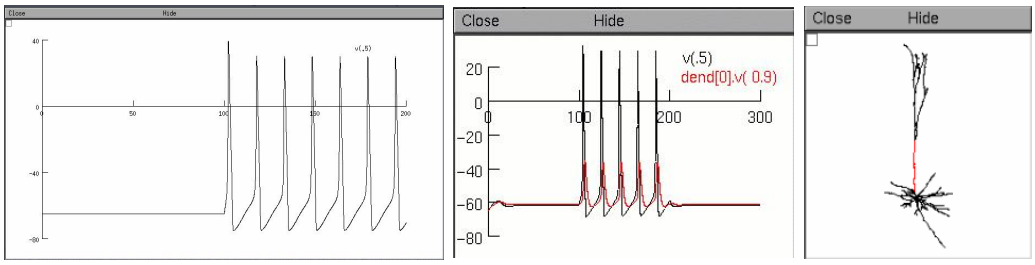

Figure 3.1: Visualization examples in NEURON [Hines and Carnevale, 1997]

3.1.2 Brian

Goodman et al. first published about Brian in 2008 [Goodman and Brette, 2008]. This tool made the coding of spiking neural networks fast, easy to use, and flexible.

In addition, Brian is written in an interpretative language, which is effective in many situations thanks to vectorized algorithms [Brette and Goodman, 2011]. However, the first Brian version is unsuitable for massive simulations that need significant resources or simulating detailed biophysical models.

Brian offers many possibilities of visualization like shown in Figure 3.2, which can automatically change the visualization depending on the size of the network. Those visualization techniques include the membrane potential graph, spikes plot, firing rate graph, and synapses connections representation.

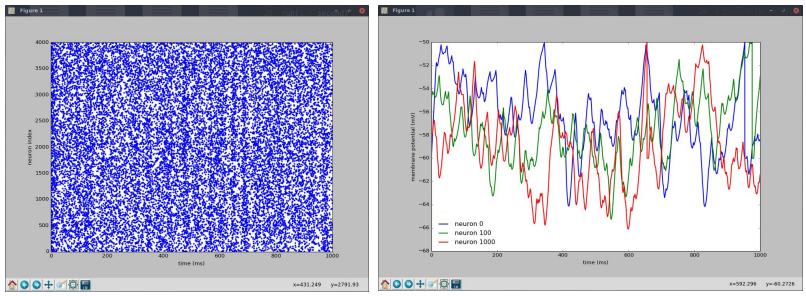

Figure 3.2: Visualization examples in Brian [Goodman and Brette, 2008]

In 2019, Brian 2 was presented [Stimberg et al., 2019]. This version has improved in performance and used Matplotlib library [Hunter, 2007] for visualization. Brian2GeNN was introduced recently (2020) [Stimberg et al., 2020]. This simulator version uses graphics hardware to accelerate the simulation of spiking neural networks.

3.1.3 Nengo

Nengo is a python library used to create and simulate very large-scale neural networks. It can create spiking neural simulations, and other sophisticated types in a few lines of code [Bekolay et al., 2014]. Nengo offers the possibility to define neuron type and learning rules, create and execute deep neural networks and simulate on hardware devices such as Spinnaker [Furber et al., 2014] or Loihi [Davies et al., 2018]. Nengo offers many types of backends that we can use for simulation: Nengo, Nengo_ocl, Nengo_mpi, Nengo_distilled. For the hardware simulators, Nengo offers two types which are Nengo_brainstorms, Nengo_spinnaker, and Nengo_loihi.

Nengo is composed of a server written in python connected to Nengo core and

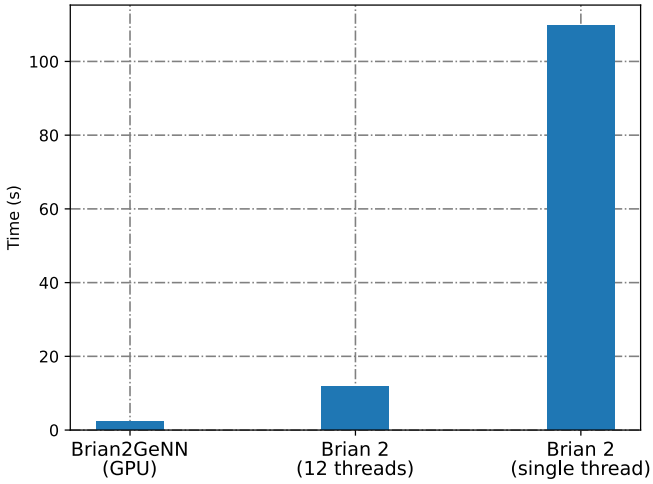

Figure 3.3: Comparison between different versions on the same task [Stimberg et al., 2020]

an interactive interface for users to manipulate. This interface is based on web technologies like HTML, D3.js, and jQuery.

When it comes to user experience, Nengo provides a unique experience for users to interact with the code directly on the interface and see the changes in real-time with an acceptable degree of interactivity. Furthermore, Nengo offers a variety of visualization techniques that we can use to follow the network activity (Figure 3.4), like membrane potential graph, spikes plot, and activation patterns.

Nengo was used to build Spaun, the world's largest functional brain model. Spaun currently contains 6.6 million neurons and over 10 billion synapses.

3.1.4 Neuronify

Neuronify is an educative tool created to simulate neurons and neural network behavior. We can use it to combine neurons with different connections and see how changes in individual neurons can lead to behavior change in big networks, it is developed in C++ and QML using the cross-platform application framework Qt by Ovilab [Dragly et al., 2017].

In Neuronify, exploring and creating neural networks is made easy by dragging

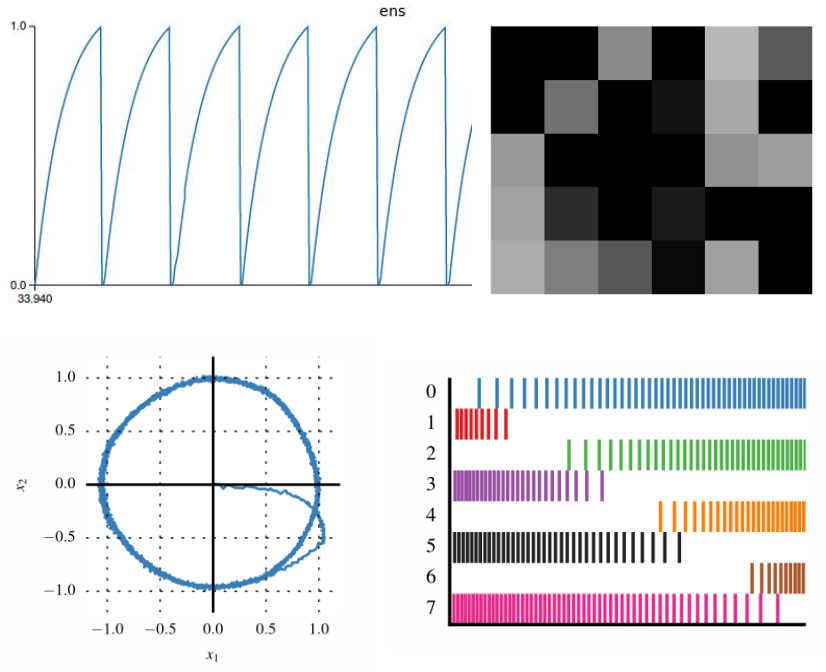

Figure 3.4: Visualization examples in Nengo [Bekolay et al., 2014]

and dropping the elements on the screen. It is available to use on desktop or mobile devices.

Neuronify offers a friendly interface and interactivity for users. For the network visualization, Neuronify has three techniques: Spike detector plot, firing rate plot, and the membrane potential graph. In Figure 3.5, we can see some visualization examples.

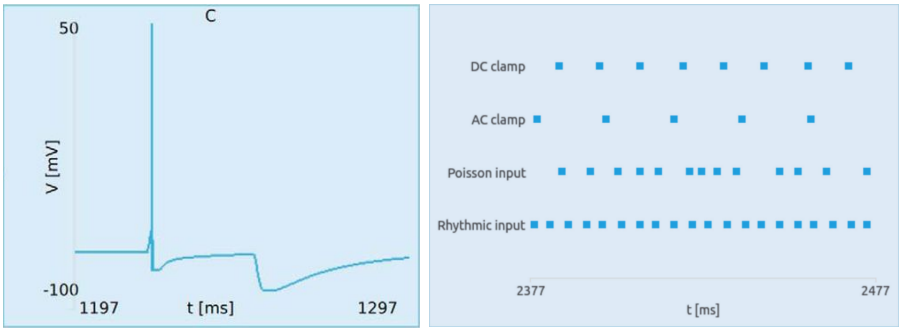

Figure 3.5: Visualization examples in Neuronify [Dragly et al., 2017]

3.1.5 Simbrain

Simbrain is a tool for constructing artificial neural networks, written in Java and under GNU license. This simulator concerns more biology or the medical field scientists, which is why we can observe many biological representations and networks. Simbrain incorporates the philosophy of making things easier for users. It comes with a variety of examples and types of networks with descriptions, in an organized way, and an easy interface to interact and work with [J, 2008].

For the visualization aspect, Simbrain provides detailed documentation. Moreover, due to the general nature of this simulator, it offers many techniques for the user to choose from, as shown in Figure 3.6, like Spikes plot, Spike visualizer, membrane potential variation graph, and the possibility to combine more than one technique.

In 2016, Tosi et al. [Tosi and Yoshimi, 2016] presented Simbrain 3.0, which is an improvement version over the old version, in terms of flexibility and visualization (see Figure 3.7).

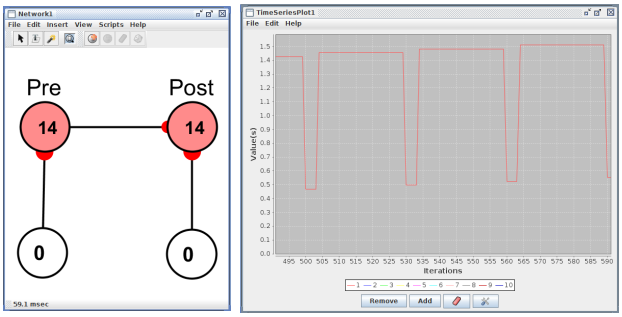

Figure 3.6: Visualization examples in Simbrain [J, 2008]

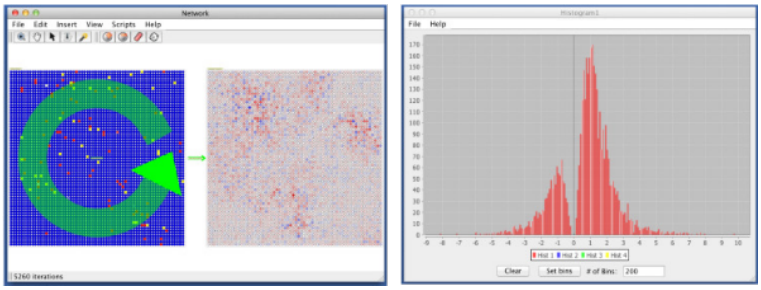

Figure 3.7: Visualization examples in Simbrain 3.0 [Tosi and Yoshimi, 2016]

3.1.6 N2S3

N2S3 (Neural Network Scalable Spiking Simulator) is a simulator created to simulate neuro-inspired hardware accelerators. It is written in Scala and Akka, giving it all the scalability and portability features Scala offers. N2S3 is an event-driven simulator based on exchanging messages between concurrent actors to mimic the spikes exchange between neurons [Wyatt, 2013]. Being flexible, extensible, and scalable makes the integration of new models or tools easy. [Boulet et al., 2017]

N2S3 has been developed from the ground up for extensibility. As a result, N2S3 allows the modeling of various neuron types and synapse models, different network topologies, various learning procedures, various reporting facilities, and user-friendly with a domain-specific language to express the experiments the user wants to simulate. It is available online as open-source software. It comes as a library with some classic experiments ready to use, such as handwritten digit recognition on the MNIST dataset [LeCun et al., 1998] and the highway vehicle counting experiment [Bichler et al., 2012].

N2S3 allows users to observe simulation outputs through network observers, which can vary from textual loggers to dynamic visualizations. N2S3 also offers a synaptic weight evolution visualizer, spike activity map, and the visualization of actual input data (see Figure 3.8).

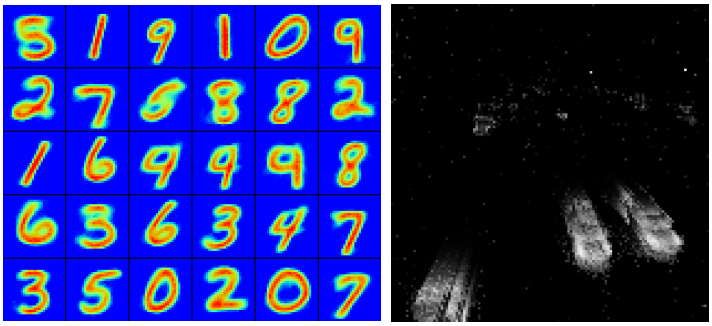

Figure 3.8: Visualization examples in N2S3 [Boulet et al., 2017]

3.1.7 NEST

NEST is an SNN simulator coordinated by the NEST Initiative [Gewaltig and Diesmann, 2007]. It has a focus on the network dynamics, size, and structure rather than the individual neurons. It is considered one of the most attractive tools for SNN simulation due to its ability to work with any network size. NEST offers two ways of creating simulations. First, using it as a Python library (PyNEST) that provides many commands to access NEST's simulation kernel and conduct a simulation. Second, by using the stand-alone application (NEST). Having the simulation kernel written in C++ provides NEST with the speed and the possibility to have optimized simulations. The used simulation language is SLI [Eppler et al., 2009] which is considered a stack-oriented language.

NEST does not offer a graphical interface for the user to create, edit and manage the simulation. Instead, it's done by either working with the command line provided by this tool or using NEST as a Python library.

NEST offers the possibility to view the network in text form to check the network structure, with the help of Python packages like Matplotlib [Hunter, 2007], and NumPy [Harris et al., 2020]. Furthermore, NEST offers a variety of visualization techniques mainly based on Matplotlib, as shown in Figure 3.9, like Membrane

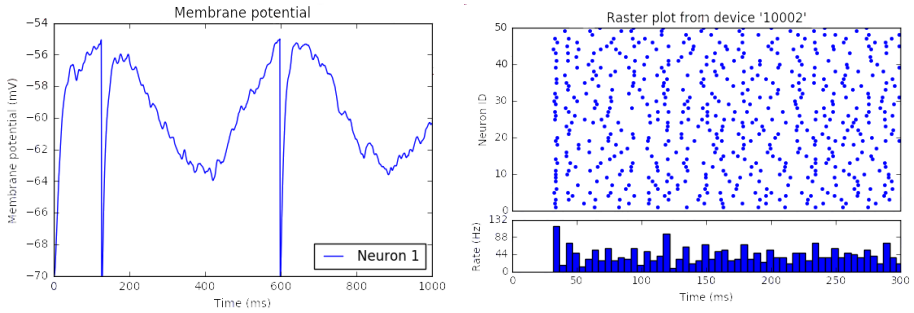

Figure 3.9: Visualization examples in NEST [Gewaltig and Diesmann, 2007]

potential representation, Synaptic weights representation, and neurons spikes plot. Moreover, other tools exist based on NEST simulations to conduct a visualization, like VisNEST [Nowke et al., 2013] which was able to visualize a network with 20 million neurons and many more synapses in the medical field.

3.1.8 PyTorch-based simulators

Since PyTorch is Python-based, the Matplotlib library is usually used for any visual display. Therefore, we can group all the PyTorch-based simulators for visual analysis. SpykeTorch [Mozafari et al., 2019], BindsNET [Hazan et al., 2018], and Norse [Pehle and Pedersen, 2021] are examples of tools made on top of PyTorch, which support gradient-based algorithms for SNN, using surrogate gradient learning.

3.2 Simulators comparison

Many visualization techniques and models exist in data science, especially with the rise of Big Data and the need to analyze such a considerable quantity of data for a better user analysis experience. Furthermore, a good and helpful visualization may lead to highlighting potential models and network and architecture improvement, so having the possibility to judge and compare a visualization technique is very important.

From a technical point of view, the simulators presented have appealing charac-

teristics like the used technology and support for large-scale simulations and more, shown in Table 3.1.

	Technology	Type of simulation	Parallelism	Scalability	GUI	CMD
NEURON	C, C++, python	Event-driven	+	+	+	+
Brian	Python, NeuroML, PyNN	Clock-driven	+	-	-	+
Nengo	Python, Numpy, HTML	Clock-driven	+	+	+	+
Neuronify	C++, Qt	Clock-driven	-	-	+	-
Simbrain	Java	Clock-driven	-	-	+	-
N2S3	Scala, Akka	Event-driven	-	+	-	+
NEST	C++, Python, SLI	Event-driven & Clock-driven	+	+	-	+
Pytorch-based	Python	Clock-driven	+	+	-	+

Table 3.1: Technical review

For the visualization aspect, we use seven criteria for comparison. Those criteria are inspired by the work of Freitas et al. [Freitas et al., 2002], and Stephen Few [Few, 2017] and another important criterion in our case, which is the interactivity level. We divide the criteria into three categories: Informative, Emotive, and finally the Interactivity.

The first category, called *Informative*, contains five criteria. Usefulness: whether this visualization is providing meaningful information or not. Completeness: if the visualization includes all the required components to be easy to understand or not. Perceptibility: whether it is straightforward to understand or not. Truthfulness: represents the degree of accuracy and validity of the visualization. Finally, Intuitiveness represents the degree of familiarity of this visualization technique for the user. The second category, called *Emotive*, contains one criterion chosen for our case, which is Aesthetics: it concerns the design part and quality of the visualization technique. Finally, the Interactivity: which involves the level of interactivity provided by this visual display, includes the different operations we can do on the graph (save, scale, zoom, etc.).

	Usefulness	Completeness	Perceptibility	Truthfulness	Intuitiveness	Aesthetics	Interactivity
NEURON	++	-	++	+	++	-	+
Brian	++	++	++	+	++	+	+
Nengo	++	+	++	++	++	++	++
Neuronify	+	-	++	+	++	++	++
Simbrain	++	++	++	++	+	+	+
N2S3	++	+	++	++	++	+	-
NEST	++	+	++	++	++	+	-
Pytorch-based	++	+	++	++	++	+	+

Good = ++ | Medium = + | Bad = -

Table 3.2: Visualization comparison

3.2.1 Discussion

From Table 3.1, we can see that various programming languages are used in the development process. However, Python is the most dominant because it is a scripting language easy to use and the vast libraries it offers for machine learning related tasks. Another thing to mention is the parallelism and large-scale simulations, which are not supported by all the simulators. This limitation may affect the simulator's performance and limit its usability while working with large networks and big data. We can see also that some simulators do not provide a graphical interface (GUI), which may affect the learning experience for the new users and makes it harder to debug and follow the execution. However, they provide command line execution (CMD), which may also be helpful when working with large networks since the lack of a graphical interface may reduce the resources needed for the simulation to run. Therefore, allowing the execution of more large networks.

From Table 3.2, we can see that all the simulators offer a good level of usefulness and perceptibility by providing a useful and easy-to-understand visual display. However, for the completeness criterion, we see that NEURON and Neuronify do not provide a visualization with the necessary components to collect and extract useful information. The reason for that can be the old techniques provided by one of the oldest simulators, which is NEURON. For Neuronify, it is the fact that Neuronify is a simple tool for beginners to discover this kind of neural network, and it is

not made for analysis purposes. For truthfulness and intuitiveness, we can see that all the simulators got straightforward visualization techniques used and known by almost everyone. Moreover, nearly all simulators had a good score for aesthetics, except NEURON, which lacks the artistic part. Finally, for the interactivity, the basic requirements, like zooming and moving, are included in most of the simulators except N2S3 and NEST, since the two simulators do not provide visualization for the analysis purpose and focus more on the simulation itself.

All the presented simulators do very well with small networks, but not all can perform well with larger ones. As a result, the visualization techniques used for simple networks become less valuable and compatible for large-scale analysis. The simulation of a large network can take a massive part of the machine resources, which affects the visual display interactivity. It may be better to separate the simulation part from the visualization one to avoid this issue. Hence, We see the need for interactive visualization tools to analyze spiking neural networks, tools that can offer visualizations for analysis purposes, and the ability to analyze the network activity over time which is crucial for SNNs and hard to provide using simulators only.

3.3 Conclusion

Analyzing spiking neural networks can lead to a better understanding of the different phenomena. However, simulation of spiking neural networks usually requires considerable resources and time. As a result, the need for better ways to run the simulations on more suitable hardware appeared. Neuromorphic hardware like SpiNNaker, Loihi, or FPGA-based boards give more possibilities and consume less energy. However, the visualization provided by the tools used with the neuromorphic hardware for simulation has the same issues. Therefore, we need to separate the simulation process from the analysis since it requires considerable resources to do both in the same tool.

In this chapter, we presented a couple of spiking neural network simulators and their visualization techniques. Furthermore, we presented a technical comparison between the simulators regarding the used technology, large-scale support, and other features. We saw that Python is dominant as a programming language. For the visualization comparison, the selected criteria are inspired by the work of Freitas et al. [Freitas et al., 2002], and Stephen Few [Few, 2017] and another measure we

added which is the level of interactivity in the visual display. Finally, we emphasized separating the simulation process from the visualization one because both require considerable resources, especially when dealing with large networks. Moreover, we use the visualizations offered by the simulators for monitoring and not analysis. As a result, there is a need for dedicated tools to provide visual analysis of the activity in SNNs, tools that can support large-scale analysis and provide a degree of interactivity and support for over time analysis. This topic is the subject of the next chapter in this manuscript, where we present VS2N: a dynamic interactive tool for the analysis of spiking neural networks.

Chapter 4

VS2N: Interactive Visualization and Analysis tool for SNN

Bio-inspired technology has attracted attention in recent years due to its massive parallelism and low power consumption, making it suitable for energy-constrained applications. In addition, this technology provides neuromorphic computing by using Spiking neural networks (SNNs). Therefore, it is considered one of the promising alternatives to the Von Neumann architecture for "more-than-Moore" computing.

In 2020, every human created at least 1.7 MB of data each second, which means people generated a total of 2.5 quintillion bytes per day[1]. With this continued increase in data, it is becoming more of a challenge to manage and explore it, especially with data-driven models like neural networks. In academia, managing big data is an active research field, such as proposing algorithms to manage big data [Wang et al., 2017], frameworks to analyze big data [Gupta et al., 2017], or applications involving media indexing, classification, or retrieval. Moreover, the use of neural networks in different fields has provided significant progress, thanks to the different applications, like stock market prediction [Moghar and Hamiche, 2020], cancer detection [Taher and Sammouda, 2011], water quality prediction [Liu et al., 2019], or tasks like image classification [Falez et al., 2019] and Content-based image retrieval [Sabahi et al., 2016, Sezavar et al., 2019]. Spiking neural networks can be more suitable for many tasks than classic neural networks because SNNs consider time during activity and process natural signals while being robust to noise. In [Yang et al., 2017], the author used Spiking Cortical Model for content-based retrieval, which produces better performance due to noise robustness and geometry invariance that it provides for features extraction and texture retrieval of images. Another work [Wu et al., 2018] presented an SNN framework for sound classification, which has proven to perform robust sound recognition tasks and achieves promising performance.

[1]Bulao, J. (2021 May 7). How much data is created every day in 2021? TechJury.

Due to the asynchronous nature of spiking neural networks, it is crucial to understand how the network is evolving during the training phase to learn more about the network and easily tune the network parameters to achieve better results. Moreover, a better understanding of the spiking neural networks behavior helps close the gap between SNNs and the traditional neural networks. In the SNN domain, several simulators propose basic visualization of the network activity, as presented earlier. Nevertheless, those visual displays are not suitable for analysis purposes, and analyzing big data requires a lot of resources. Furthermore, this big data makes it challenging to do the visual analysis manually without a dedicated tool to process the data. Therefore, we present VS2N in this chapter, a web-based tool for post-mortem interactive dynamic visualization and analysis of spiking neural networks. In addition, VS2N provides the possibility to follow the network learning process, move back and forth in time, and different modules for analytic purposes.

The rest of this chapter is organized as follows: Section 4.1 represents related works concerning visualization and neural networks. Section 4.2 presents analysis requirements in the case of spiking neural networks. Section 4.3 presents technical aspects of VS2N and the different components. In Section 4.5, we describe use-cases to showcase the advantage of VS2N. Section 4.6 gives a comparison between VS2N and similar tools. Finally, we review the limitations of VS2N and the perspectives in Section 4.7.

4.1 Related work

With the growing interest in neural networks from different fields, an increasing need to better understand and intercept causality in this type of network has appeared, mainly when used in critical areas. However, neural networks are broadly considered a black box since it is often hard to interpret some decisions, and one of the ways to overcome this issue is to use visual analytics. Noh et al. [Noh et al., 2015] presented one of the earliest works in classic neural networks that used visualization to understand the network behavior better. The author reconstructed the features learned by the network from the last layer to the input layer to understand how the network is reacting to input data (Figure 4.1).

Over the years, many tools were made to answer a specific question or better understand the behavior of convolutional neural networks (CNNs), especially when

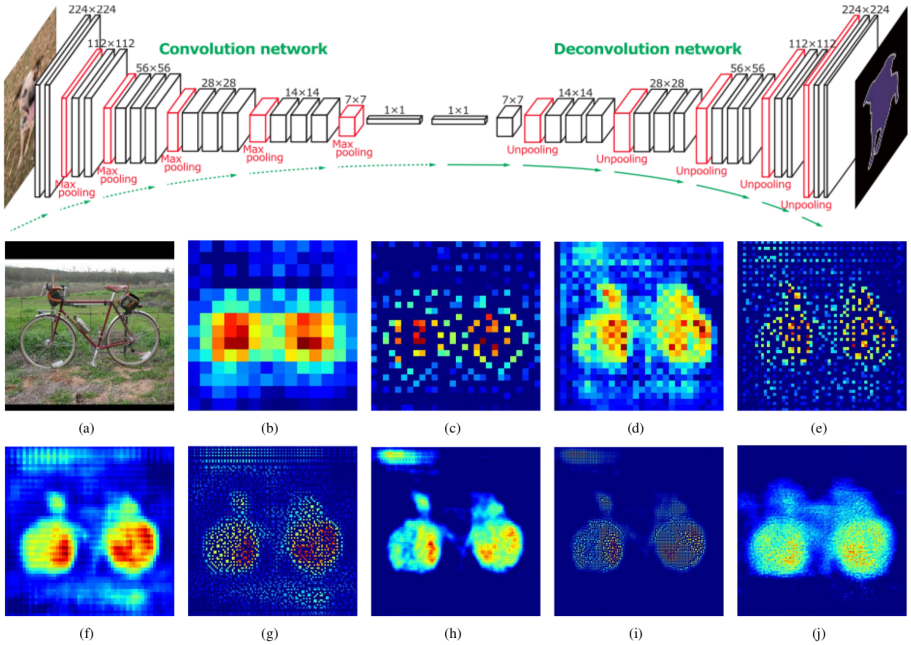

Figure 4.1: Deconvolutional network

dealing with deep learning and deep neural networks. For example, in [Krause et al., 2014], the authors introduce a tool for interactive features selection to understand better how predictive features are ranked across feature selection algorithms. This tool leads to essential insights when tested on a case study from the clinical research field. Moreover, tools presented for deep neural networks (DNNs) [Kahng et al., 2018, Zurowietz and Nattkemper, 2020] help the user to better explore complex DNNs, by providing visualization approaches to convolutional neural network layers. Besides, a similarity display can reveal how each layer perceives the input in a deep neural network. Finally, in [Strobelt et al., 2018], a visual analysis tool is presented for recurrent neural networks (RNNs) to understand the hidden state dynamics, which leads to a better network.

For spiking neural networks (SNNs), analyzing large networks is challenging due to the asynchronous nature and spikes for communication. In [Marks, 2017], the authors present a framework for immersive and intuitive 3D visualization of the network in virtual reality. This framework improves the user's abilities to investigate and examine SNNs. Another work [Kasiński et al., 2009] presents a 3D interactive visualization tool of SNN by visualizing individual neurons and their connections. This work focuses on the clarity of the network exploration and the implementation

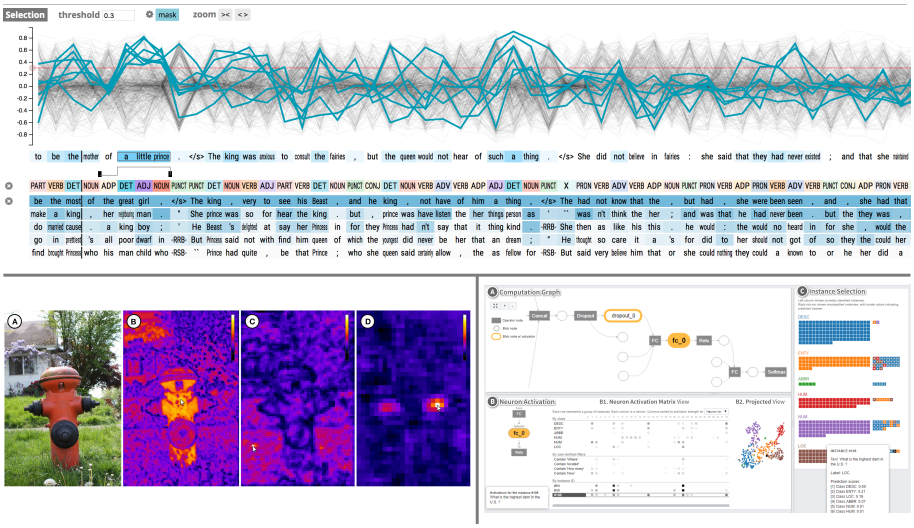

Figure 4.2: Example of CNN visual display tools

issues related to 3D network representation. Finally, another multipurpose tool is presented in [Senk et al., 2018] to visualize the network layers in 2D and 3D, using coordinated multiple views for massively parallel neurophysiological data.

If we analyze the presented tools, we can see that they have some points in common, such as web-based tools, targeting a specific problem, and limited to a defined simulator. We can summarize the novelty of VS2N in four features: 1. Modular nature: the visualizations are grouped into modules. Each module may target a specific question or problem. Anyone can add new modules for a particular analysis. 2. Simulator-independent: we can use any simulator as long as the collected data follows specific schemas, which VS2N can understand. 3. Scalability: backed by the combination of Apache Spark and MongoDB for data processing, we can deploy VS2N on multi-nodes or clusters for more solid performance. 4. Dynamic analytics: VS2N provides the possibility to walk in time with the evolution of the network during activity, which is not possible using the existing tools for data analysis. This feature is significant when the network evolution is done over hours of activity, which is the case in spiking neural networks.

In MongoDB, we put data in collections, and each collection contains documents in JSON (which are similar to rows in a relational database). A *document schema* is a JSON object that contains information about the shape, fields, and type of data stored in that document. In VS2N, we use predefined schemas to read the stored

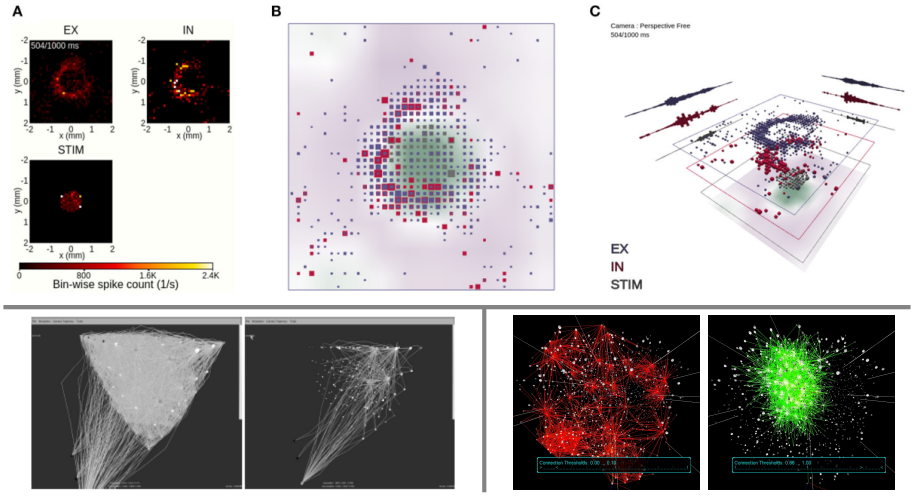

Figure 4.3: Example of SNN visual display tools

data, and simulators need to consider it to use VS2N[2].

4.2 Requirements analysis

In order to analyze spiking neural networks, we can break the network components into three entities:

4.2.1 Input data

Neural networks are data-driven models, meaning that the quality of the input data plays a significant role in network performance. Preparing and cleaning the input data is a time-consuming operation. However, it can help observe any correlations and reduce biases. The data analysis involves studying and observing the input data during the network activity and the network reaction to it. This analysis can differ from one type of input data to another since datasets come in a different format (see Figure 4.5).

[2]https://gitlab.univ-lille.fr/bioinsp/VS2N/-/wikis

```
{
  "properties" : {
    "_id" : { "bsonType": "objectId" }, // id
    "n"   : { "bsonType": "string" },   // simulation name
    "L:N" : {                           // layer name : neurons
              "Layer1_name":  { "bsonType": "int" },
              "Layer2_name":  { "bsonType": "int" },
              ...
            },
    "T" : { "bsonType": "string" },     // simulation date
    "D" : { "bsonType": "string" }      // dataset name
  }
}
```

General information schema

```
{
  "_id" : ObjectId("603834c376e245cd78319032"),
  "n" : "My network",
  "L:N" : { "Input": 784, "layer1": 100 },
  "T" : "26-02-2021-00:37:39",
  "D" : "Mnist"
}
```

Example

Figure 4.4: VS2N MongoDB schema for one collection

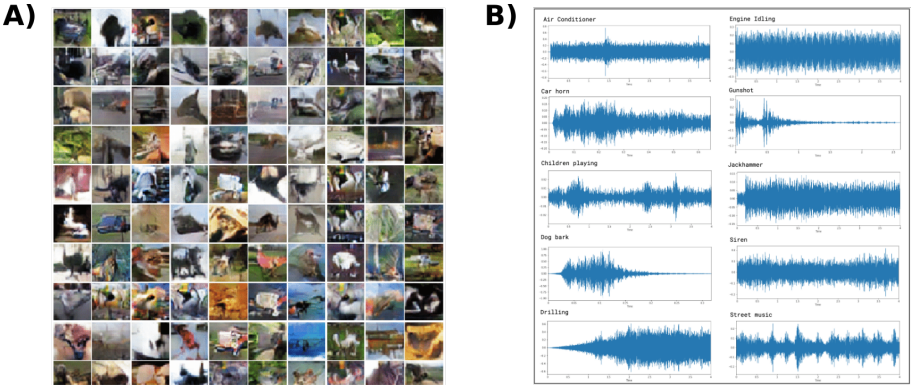

Figure 4.5: Input data examples: A. CIFAR-10 B. Urbansound8K

4.2.2 Neurons

Neurons are the main ingredients of the network. The neuron behavior depends on the type of neuron and can be affected by the network activity, the information coding, and the network architecture. Neuron analysis affects the membrane activity, the spike frequency, and the neuron behavior to an input. By analyzing all neurons, we can monitor the network performance and identify any possible improvements.

4.2.3 Synapses

Synapses keep the neurons attached and help spread activity across the network. However, the number of synapses in a network is more than the number of neurons, making it challenging to analyze their activity. We can learn more about the neuron's response to specific input and the learned features, which is more challenging in multilayer networks by analyzing synapses. The analysis may help reduce unnecessary synapses while preserving good performance, which is helpful for better hardware implementation.

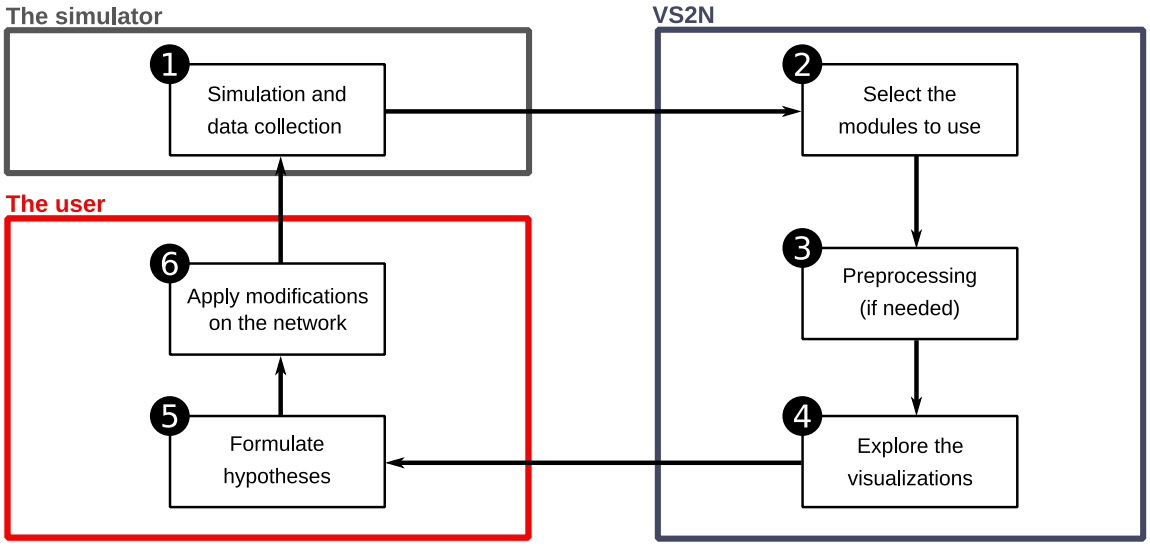

Figure 4.6: The visual analysis workflow

As described in Figure 4.6, to use VS2N, the user (1) start by collecting the data from the simulation. In our case, we used the N2S3 simulator [Boulet et al., 2017]

and Nengo [Bekolay et al., 2014]. We can use any simulator as long as the collected data follows the schemas from VS2N. In (2), select the simulation and the modules on VS2N. Then in (3), launch the pre-processing by VS2N using Apache Spark. (4) start exploring the analysis using the web interface and detect any patterns or phenomena. (5) formulate hypotheses based on the observations. Finally, (6) apply adjustments to the network and start the simulation and data collection again.

4.3 VS2N components

VS2N is a web-based tool based on Flask [Miguel Grinberg, 2018], a micro web framework written in Python. The backend is composed of two main parts: MongoDB[1], for saving data obtained from the simulation, and Apache Spark[2], for any pre-processing on the data. Due to the nature of MongoDB and Apache Spark. This organization makes it plausible to scale in terms of computation power and deploy on the nodes of a distributed cluster (see Figure 4.7). In addition, VS2N uses Dash library[3] to create web interfaces and interactive visual displays using Python.

Since Python is the unified language used, we can take advantage of the different machine learning libraries in Python (Scikit-learn [Pedregosa et al., 2011], PyTorch [Paszke et al., 2019], TensorFlow [Abadi et al., 2015], etc.) and combine it with the visual displays for a better experience.

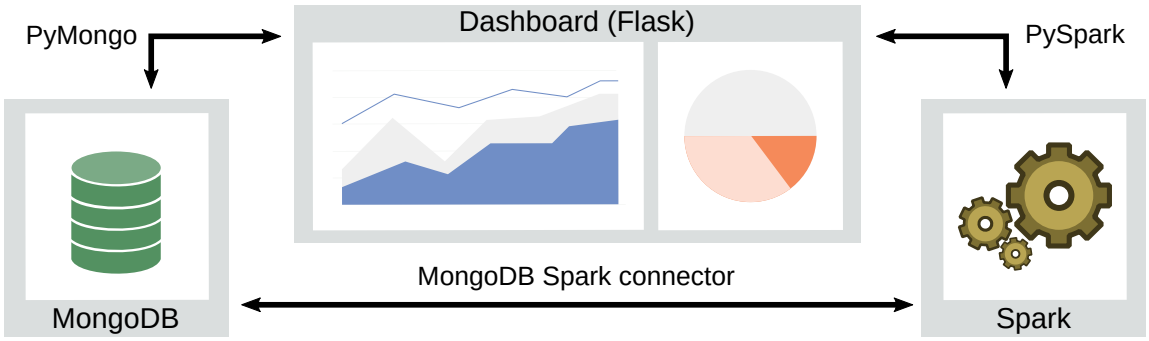

Figure 4.7: VS2N components

[1] www.mongodb.com
[2] www.spark.apache.org
[3] www.plotly.com/dash

4.4 Visualization methodology

In this section, we present the different analysis modules in VS2N, the used visualizations, and the purpose of each module. The network used during the presentation of the modules is a single dense layer network and the images dataset as input.

4.4.1 General analytics module

This module summarizes the network performance and the general information, like the number of neurons and layers, network accuracy, and the used dataset. This module is the first one used with every analysis. Therefore, it does not require any preprocessing, and we can use it even while the simulation is still running. It is made of three main parts:

1. **General information:** represents a summary of the network accuracy, used topology, and dataset. We extract this information from the info collection. This collection is filled at the start of the simulation, except for the accuracy. Using this part, we can learn about the network structure before analysis using the information about the network components, which will affect how we approach the visualizations during the visual examination and distinguish between the different network traces we collected using the "simulation date."

Information			
Neurons	100	**Layers**	1
Input	784	**Dataset**	MNIST
Simulation Date	10-06-2021 15:43:41	**Accuracy**	84.35 %

Figure 4.8: General information

2. **Network activity:** a general visual display of the network activity, such as spikes, neurons potential, synapses update, and loss update during training. This part is helpful to observe any patterns in the activity, the learning evolution of the network, and any correlation between the different graphs. We

can select the information to show the type of graph and move the focus on a specific region of the representation using the selector at the bottom.

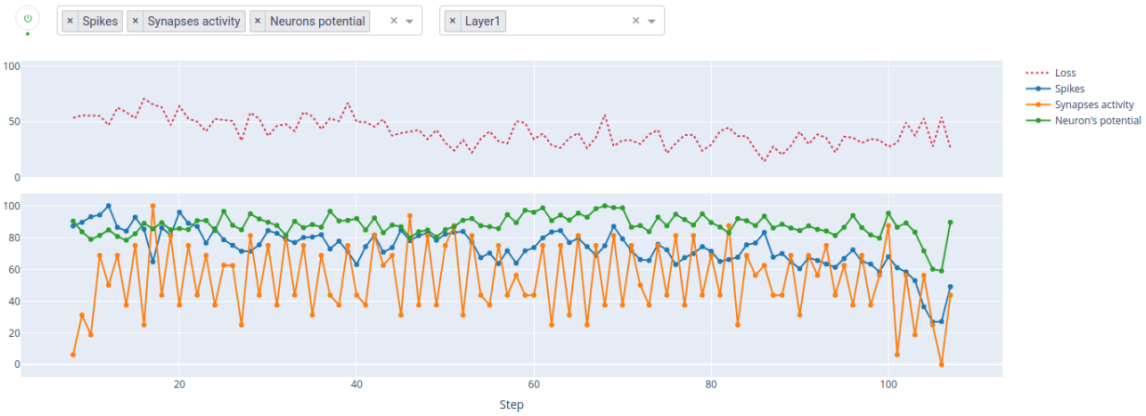

Figure 4.9: Network activity

3. **Dataset overview:** represents information on the actual input of the network at that period. This visual display illustrates the number of each input (grouped by labels if it is a labeled dataset, otherwise this visualization is not shown), and it does not depend on the learning type (supervised or unsupervised) but only on the dataset (labeled or not). This treemap representation helps the user observe the distribution of the used dataset and all the classes over time.

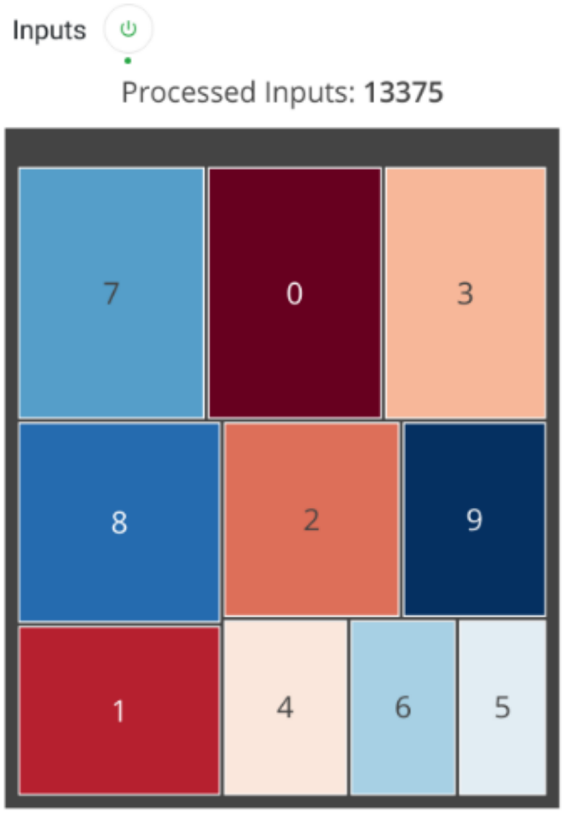

Figure 4.10: Dataset overview

4.4.2 Neuron analytics module

This module contains a group of interactive visualizations to observe the activity of each neuron separately or grouped. The user can analyze the behavior of the selected neuron with the guidance of spikes and potential activity while considering the input data that led the neuron to spike. However, to use this module, we need to wait until the end of the recorded network activity (learning or inference) since it requires some preprocessing using Apache Spark, which we cannot do before collecting all the data. The different components of this module are the following:

1. **Layer and neuron selector:** this selector controls what information to display, which is helpful if we want to follow or compare the evolution of only

selected neurons. Besides, by filtering what information to show, we reduce the required computation resources and focus on the goal of the analysis. When the user selects a neuron, more visualizations regarding this element are displayed (4, 5, and 6).

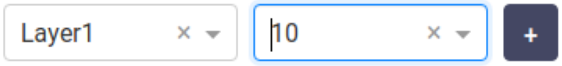

Figure 4.11: Layer and neuron selector

2. **3D spike frequency:** this component provides a 3D illustration of the spike frequency per neuron in the output layer (neuron ID on the X-axis, spike frequency on the Y-axis, and Z-axis for the respective class). In the case of frequency-based coding, this visualization provides the user insight into the neuron's spike frequency compared to other neurons detecting the same class and the other classes. Therefore, how neurons from one class are distributed over the 3D space indicates the network performance and possible class confusion due to similarities. However, this representation is only for labeled data, and VS2N will discard it if we use non-labeled data.

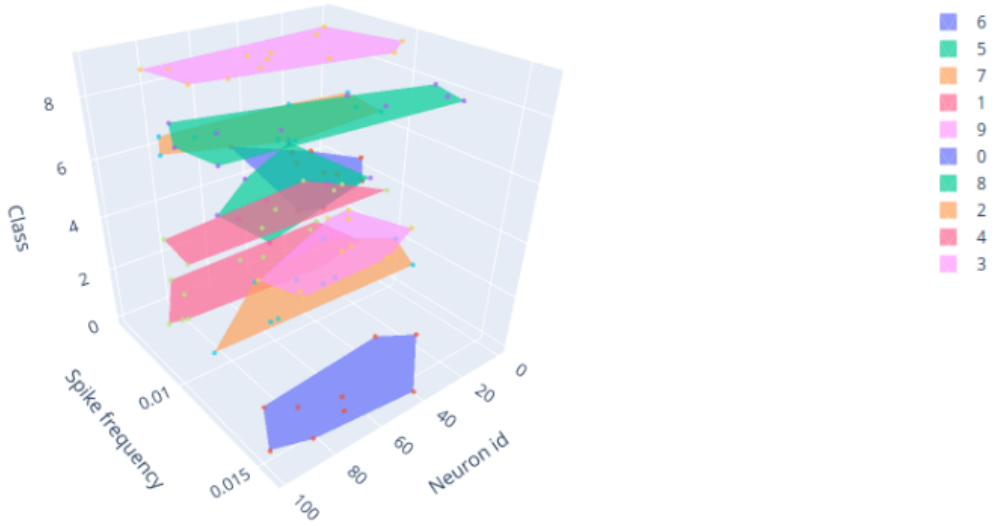

Figure 4.12: 3D spike frequency

3. **Neuron spikes per class:** this representation is visible only when we select a neuron from a class in the 3D spike frequency component. This graph provides information on the number of spikes per neuron in the chosen class. It is complementary to the previous one and offers more information to better analyze the neuron activity in the same class. Using this graph, we can compare the number of spikes in one class and check the presence of any dominating neurons. Similar to the last component (the 3D spike frequency component), this representation is only visible if the network uses labeled data.

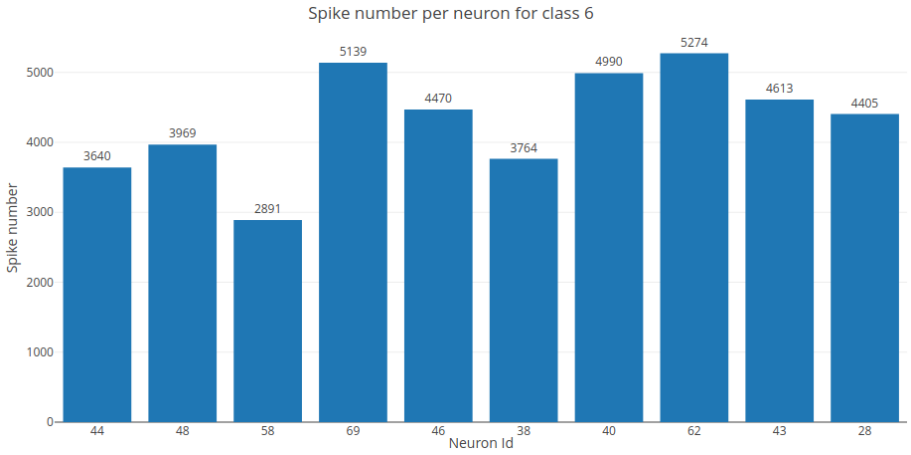

Figure 4.13: Neuron spikes per class

In this module, VS2N adds three new graphs upon neuron selection by the user. Those visualizations are added for each neuron selected by the user. The three representations are:

4. **Neuron spikes activity:** this represents the neuron's spiking activity during the recorded period. The number of spikes is accumulated per step, defined by the user (1s by default). This representation helps the user confirm any silent neuron's existence once we identify it using the previous visualizations in this module.

5. **Neuron potential activity:** this represents the membrane potential activity of the neuron. VS2N displays the average of the membrane potential per step. Using this component, we observe the internal neuron phenomena such as voltage leakages (in the case of LIF neurons) or refractory periods after a

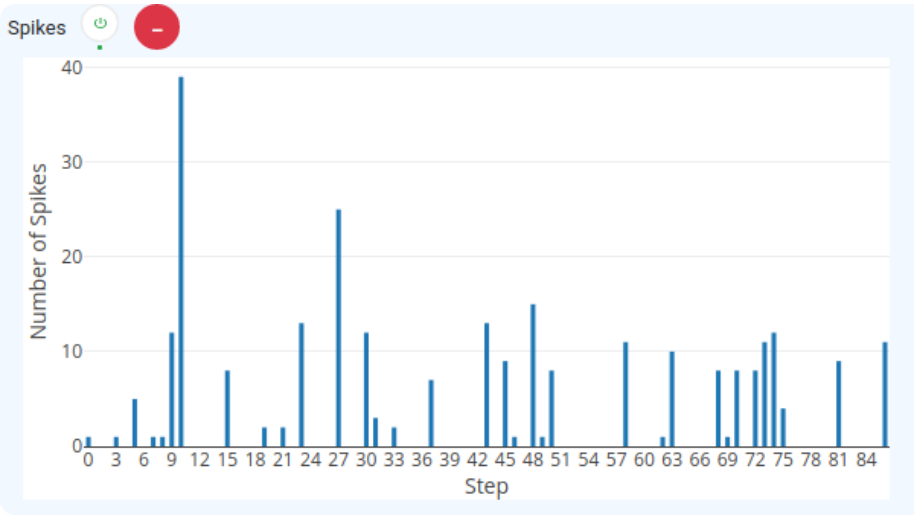

Figure 4.14: Neuron spikes activity

spike. Moreover, we can test and validate any newly implemented formulas of a neuron.

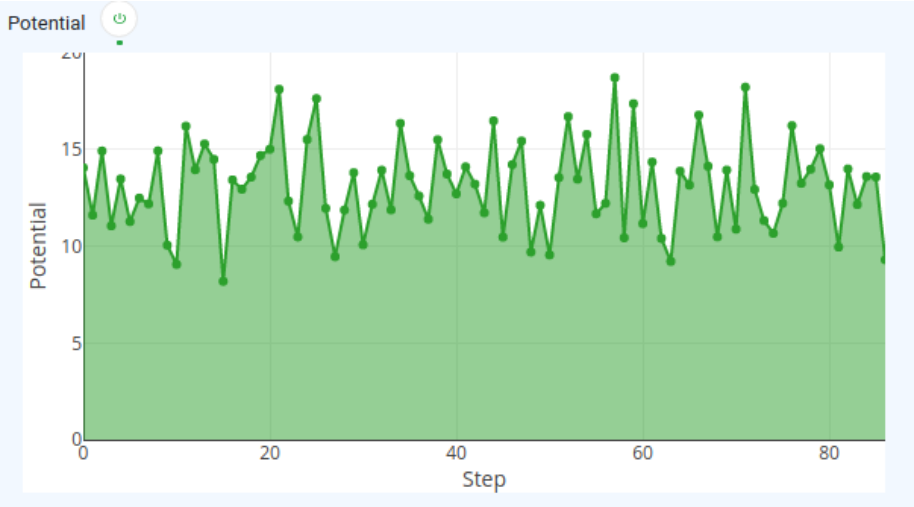

Figure 4.15: Neuron potential activity

6. **Neuron class activity:** represents the input class that made the selected neuron spike. This visualization helps observe and compare the neuron activity per class during the analysis period and how it changes over time. Moreover,

we can see how some neurons react to a couple of classes simultaneously, classes with similar features. However, since this representation also depends on the input data, which needs to be labeled, it is visible only if the dataset is labeled. Otherwise, the space on the screen is shared between the two previous graphs.

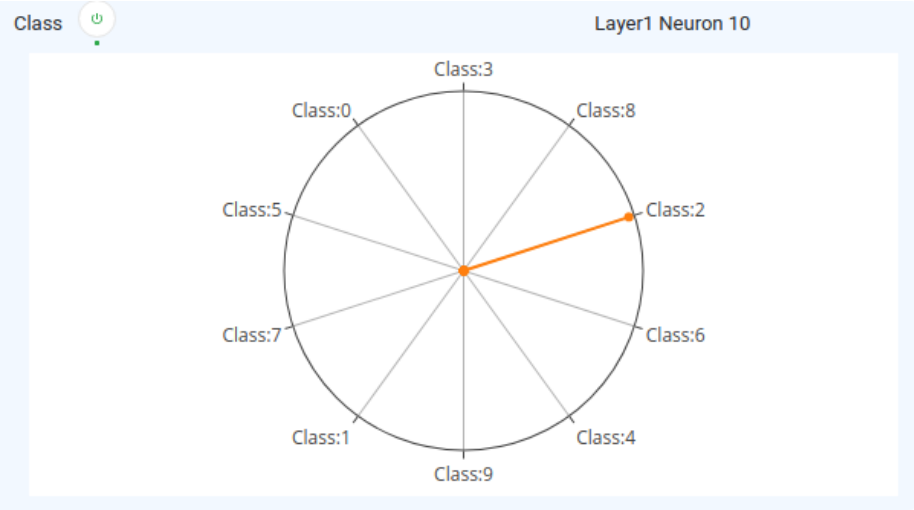

Figure 4.16: Neuron class activity

4.4.3 Synapse analytics module

This module includes a group of interactive visualizations to watch the synaptic activity of each neuron. The synaptic activity in a neural network represents the learning process. Therefore, by analyzing it, we can learn more about this phenomenon. The visualizations in this module provide insight into how the synapses react to input data. Any modification on the detected class by the neuron can be seen by a slight variation in the mean synapse weights and the heatmap. It is also helpful to follow the activity on the synapses, especially when applying pruning, where we can see the effect on the network. Unfortunately, this module requires preprocessing after the end of the simulation, which prevents us from using it while the simulation is still running. This module is composed by:

1. **Layer and neuron selector:** when the user selects a neuron, more visualizations regarding this element are displayed (3 and 4). We can choose the

layer name and the neuron ID, and since we are dealing with synapses, we can specify the dimensions of the heatmap shown to see the patterns correctly.

Figure 4.17: Layer and neuron selector

2. **Layer heatmap:** represents an overview of the selected layer heatmap at the end of the training. This visualization is static and requires preprocessing at the end of the simulation to generate. Therefore, it is not possible to use during the simulation. However, this representation helps the user get a clear overview of the features learned by the selected layer, which is helpful to evaluate the learning process. Moreover, for a multilayer network, reconstruction is needed to get the proper heatmap representation.

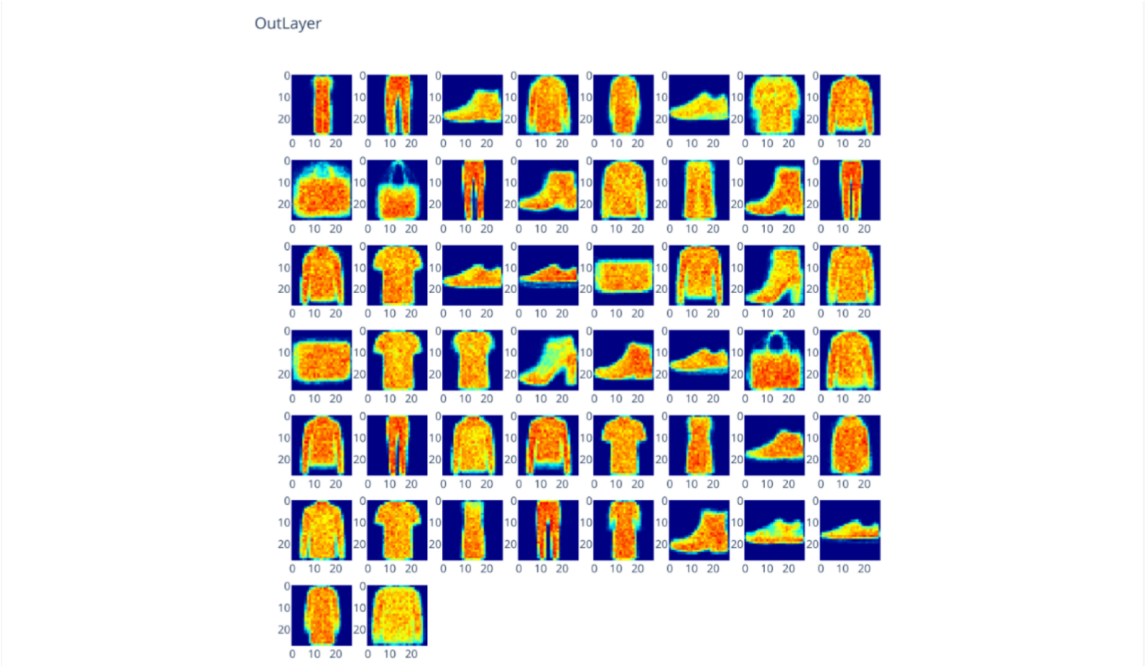

Figure 4.18: Layer heatmap

Once we select one or more neurons, VS2N adds two other components, representing information about the activity of the selected neuron. Those components are stacked

dynamically under the layer heatmap representation, and we can remove them easily. Those representations are:

3. **Synapses weight activity:** this representation combines a vertical heatmap and a graph. The graph illustrates the mean value of the neuron weights, and the vertical heatmap represents the weight distribution throughout possible values. Using this visualization, the user can observe how the distribution of the weights evolves compared to other neurons from the same or different classes.

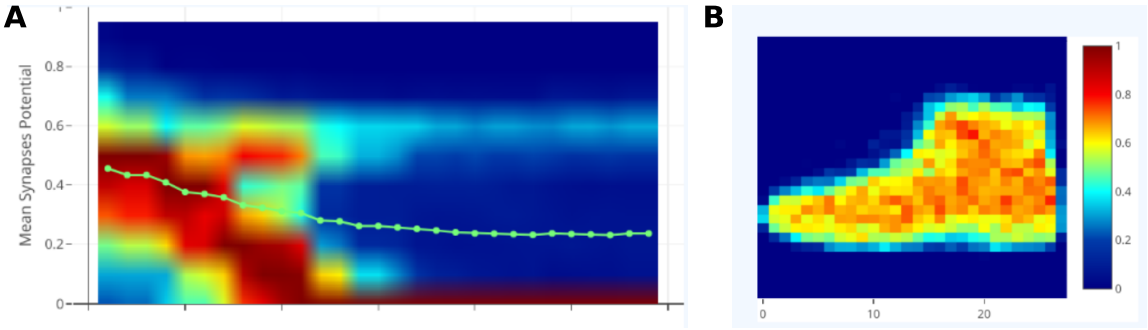

Figure 4.19: A) Synapses weight activity and B) Neuron heatmap

4. **Neuron heatmap:** this component is complementary to the earlier visualization. It represents the synapses heatmap of the selected neuron. The main distinction between this representation and (2) is that this representation gets updated over time while the layer heatmap represents the final values of synaptic weights. This heatmap allows the user to detect any change in what the neuron learns during the simulation.

4.5 VS2N use-cases

This section presents two use-cases where we use VS2N to observe an activity, answer a question, or validate a hypothesis. For the simulation, we use the N2S3 simulator, an open-source, scalable spiking neuromorphic hardware simulator [Boulet et al., 2017], and the Nengo simulator [Bekolay et al., 2014]. For the dataset, we use MNIST dataset [LeCun et al., 1998], which contains handwritten digits (60k for

training, 10k for testing). Finally, all the use-cases are applied on a network with a single dense layer, which uses simplified STDP [Querlioz et al., 2011] as a learning rule and LIF neurons.

4.5.1 The MNIST last 10k effect

When using the MNIST dataset, we saw an accuracy drop in the last 10k input of the training data. This drop is observed on small to medium networks (less than 1000 neurons). However, it does not appear on large networks. Furthermore, from this observation, we assume that the last 10k contains different images, which affects the network learning process. As a result, by training the same network using MNIST inverted, we see that this drop disappears from the last 10k and appears at the start of the training, as seen in Figure 4.20.

We use Nengo for the simulation in this use-case, which is not the simulator used to obtain the results shown in Figure 4.20.

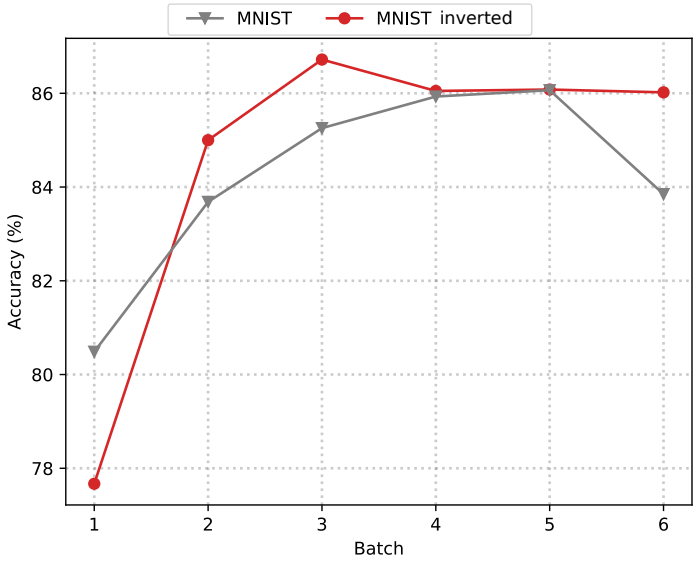

Figure 4.20: Network accuracy using 900 neurons with MNIST (1 batch = 10k)

We use VS2N to observe the network activity during training and the last 10k using the general analytics module. The goal is to validate the existence of this phenomenon by identifying any change in activity around the period when we introduced the 10k input.

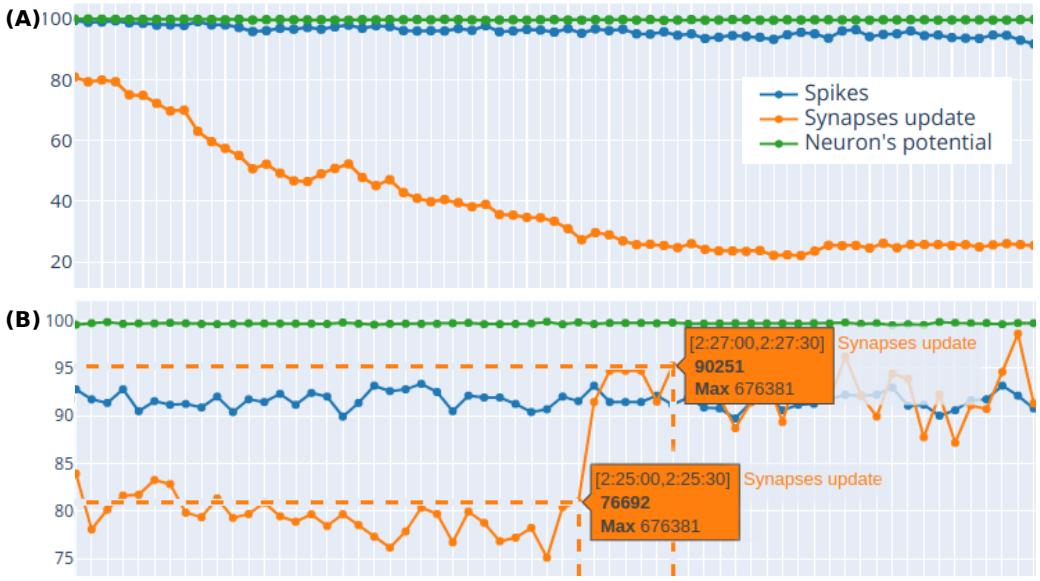

Figure 4.21: Network activity: (A) the first 15k input, (B) the last 10k

In Figure 4.21(A), we can see at the beginning of the training, the number of spikes and neurons potential activity is stable. However, synaptic weights update decreases and become stable after a couple of inputs, which is expected due to training and the nature of the MNIST dataset, which contains centered handwritten digits. Although, after 50k of input, we can see an interesting pattern at the start of the last 10k (Figure 4.21(B)), with a noticeable increase in the synaptic weights update (more than 10%) that remains until the end of the simulation. Since we are using a single dense layer in this network, this increase in synaptic weights update means that the patterns learned by the neurons are changing, which will affect the network performance, as reported in Figure 4.20.

4.5.2 Network compression

One technique to compress the network is by pruning one or more elements in the network. This element can be a neuron, a synapse, or kernels in the case of a convolutional network. In the case of synapses, we usually use a defined threshold as a criterion of pruning. If the synapse weight value is below the threshold, we consider the synapse as not critical, and we will remove it. The threshold selection is usually based on experimentation, and it is interesting to watch the effect of the

prune operation on the network.

For this use case, we use N2S3 to simulate a single-layer network. After training using half of the MNIST dataset, we apply the prune operation on synapses (threshold=0.2). Then, using VS2N, we observe the general activity of the network once pruned and the impact of this operation on two randomly chosen neurons.

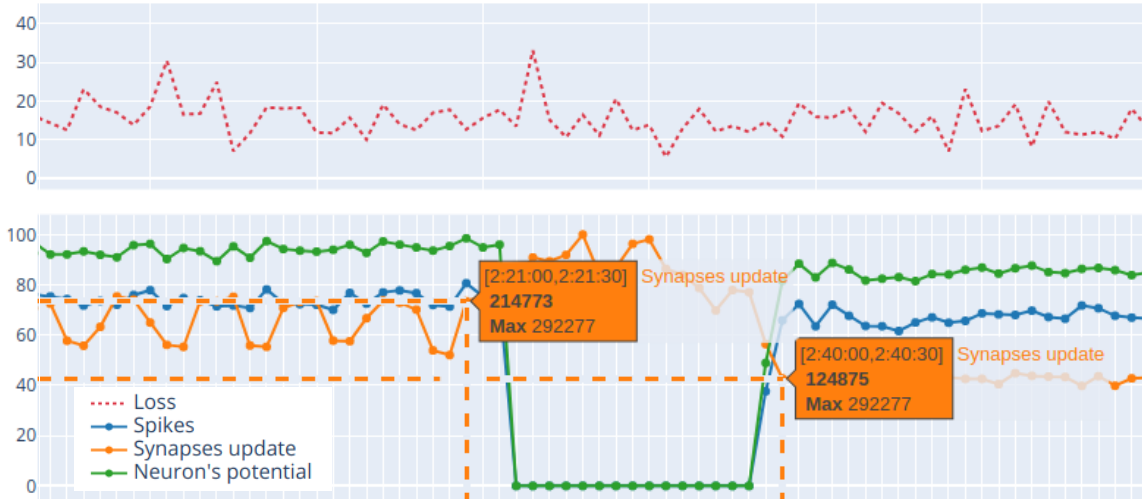

Figure 4.22: Network activity after 30k input (during the prune operation)

We can see in Figure 4.22 the activity of the network before and following the prune operation. The spikes and neuron's potential activity values slightly decreased after removing more than 50% of the synapses. We expect to see a slight decrease in activity since we removed only the weak synapses due to the MNIST dataset's nature of having centered handwritten digits. We can see a drop in values and a more regular graph after pruning for the activity of the synaptic weights. Even though the removed synapses are weak, they are still updated during training, as shown in the graph. We can see a slight increase just after pruning for the loss graph, but it returns to normal again. Finally, we can see in Figure 4.22 that after the prune operation, there was a period of silence in the network. This unexpected pattern is due to how the pruning process is implemented in the simulator. In N2S3, we had this issue with simulation time when applying the prune operation. In Figure 4.23, we can see that by using small networks, the simulation time decreases when prune is used, which is what we expect. However, when the network size starts growing, the simulation becomes slower than the none-pruned one, which is the opposite of what we expect compressing a network since we will have fewer components and less

computation, which should reduce the simulation time. Based on our observation using VS2N, we can see that the issue behind the longer simulation time is, in fact, the prune function in the simulator, which blocks the activity in the network and causes the simulation to take longer.

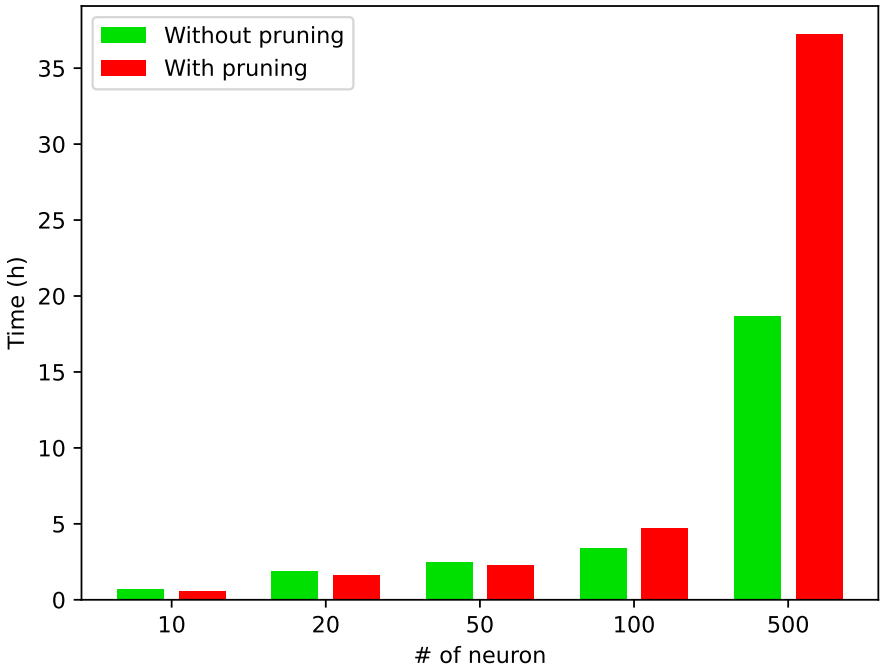

Figure 4.23: Simulation time with and without pruning

In Figure 4.24, we can see the synaptic activity of two neurons using the synapse analytics module. After 15k of input (Figure 4.24(A)), we can see the average value of synapse weights is close for both neurons. But, the distribution of the synaptic weights over time (left) is different because the two neurons are learning different classes, as seen in the heatmap (right). In Figure 4.24(B), we can see the outcome of applying the prune operation on the two visual displays. The mean synaptic weights increased since we removed weak synapses, and we can see an update in the distribution of the synaptic weights (left). The heatmap (right) shows that the applied threshold (0.2) does not influence the learned class.

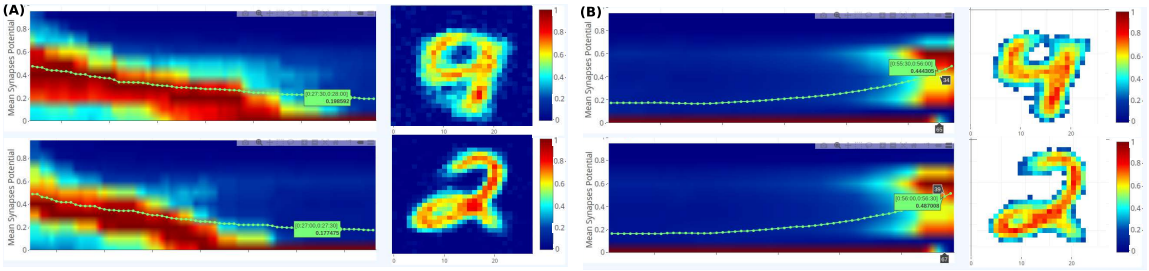

Figure 4.24: Synapses activity of two neurons: (A) the first 15k input, (B) after 30k input

4.6 VS2N compared to similar tools

Since other tools for visual analysis of neural networks exist, we grouped some of them to compare with VS2N in terms of some criteria, such as Scalability, Interactivity, and other criteria related to visualization. It is worth mentioning that not all those tools support the visualization of SNNs; some are made for convolutional neural networks.

Kahng et al. presented ActiVis [Kahng et al., 2018], which provides a visual exploration of industry-scale deep neural network models by integrating multiple coordinated views, such as an overview of the model computational architecture graph and a neural activation view for model discovery and comparison. Zurowietz et al. presented in 2020 IFeaLiD [Zurowietz and Nattkemper, 2020], an interactive visualization for feature localization in deep neural networks. The tool interprets neural network layers as multivariate feature maps and visualizes the similarity between the feature vectors of individual pixels of an input image in a heat map display. Using this tool can reveal how different network layers perceive the input image and how the perception of one particular image region compares to the perception of the remaining image. TensorBoard [Abadi et al., 2015] is another tool that comes with TensorFlow. It is a web-based tool for providing the measurements and visualizations needed during the machine learning workflow. Using TensorBoard, we can track experiment metrics like loss and accuracy, visualize the model graph, project embeddings to a lower-dimensional space. It is a good way to cross-validate the hyperparameters.

For the tools with SNN support, we have NeuVis [Marks, 2017], which provides immersive visualization of 3-dimensional spiking neural networks in virtual reality.

Furthermore, Kasiński et al. presented SNN3DViewer [Kasiński et al., 2009], a 3D visualization tool for spiking neural network analysis. This tool is dedicated to supporting the study of dynamical processes in large spiking neural networks. Finally, we have VIOLA [Senk et al., 2018], a multipurpose and web-based visualization tool for neuronal-network simulation output. It combines and adapts modern interactive visualization paradigms, such as coordinated multiple views, for parallel neurophysiological data.

Features	ActiVis	IFeaLiD	TensorBoard	NeuVis	SNN3DViewer	VIOLA	**VS2N**
Scalability	X		X	X			**X**
Interactivity	X	X	X		X	X	**X**
SNN support				X	X	X	**X**
Python-based	X		X			X	**X**
3D visualization				X	X	X	
Real-time analysis		X			X		**X**
Web-based interface	X	X	X			X	**X**
Simulator-independent					X		**X**

Table 4.1: VS2N features compared to similar tools

In Table 4.1, we can see that the most common feature between those tools is the *Interactivity*, which is what we expect since we are talking about visualization tools, and the interactivity is a must for a better analysis experience. Next, we have *Web-based interface*, as we can see a lot of those tools are web-based, which allows great portability and fewer deployment issues. Then, we have *Scalability*, *Python-based*, and *SNN support*, as we can observe in Table 4.1, more than half the tools are Python-based, which reflects the impact of Python in the field of machine learning or visualization. Besides, most of the tools are scalable and support large neural networks and spiking neural networks. For the *3D visualization* and *Real-time analysis*, we can see that few tools provide it, which may reflect the difficulty of having 3D visual displays when dealing with large scale networks, and the same thing for the real-time analysis. Finally, we can see that *Simulator-independent* feature is not available in the majority of the tools since they usually target a specific question or a type of network and support data generated from a particular source. The features used in this comparison can change from one application or field to another, affecting the evaluation of those tools.

4.7 Conclusion

The ability to interpret the behavior of a neural network plays a crucial role in better exploring this technology, especially in critical applications where we need to support each decision. Another reason is the issues related to biases and discrimination reported in some neural networks applications, which we can reduce if we can detect them earlier.

This chapter presents VS2N, a simulator-independent tool for dynamic visual analysis for SNNs.VS2N is a result of the lack of tools for SNN analysis since simulators do not provide visualizations for analysis purposes. VS2N offers the ability to analyze the network activity interactively. Using VS2N, we can add new modules, which contain a group of visualizations for one question or a component in the network. Then, we showed two use-cases with data collected from two different simulators to validate the existence of a phenomenon in one use case and observe the prune operation in the other one with the help of VS2N. Finally, we compared similar visualization tools based on selected features. This version of VS2N mainly supports the analysis of shallow networks. Furthermore, due to the complexity of the multilayer networks, we need to add modules that suit this type of network, like features reconstruction and 3D visualization of the activity of the hidden layer. Moreover, the pre-processing step at the first time (using Apache Spark) takes time due to the massive amount of data, which we can reduce by deploying VS2N on the nodes of a distributed cluster. We will address those limitations in the future. Finally, it is worth stating that VS2N can be used to analyze the activity during the simulation without waiting for the simulation to finish, which can be helpful when simulating large networks that can take considerable time. However, only the general analytics module will be available in this case since the other modules require pre-processing, which is impossible to do while simulating for now. Using VS2N, we proposed a novel approach for compressing spiking neural networks dynamically by observing the network evolution during training. We discuss this contribution in the next chapter of the manuscript.

Chapter 5

Progressive Compression and Weight Reinforcement for SNN

In the past, researchers presented different hardware implementations of the neuromorphic architectures using various hardware components [Merolla et al., 2011, Shahsavari and Boulet, 2017]. Yet, the commonly observed characteristic when working with SNNs is the big network size. Using large networks, we get numerous neurons and synapses with the rising complexity of the hardware implementation, which makes it also hard to analyze. For example, using the MNIST dataset [LeCun et al., 1998], Diehl and Cook [Diehl and Cook, 2015] increased the number of neurons by more than 60 times to enhance the performance by 12 %. Lee et al. [Lee et al., 2016] used two hidden layers of 800 neurons each to get an average accuracy of 98.6 % on MNIST. In the work of Diehl et al. [Diehl et al., 2015], the authors implemented a network with two hidden layers of 1200 neurons each, and this network gave an average accuracy of 98.64 % using the same dataset. From the given examples, we can see that we need larger networks for better performance, which introduces the following challenges :

- It is harder to analyze larger networks compared to small ones.
- Having large networks means more extended simulations with more computational resources, affecting energy efficiency.
- It is harder to implement larger networks in hardware using technologies like the memristive crossbars [Strukov et al., 2008, Merolla et al., 2011], due to the design challenges known when deploying a large network [Liu et al., 2015].

In SNN, spikes are the mean of communication. They are transmitted by the synapses, representing the communication channel between the neurons. The number of synapses in a network scales with the number of neurons, so having an efficient

way to use them leads to a better energy-efficient network and decreases the cost of production on hardware. Pruning is widely used in neural networks to reduce network size and complexity. The evolution of the human brain motivates this technique since synaptic connections are pruned from the early stages [Huttenlocher, 1979]. Such phenomena inspired researchers to adapt this technique to get a smaller network while preserving the same performance or with a bit of loss [Cun et al., 1990, Hassibi et al., 1994].

This chapter presents a novel approach to reduce the network size dynamically [Elbez et al., 2022]. We proposed this approach after observations were done using VS2N on the network evolution during the learning phase. The compression concerns the synapses in the network and is composed of two formulas: the first defines the threshold value to use for pruning. The second reinforces the synaptic weights to preserve the average spiking frequency. Combining these two techniques can produce trainable compressed networks that can achieve better accuracy than baseline when trained again. The rest of this chapter is organized as follows: Section 5.1 presents related work to neural network compression and finishes by comparing them to our approach. Section 5.2 contains background about the used network, neuron, and synapse models, with a presentation of the STDP learning rule. Section 5.3 illustrates the contribution in details. Finally, in Section 5.4, we present and discuss the results.

5.1 Related work

We can classify previous works on compressing Convolutional Neural Networks (CNNs) into four categories: 1- Parameter quantization [Wu et al., 2016], 2- Low-precision weights [Son et al., 2018, Zhou et al., 2017], where the weights are stored with low precision to decrease memory cost, 3- Knowledge distilling [Hinton et al., 2015, Kim et al., 2018], which consists in training a small network to reproduce the learned function of a larger network, instead of training the small network, 4- Pruning by reducing the number of synapses or neurons based on a criterion.

From the literature, we organize pruning into three types: Weights pruning [Carreira-Perpinan and Idelbayev, 2018, Han et al., 2015, Liu et al., 2018], which concerns the removal of unnecessary synapse connections in the network, based on a static threshold. Filter pruning [He et al., 2019, Huang et al., 2018, Li et al., 2017], which consists

in removing filters instead of synapses to reduce the computation time and the size of the network. For example, in Luo et al. [Luo et al., 2019], the authors used filter pruning on the VGG-16 model, which produced a compressed version with 2.66 MB in model size (instead of 528 MB) while preserving an acceptable level of accuracy. Finally, in neuron-based pruning, we execute the prune operation on neurons, which have a weak impact on the network. For example, in the work of Yu et al. [Yu et al., 2018], the authors made pruning neurons based on a neuron importance score, applied on AlexNet, and reduced computation cost by 60 % with a slight accuracy loss.

SNNs can process natural signals. They can be implemented physically via ultra-low-power based devices; for instance, on CMOS [Sourikopoulos et al., 2017] or memristors [Shahsavari and Boulet, 2017]. CMOS artificial neuron is a simple component (six transistors operating in the subthreshold mode). The energy consumption is several orders of magnitude lower than artificial neurons reported in the literature, but also two to three orders of magnitude lower than the energy consumption of a living neuron.

Concerning SNNs, we use similar pruning techniques. For example, Rathi et al. [Rathi et al., 2019] combined pruning while the network is learning with a weight quantization technique. Using a 2-layer SNN of 6400 neurons, we can achieve highly compressed networks while preserving a good performance using a static pruning threshold. We can apply this technique by deactivating the synapses that are deemed unnecessary, based on their performance using the STDP learning rule [Bi and Poo, 1998]. Then, we remove the remaining non-critical synapses from the network at the end of the learning phase. Another work by Saunders et al. [Saunders et al., 2019], based on the locally connected Spiking Neural Network (LC-SNN), using a 2-layer SNN of 900 neurons. They performed the prune operation just once at the end of the learning phase, which removed 50 % of synapses and a 90 % maintained accuracy.

Shi et al. [Shi et al., 2019] used a soft-pruning method during the training process, which prevents the unnecessary update of the network parameters. Using the MNIST dataset, this approach maintained 90 % accuracy despite a slight accuracy loss and 75 % of synapses removed. Cho et al. [Cho et al., 2019] applied a distance-based prune on CMOS SNN chip by eliminating connections between a group of neurons based on the distance between them, and this resulted in a spike traffic drop by 52 %. Finally, Chen et al. [Chen et al., 2018] presented a three-step prune, the first two concern removing neurons with low activity; the third one is pruning weak synapses. The prune operation in this work was a part of converting a CNN to an SNN, which

lowers the computational operations.

We can see that the existing works in terms of pruning focus on how to treat the pruned synapses, when to prune, or the used criteria for pruning while keeping the same static threshold, which is selected based on experiments. In contrast, our contribution deals with the threshold value selection and how to treat the maintained synapses in the network. It is worth mentioning that we can use our contribution to complement the existing prune approaches. The originality of this approach is the use of a dynamic threshold for pruning instead of a static one by using a progressive pruning function that computes a new threshold with every batch, based on the previous prune operation. As a result, the modified network can maintain the same performance during every learning batch compared to the baseline with up to 80 % smaller network. Moreover, we present a synapse reinforcement for the essential synapses using the dynamic reinforcement function. This function helps the network preserve an average spiking frequency close to the baseline. Combining these two techniques can produce trainable compressed networks that can achieve better accuracy when trained again.

5.2 Background

This section presents the used network, its components, the learning rule, and the dataset used in our experiments.

5.2.1 Network topology

The used topology in our experiments is similar to the work of Diehl and Cook [Diehl and Cook, 2015] on the classification of handwritten digits [LeCun et al., 1998]. We can see in Figure 5.1 that the topology is a 2-layer spiking neural network composed of the input layer, where every neuron represents a pixel and transforms the input (pixel density) to a Poisson-spike train. This layer is feed-forward fully connected to the next one using excitatory synapses. The second one is the unsupervised layer with lateral inhibition provided using the inhibitory connections between this layer's neurons. The lateral inhibition creates competition between the neurons and prevents one from taking over all the inputs. We assign excitatory neurons to classes based on the highest response to a digit class over a subset of the training set when

training is finished. We also use Homeostasis [Marder and Goaillard, 2006], which is an adaptive membrane threshold technique. Furthermore, we use Homeostasis to ensure that every excitatory neuron learns a unique feature. This technique is achieved using equation 5.1.

$$V_{\text{threshold}} = V_t + \tau \tag{5.1}$$

V_t is a constant that represents the initial threshold defined in the network. τ changes over time. Therefore, the value of τ will increase if the neuron fires often. This neuron will need more inputs to fire again and decays if the neuron's activity is less often, which guarantees a fair and balanced spiking activity between all the neurons in the network.

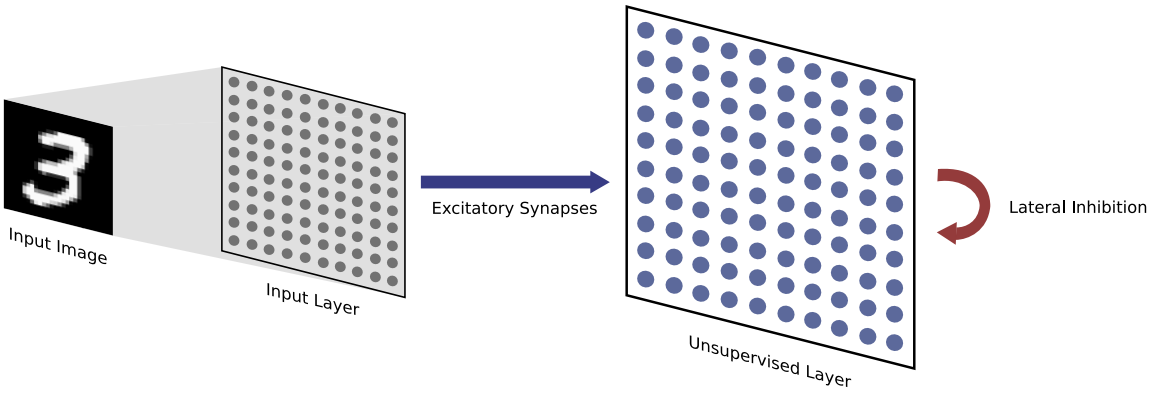

Figure 5.1: We use the input layer to convert the pixel intensity to the Poisson-spike train. The Poisson-spike trains are introduced to the unsupervised layer using excitatory synapses. The lateral inhibition is provided by the inhibitory synapses between neurons of the unsupervised layer.

In this work, we used the N2S3 simulator [Boulet et al., 2017]: an open-source, scalable spiking neuromorphic hardware simulator, written in Scala and based on the Akka actor system.

5.2.2 The neuron model

We use the Leaky-Integrated-and-Fire (LIF) model [Burkitt, 2006] to simulate the neuron membrane potential. LIF is one of the simplest models that include time in its operation. Based on the input from other neurons in the network, the neuron's membrane potential will get updated. The membrane potential keeps increasing until it reaches a precise threshold. Then, the neuron sends a spike and starts a refractory period before accumulating the received spikes again. We can see the membrane potential evolution in Figure 5.2. The LIF voltage equation is presented by:

$$\begin{aligned} \tau_{\text{leak}} \frac{\partial v}{\partial t} &= [v(t) - v_{\text{rest}}] + r_m z(t), \\ v &\leftarrow v_{\text{rest}} \quad \text{when} \quad v \geq v_{\text{th}} \end{aligned} \tag{5.2}$$

v is the membrane voltage, and v_{rest} is the resting potential after a spike. τ_{leak} is the time constant of the leak with $\tau_{\text{leak}} = r_m c_m$, where r_m is the membrane resistance and c_m is the membrane capacitance.

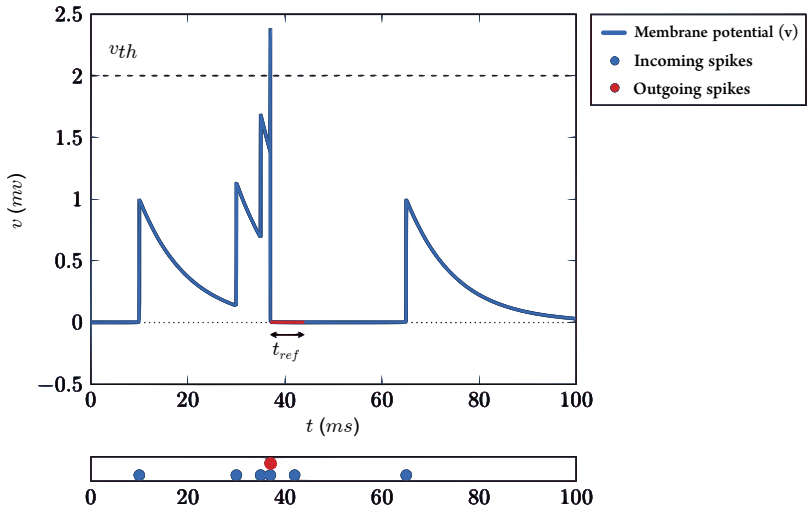

Figure 5.2: Membrane voltage activity of LIF

5.2.3 The learning rule

Most of the time, spiking Neural Networks use a local unsupervised learning rule (every synapse knows the activity of its post- and pre-synaptic neuron only). This

rule is biologically plausible and suitable for hardware implementation since it does not require a lot of computation. However, it is challenging to get high recognition rates compared to global learning rules because of its locality.

In our experiments, we use the Simplified Spike Timing Dependent Plasticity (STDP) rule, which is easier to implement with nanodevices [Querlioz et al., 2011]. This rule uses pre-synaptic and post-synaptic neuron activity to update the weight of a synapse. When a spike occurs in post-synaptic neuron at t_{post} , we check the presence of a spike in pre-synaptic (t_{pre}) in a limited window T_{STDP}. If a spike from the pre-synaptic neuron exists in this window, we boost the weight of the synapse. Otherwise, we lower it. This simplified STDP rule is represented as:

$$\Delta_w = \begin{cases} a_+ e^{-\beta_+ \frac{W - W_{min}}{W_{max} - W_{min}}} & t_{\text{post}} - t_{\text{pre}} < T_{\text{STDP}} \\ -a_- e^{-\beta_- \frac{W_{max} - W}{W_{max} - W_{min}}} & otherwise \end{cases} \tag{5.3}$$

a_+ and a_- represent the alpha parameters, β_+ and β_- are the beta parameters. The user sets the alpha and beta parameters at the start of the simulation. W is the current weight. W_{min} and W_{max} are the boundaries of the possible weight values. We set T_{STDP} to 50ms in our simulations.

5.2.4 The MNIST dataset

We use the MNIST handwritten dataset [LeCun et al., 1998] for the simulation. This dataset comprises 28x28 pixel images of handwritten digits with labels from 0 to 9. We split training images into six batches of 10,000 for our experiments to easily follow the network learning progress using the presented approach. The pictures are presented to our network input layer and processed in the format of Poisson-distributed spike trains. The input layer neuron weights are randomly initiated based on a uniform distribution between 0 and 1.

We can see in Table 5.1 the different parameters used in the network.

Input Stream			
MinFrequency	0Hz	MaxFrequency	22Hz
ExpositionDuration	350ms	PauseDuration	150ms
Neuron			
VoltageThreshold	35mv	RestingVoltage	0mv
RefractoryPeriod	1ms	InhibitionPeriod	10ms
STDP			
MaxWeight	1.0	MinWeight	0.0
a_+	0.01	a_-	0.005
β_+	1.5	β_-	2.5

Table 5.1: Network parameters used in the experiments

5.3 Presentation of the contribution

Compressing Spiking Neural Networks (SNNs) results in a network with reduced parameters and less complexity. In addition, compressed networks are easier to implement in hardware using the existing technologies [Shahsavari and Boulet, 2017, Merolla et al., 2011] and help to reduce the energy consumption of the network.

Pruning non-critical synapses is a widely used technique in ANNs and SNNs. The recent works use two different approaches to compress the network: at the end of the training process or during the training process. Both approaches use a static threshold.

Our tests showed that the neurons specialize in detecting a specific class early while learning. We can confirm this phenomenon in the network performance, which improves early (see Figure 5.3). Therefore, we can start removing synapses at an early stage. At the same time, the network can maintain most of its performance based on STDP, as indicated in recent works [Rathi et al., 2019, Saunders et al., 2019]. Let's run another training epoch on the new reduced network. We will get a network with better performance, with the opportunity to prune based on a higher threshold, resulting in a more compressed network with similar or better performance.

Based on this observation and the recent contributions, we propose *a progressive*

pruning function to control a dynamic threshold which we calculate and use to prune after each training batch. Additionally, we use *a synaptic weight reinforcement mechanism* to preserve the average spiking frequency of the network close to the baseline or better, by a reinforcement of the critical maintained synapses.

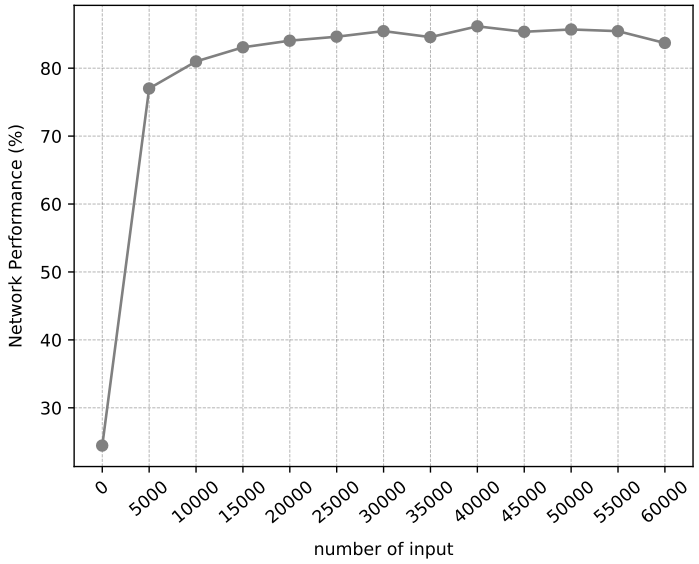

Figure 5.3: Learning progress in SNN using MNIST

5.3.1 Progressive Pruning

Progressive Pruning (PP) decreases the number of excitatory synapses between two layers. This reduction is performed after each batch during the training, using a dynamic pruning threshold $T_n, n \in \mathbb{N}$. We calculate this threshold using equation 5.4.

$$T_{n+1} = T_n + \alpha * (C_{r_n}/C_n) \qquad n \in \mathbb{N} \tag{5.4}$$

n is the batch number, and T_{n+1} represents the threshold to use for the next batch. α is a constant representing the initial threshold, and we discuss the selection of α in the next section ($\alpha = 0.05$ in this work). T_n is the old threshold used in the last batch ($T_0 = 0$). C_{r_n} represents the number of remaining synapses between the two layers at batch n. Finally, C_n represents the total number of synapses before pruning on this batch (which equals to the remaining synapses from the last batch, $C_n = C_{r_{n-1}}$).

In equation 5.4, we specify the threshold based on the previous pruning performance (if the pruning rate using T_n from the last batch is significant, Δ_{T_n} will be small and vice versa). Using this approach, we can prevent the network from having a significant performance degradation while eliminating the non-critical synapses.

5.3.2 Dynamic Synaptic Weight Reinforcement

Applying Progressive Pruning on the network decreases the average spiking network frequency, positively impacting our network's energy consumption. However, it may cause a frequency loss in multilayer networks, which is a known issue in Convolutional Spiking Neural Networks (CSNN) as described in [Falez et al., 2018]. Furthermore, the network cannot learn by having a low or no activity at all at the last layer.

Based on those observations, we propose the Dynamic Synaptic Weight Reinforcement (DSWR), which concerns the critical synapses of the network. Using DSWR, we push the neurons that did not specialize in a specific pattern or class to do so and maintain the average spiking frequency of the network near the baseline. We execute this procedure after each pruning operation based on equation 5.5.

$$W_{n+1} = W_n + \beta * T_n, \qquad n \in \mathbb{N}, \quad W \in [0, 1]. \tag{5.5}$$

n represents the batch number, and W_{n+1} is the new weight of the concerned synapse. W_n represents the current weight of that synapse. β is a defined constant determined based on experiments, and we discuss it in the next section (in this work β= 0.1). T_n represents the currently used threshold for pruning from equation 5.4. $\beta * T_n$ represents the value of the weight reinforcement (Δ_w) at batch n. Using this equation, we can see that the reinforcement value depends on the compression rate from equation 5.4.

Using the presented functions, we decide how much to increase the weight to maintain the average spike frequency based on the network state. Furthermore, in Figure 5.4 we can see a detailed flowchart concerning the proposed approach.

5.3.3 α and β parameters value selection

Picking the values of parameters α and β is a multiobjective optimization problem. The two objectives are the network performance and the compression rate. We exper-

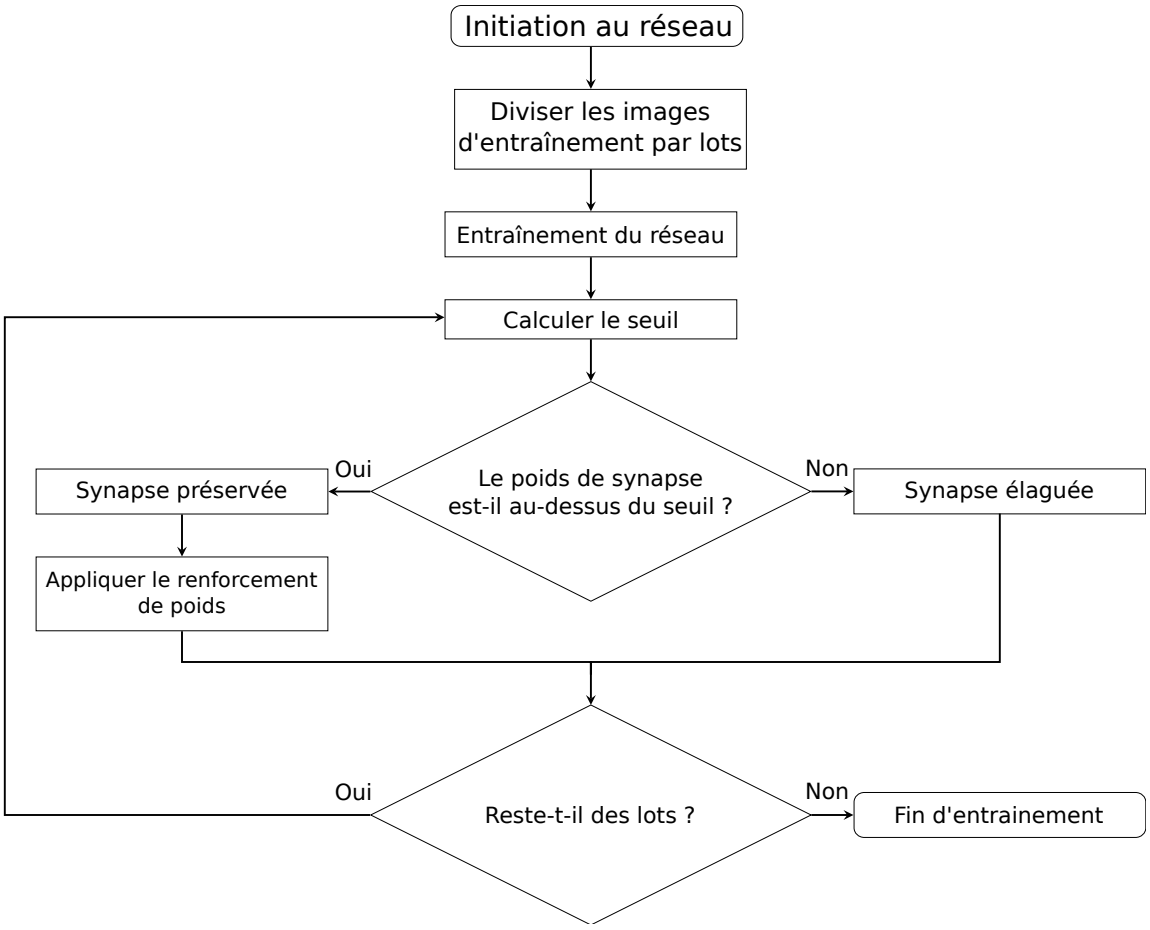

Figure 5.4: Flowchart of this approach

imented with different values and compared them for these two objectives, getting a Pareto front of non-dominated values. We finally picked the best performance values with a compression rate above 76 %.

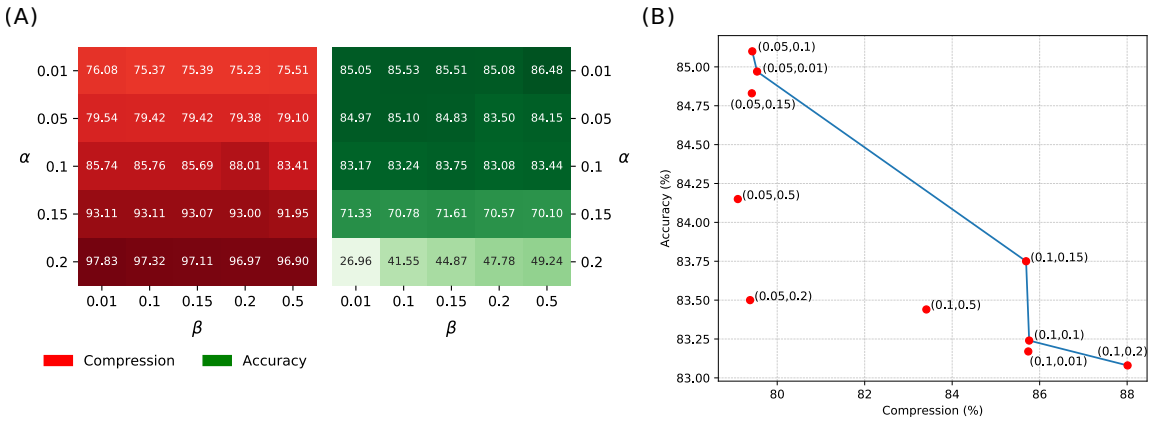

Figure 5.5: (A) α and β selection, (B) accuracy and compression depending on α and β (line = Pareto front).

In Figure 5.5 (A), by trying different combinations of α and β, we can get a high compressed network. However, a higher loss in network performance. The selection of α and β may change from one use case to another, depending on the context of the application. Figure 5.5 (B) contains the Pareto front of non-dominated solutions, which includes the possibilities that respect the following conditions:

- The performance should be better than the baseline recorded in [Diehl and Cook, 2015] (82.90 %).
- The compression rate needs to be better than 76 %.

Based on the results and these conditions, we select $\alpha = 0.05$ and $\beta = 0.1$, which delivers an accuracy of 85.10 % and a compression rate of 79.42 %, using a network composed of 100 neurons.

5.4 Experimental validation and discussion

This part describes our experimental validation using a network composed of 100 neurons. The network has an input layer connected to the unsupervised layer by

78400 excitatory synapses. We conducted the simulation on each case ten times using one epoch (of 60000 samples) and reported these ten simulations' averages. Furthermore, we ran tests on 400, 900, and 1600 neuron networks to evaluate the performance of our approach on different network sizes and compared it to similar works.

5.4.1 Evaluation of PP and DSWR

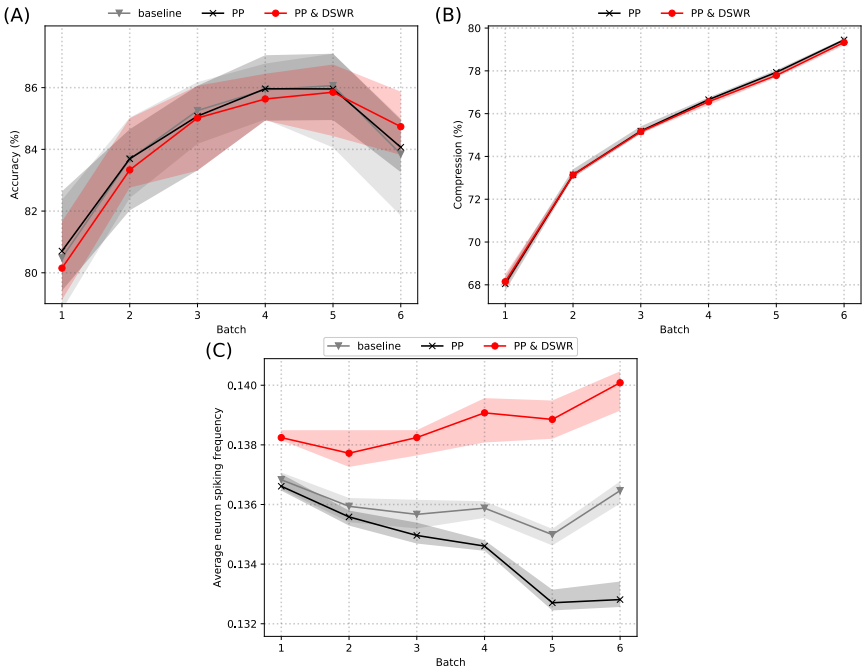

Figure 5.6: (A) Accuracy, (B) Compression, and (C) Average neuron spiking frequency for baseline, PP and PP & DSWR using a network of 100 neurons

This section evaluates the use of each function alone compared to the baseline, using a network of 100 neurons. We present in Figure 5.6 the accuracy, the compression rate, and the average spiking frequency of the network, using one training epoch of the MNIST dataset. From Figure 5.6 (A), by using PP & DSWR, we get a slightly better accuracy compared to the baseline and to using only PP. However, we can see a loss in accuracy after the fifth batch in all three cases (baseline, PP, and PP & DSWR). We relate this loss to MNIST's nature. The last batch contains

additional features (numbers), which the network could not learn well only in one epoch with 100 neurons. This issue is not present when using a larger network (see Figure 5.7 (B)). For compression rate, we can see that the compression goes up during the learning process, thanks to the proposed approach that guarantees a dynamic threshold. There is no visible difference between using only PP or PP & DSWR on the network compression rate, which is possibly due to the use of shallow networks only in our tests so far (see Figure 5.6 (B)).

Concerning the average neuron spike frequency (Figure 5.6 (C)), we see a loss in frequency when only PP is used compared to the baseline. However, using both PP and DSWR, we get a reasonable compression rate and accuracy while preserving a spike frequency similar to the baseline.

5.4.2 Accuracy and compression

# neurons	Paper	Learning rule	Pruning approach	Train after pruning ?	Epochs	Accuracy ± std	Compression ± std
100	Rathi et al. (2019)	Exponential STDP	Static threshold	No	1	79.50	75.00
	Diehl and Cook (2015)	Exponential STDP	–	–	1	82.90	–
	This work (baseline)	Simplified STDP	–	–	1	84.47 ± 1.55	–
	This work	Simplified STDP	Static threshold	No	1	79.69 ± 0.22	73.99 ± 0.05
	This work	**Simplified STDP**	**PP & DSWR**	**Yes**	**1**	**85.10 ± 0.83**	**79.42 ± 0.06**
400	Diehl and Cook (2015)	Exponential STDP	–	–	3	87.00	–
	This work (baseline)	Simplified STDP	–	–	1	87.77 ± 0.57	–
	This work	Simplified STDP	Static threshold	No	1	79.27 ± 0.51	70.14
	This work	**Simplified STDP**	**PP & DSWR**	**Yes**	**1**	**87.84 ± 0.40**	**79.52 ± 0.03**
900	This work (baseline)	Simplified STDP	–	–	1	88.35 ± 0.28	–
	This work	Simplified STDP	Static threshold	No	1	84.23 ± 0.24	66.77
	This work	**Simplified STDP**	**PP & DSWR**	**Yes**	**1**	**89.23 ± 0.91**	**77.28 ± 0.69**

Table 5.2: Accuracy and compression compared to the baseline and other similar works

In Table 5.2, we compare our results to the baseline and the recent works using similar network topology. Our approach compresses the network and gets higher accuracy in various network sizes due to the progressive and iterative pruning and reinforcement. For 100 neurons, the work presented by Rathi et al. [Rathi et al., 2019] could not keep the same accuracy after pruning. Possible reasons are the weight quantization technique, no training after pruning, or the one-time pruning approach. Using one epoch and without training after pruning, they had an accuracy of 79.50 %, which is less than the baseline with a 75 % compression rate. This performance is similar to our tests using a static threshold without training after pruning.

Using PP & DSWR with 100, 400, and 900 neuron networks, we get better accuracy and a compression rate of 79 %. In work presented by Diehl and Cook [Diehl and Cook, 2015], they conducted tests on a network composed of 400 neurons using three training epochs. The accuracy was 87 % (similar to our 400 neurons network baseline). We can see in Table 5.2 that we can compress the network up to 79.52 % with 87.84 % accuracy using PP & DSWR with only one epoch.

We can see clear improvement using PP and DSWR compared to our baseline. However, it is still far from state of the art on the MNIST dataset (over 98%). This gap is due to the size of the used network in our experiments, which is relatively smaller and contains just one trainable layer. In addition, we restrict our simulations to only one training epoch instead of having more than just one (like in Diehl et al. [Diehl et al., 2015]).

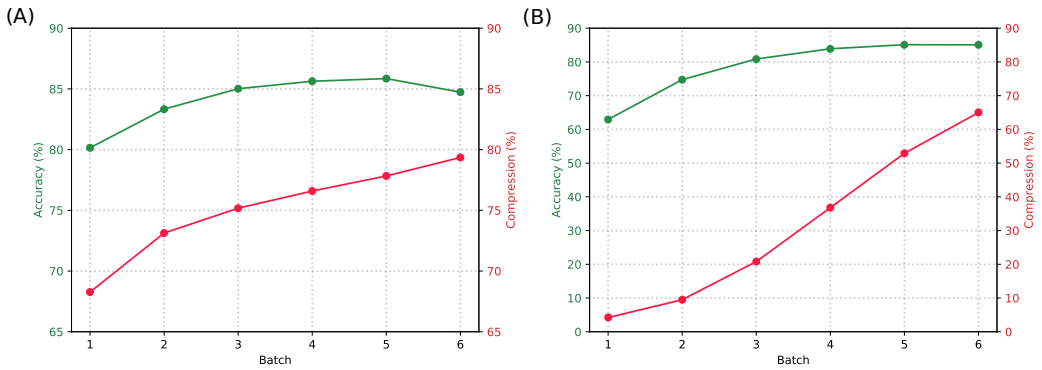

Figure 5.7: Accuracy and Compression evaluation using PP & DSWR for (A) 100 neurons and (B) 1600 neurons

In Figure 5.7 (A), the network is compressed for more than half the initial synaptic connections in the first batch. The compression rate increases after each batch, while the accuracy gradually does the same.

We can see in Figure 5.8 the evolution of the threshold during training, we can see that the threshold keeps increasing, it starts slowly at the beginning which is due to the fact that the network is at early stage during training. After that we see a constant increase in the threshold value until the end of the six batches.

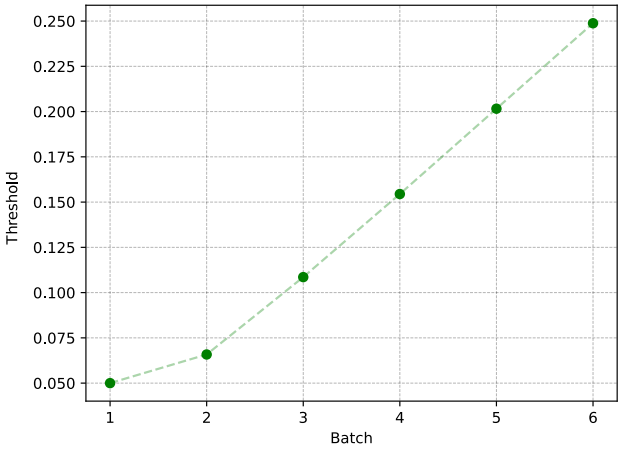

Figure 5.8: Threshold activity during training

# neurons	Paper	Learning rule	Pruning approach	Train after pruning ?	Epochs	Accuracy ± std	Compression ± std
1600	This work (baseline)	Simplified STDP	–	–	1	89.07 ± 0.77	–
	This work	Simplified STDP	Static threshold	No	1	82.40 ± 0.12	43.03
	This work	**Simplified STDP**	**PP & DSWR**	**Yes**	**1**	**85.31 ± 0.24**	**65.02 ± 0.04**
	Diehl and Cook (2015)	Exponential STDP	–	–	7	91.90	–

Table 5.3: Accuracy and compression using a network of 1600 neurons

5.4.3 Our approach on larger networks

Although the selected parameters (α and β) give good results with small to medium networks, as shown in Table 5.3, using this approach on a larger network may have a different impact. Therefore, in this part, we present the results we had while preserving the same parameters (α and β) and using a network of 1600 neurons equivalent to the one used in Diehl and Cook [Diehl and Cook, 2015].

From Figure 5.7 (B), when compared to the previous experiments, we notice that we have a loss in accuracy (85.31 %). Besides, the network needs up to three batches to get accuracy above 80 % compared to smaller networks where only one batch is enough. Furthermore, for the compression rate, we can observe that the network this time is less compressed (65.02 %) compared to smaller networks and the results of Diehl and Cook [Diehl and Cook, 2015]. Besides, we can see that the compression starts from a small rate (4 %) in the first batch, which is not the case with earlier examples with smaller networks.

To investigate and explain this result when using a larger network. We suggest two hypotheses based on our observations:

1. The initial value of α and β used with small to medium networks may not work with a large network and need to be appropriately adjusted.
2. A larger network with more neurons requires more time to learn before pruning, so we need to review the number of batches per epoch before pruning to give more time for the network to learn.

To answer this question of what is happening in the network, we can check Figure 5.9. The network did not have enough time to start learning features, and the low accuracy represents this after the first batch. This decrease in network accuracy is due to the competition between neurons to learn. The more neurons we have, the more time it will take to start learning. From the compression rate progress in Figure 5.7 (B), we can discard the first hypothesis. Since the problem is not linked to pruning using a high threshold, based on the low compression rate we had (4 %). It is associated with the second hypothesis about the time required for the network to start learning, which also justifies the lower compression rate.

When dealing with larger networks, it is preferable to decrease the number of batches per epoch, delay the prune operation, and use more than one epoch for

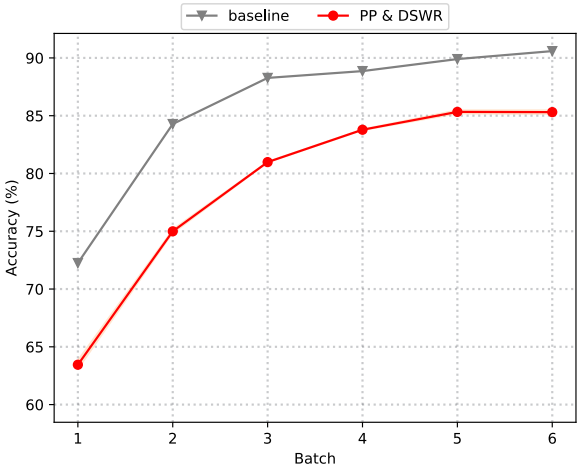

Figure 5.9: Accuracy for the baseline and PP & DSWR using a 1600 neuron network

Number of neurons	**Number of epochs**	**Accuracy ± std**	**Compression ± std**
1600	1	85.31 ± 0.24	65.02 ± 0.04
1600	**3**	**89.65 ± 0.11**	**80.80 ± 0.43**

Table 5.4: Accuracy and Compression rate using 1 and 3 training epochs

training to give the network more time to learn. For example, in Table 5.4, using three epochs and applying the presented approach twice each epoch, we get better accuracy than the baseline (89.65 %) with a compression rate of 80.80 %.

5.4.4 Trainable Compressed Network

Lowering the number of synapses in a network is useful for time and energy reduction, especially when implementing it on hardware. One feature commonly tested with reduced networks is the possibility of having trainable networks after compression. Unfortunately, as for now, it is not always possible to train a small neural network and get good results.

In [Frankle and Carbin, 2019, Zhou et al., 2019], the authors present a lottery tickets hypothesis for CNNs. This hypothesis states that a randomly initiated dense neural network contains a subnetwork that, if isolated and trained alone, can match the accuracy of the original network. Unfortunately, it is hard to train a pruned network from the beginning in CNNs, which gives worse accuracy than the original model, and it is considered not trainable. The solution was to train the pruned network with the initial synapse weights, not randomly initiated synapse weights, to learn well. However, we did not yet test such a hypothesis in Spiking Neural Networks (SNNs).

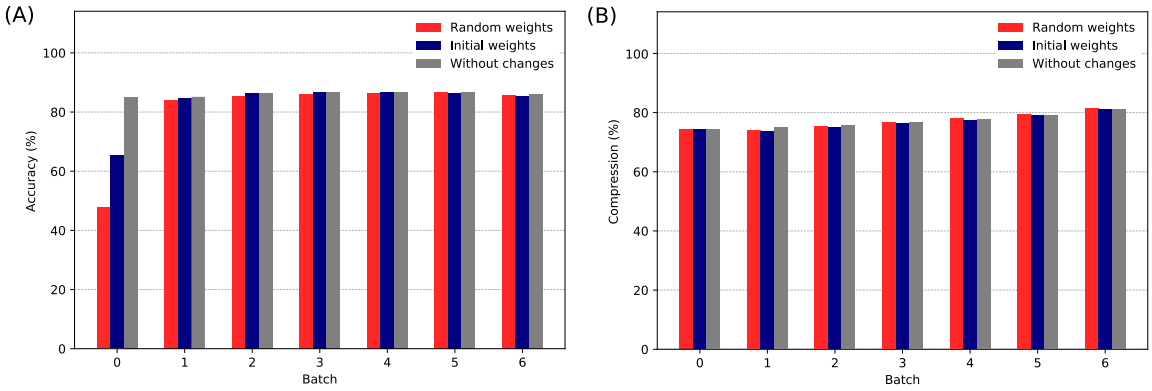

Figure 5.10: Accuracy (A) and Compression rate (B) during training of a 100 neurons network

In Figure 5.10, we test if the compressed network is trainable using a 100 neuron network. We try three use-cases: using the new compressed network without

changing synapse weights, using random initial weights, or using the initial weights of the original network. We notice that after one batch that the network with initial synaptic weights without training reaches more than 60 % accuracy compared to the network with randomly initiated weights which barely reaches 50 %. This observation was also shown in [Frankle and Carbin, 2019] for CNNs. At the end of the six training batches, we get an accuracy near 85 % for the three use-cases. For the compression rate, we observe that the network is more compressed in the three use-cases to reach a compression rate of 80 % while maintaining a better accuracy compared to baseline. This result confirms that some observations presented in [Frankle and Carbin, 2019] are valid in SNNs too. The trainable network gets a higher compression rate when trained again irrespectively of the initialization strategy of the synaptic weights that only impacts the speed of learning, not the quality.

5.4.5 Pruning batch effect on the compression

During our experiments, we used a single MNIST dataset epoch (60k input samples) divided into six batches of 10k samples each. We study in this section the influence of the size and number of batches on the accuracy and compression. For instance, in Figure 5.11, we compare one batch of 60k samples, three batches of 20k samples, and six batches of 10k samples using a 100 neurons network.

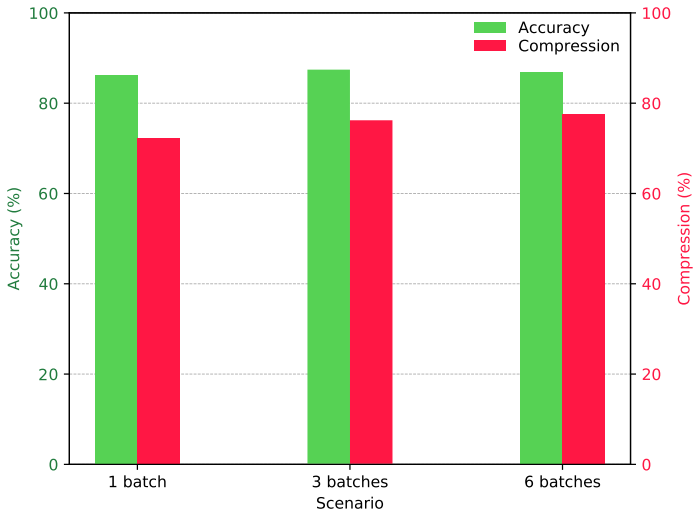

Figure 5.11: Accuracy and compression rate using 1, 3, and 6 batches for a 100 neurons network

We see in Figure 5.11 that the accuracy of the network did not decrease, and there is no visible impact of the variation of the number of batches on the accuracy. However, the compression rate increases when we increase the number of batches because of the approach with the dynamic threshold. So, in this case, having more than six batches will produce a better compression rate but with a possible loss in network accuracy. Therefore, the number of the batch to use depends on the application, similar to the α and β.

5.5 Conclusion

Recently, many studies shed light on the irrational effectiveness of Neural Networks and our profound inability to understand causality and correlation in neural networks. Therefore, we can see pruning a neural network as a discriminant analysis for dimensionality reduction. This chapter presents a novel approach to reduce spiking neural network size while preserving a similar or better accuracy by using Progressive Pruning (PP) with a dynamic threshold and a Dynamic Synaptic Weight Reinforcement (DSWR). We proposed this approach after we analyzed the network activity using VS2N. Besides, in this chapter, we compare our approach to related work using a static threshold. This work presents another possibility to prune a network and even combine it with the existing techniques. The novelty of this approach is the use of a dynamic threshold instead of a static one, which is the case with existing works.

The proposed approach can produce a network with better accuracy than the non-compressed network when applied to small or medium networks and only one epoch. But when dealing with larger networks, we need more than one training epoch with larger batches to provide sufficient time for the network to learn before pruning. Moreover, only one epoch of training and the use of small networks are the reason behind the lower performance than state of the art for the SNN using MNIST dataset, which is recorded on much bigger networks with more than one training epoch. On the other hand, the compressed networks were proven trainable to reach even better accuracy and a better compression rate by using the weights of the initial synapses of the original network. The idea behind it is from Frankle et al. [Frankle and Carbin, 2019] work on convolutional neural networks. This result implies that the Lottery Ticket Hypothesis explained in the same work may also be true for spiking neural networks. In future work, we will test our approach using other challenging datasets multilayer networks and study the behavior of the smaller

network by visual analysis using VS2N to improve the proposed approach.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

SNN can provide low-power solutions with less carbon footprint through neuromorphic hardware. Despite not being able to compete with conventional neural networks and deep learning due to low performance. Moreover, the black-box nature of neural networks, in general, makes it challenging to analyze and debug; thus, it is not a trivial task to improve them. The interpretability of neural networks in recent years has become increasingly important since counting only on a single metric, such as the network accuracy, is an incomplete description of most real-world tasks [Doshi-Velez and Kim, 2017]. In this manuscript, we addressed the issue of spiking neural networks interpretability using the visual analysis approach. To improve the network performance and better understand the internal phenomena. The collected information and improvement on the network will ensure better spiking neural networks training using software simulators and an optimized implementation on neuromorphic hardware later on.

The first contribution in this manuscript is the study of the visualization techniques in SNN simulators (Chapter 3). This study from the technical and visual analysis aspect of the simulators shed light on the diversity of the used programming languages—moreover, the provided visualization techniques and their similarity [Elbez et al., 2018]. At the end of this study, we concluded that to analyze SNNs using visual analysis, we need to have tools that are more dedicated to analysis than depending on what simulators provide for this task. This limitation is because the visualizations provided by the simulators are primarily for monitoring purposes rather than analysis. Besides, a simulation usually takes most of the machine resources, which makes it challenging to perform visual analysis using the same tool. Therefore, we ended this study by suggesting separating the simulation process from the

analysis one by having dedicated tools for analyzing spiking neural networks. Those tools can support the big data generated by the large activity of the network during training and provide enough interactivity during the analysis.

The second contribution in this work is the development and presentation of VS2N (Visualization tool for Spiking Neural Networks) [Elbez et al., 2021]. In Chapter 4, we described VS2N, a web-based tool for post-mortem interactive dynamic visualization and analysis of spiking neural network simulations. The novelty of VS2N compared to the existing visual analysis tools can be summarized in four points: 1. Modular nature: VS2N groups the visualizations into modules. Each module may target a specific question or problem, and anyone can add new modules for a particular analysis. 2. Simulator-independent: we can use any simulator as long as the collected data follows specific schemas, which VS2N can understand. 3. Scalability: backed by the combination of Apache Spark and MongoDB for data processing, we can deploy VS2N on multi-nodes or clusters for more solid performance. 4. Dynamic analytics: VS2N provides the possibility to walk in time with the evolution of the network during activity, which is not possible using the existing tools for data analysis. This feature is significant when the network evolution is over hours of activity, which is the case in spiking neural networks. To showcase the use of VS2N, we presented two use-cases. The first use-case is *"The MNIST last 10k effect"*: when using the MNIST dataset, we saw an accuracy drop in the last 10k input of the training data. We can see this drop on small to medium networks (less than 1000 neurons). However, it does not appear on large networks. Furthermore, from this observation, we assume that the last 10k contains different images, which affects the network learning process. As a result, by training the same network using MNIST inverted, we see that this drop disappears from the last 10k and appears at the start of the training, as seen in Figure 4.20. Using VS2N after 50k of input, we saw an interesting pattern at the beginning of the last 10k (Figure 4.21(B)), with a noticeable increase in the synaptic weights update (more than 10%) that remains until the end of the simulation. Since we used a single dense layer in that use-case, the increase in synaptic weights update means that the patterns learned by the neurons are changing, which will affect the network performance and cause the effect we observed in Figure 4.20. The second use-case is *"Network Compression"*: using VS2N, we observed the general activity of the network once pruned and the impact of this operation on two randomly chosen neurons. After the analysis, we found the expected behavior from the network's neurons. Moreover, an unexpected pattern appeared. After the investigations, we found out that it is related to an issue with how the prune operation is performed in the simulator. Therefore, using VS2N, we confirmed the existence of this issue

that was affecting the simulation time when compressing the network. Finally, we compared similar visualization tools based on selected features in this chapter.

The last contribution in this manuscript is the presentation of a novel approach to compress a spiking neural network, which was proposed based on the visual analysis conducted on SNN using VS2N [Elbez et al., 2022]. By taking a look at the stat of the art when it comes to pruning a neural network, we can see that the existing works in terms of pruning focus on how to treat the pruned synapses, when to prune, or the used criteria for pruning while keeping the same static threshold. However, our contribution deals with the threshold value selection and how to treat the maintained synapses in the network since we observed during training that neurons' development is happening most in the first batches of training. Therefore, the need to apply pruning after a couple of batches. However, since this scenario may change from one input type to another, we need to have dynamic threshold values that adapt to the situation during training, and that is what we present in Chapter 5 using PP (Progressive Pruning) and DSWR (Dynamic Synaptic Weight Reinforcement). Progressive pruning decreases the number of synapses between two layers. This reduction is performed after each batch during the training, using a dynamic pruning threshold $T_n, n \in \mathbb{N}$, which we calculate using equation 5.4. Dynamic Synaptic Weight Reinforcement (DSWR) concerns the critical synapses of the network. Using DSWR, we maintain the average spiking frequency of the network near the baseline, and we execute it after each pruning operation. This approach can improve accuracy than the non-compressed network when applied to small or medium networks and only one epoch. But when dealing with larger networks, we need more than one training epoch with larger batches to provide sufficient time for the network to learn before pruning.

6.2 Future work

The interest in the interpretability of neural networks, in general, appeared in recent years due to the increased need to justify and explain the results of deployed models in real-life applications and also to the different discrimination examples recorded. Additional knowledge and understanding will help reduce such issues since we will detect them before production. Visual analysis is one of the ways to interpret neural networks in general and SNN in particular. Still, additional work is necessary to make the visual analysis effective, especially with the increasing complexity of neural

network models, and to study the visual analysis effect on hardware-based networks running on neuromorphic architectures.

6.2.1 Big data in visual analysis

One of the key elements to guarantee a good visual analysis is the degree of interactivity. It can be challenging to achieve a high degree of interactivity when dealing with big data, which is the case in spiking neural networks. Many works focus on challenges and tools to manage large quantities of data [Wang et al., 2017, Maitrey and Jha, 2015, Fu et al., 2016], and some of them focus on exploring the existing big-data frameworks for machine learning applications [Gupta et al., 2017]. On the other hand, big-data visualization demands more processing and requirements to manage a large amount of data and provide a visualization with good interactivity for analysis purposes. This can be seen in the different works on big-data visualization [Wang et al., 2015, Ali et al., 2016, Qin et al., 2018, Li and Zhou, 2017, Sansen et al., 2017], which proposes techniques for visual analysis and use cases.

Since they are data-driven models, neural networks require a lot of data for training. As a result, the training may take a lot of time, and the quantity of data collected can be huge, making visual analysis a challenging task. Moreover, in the case of spiking neural networks, we have the asynchronous nature of the network and use of the temporal dimension making it more challenging to analyze. Despite being able to explore big data frameworks such as Hadoop [Ghazi and Gangodkar, 2015], Spark [Fu et al., 2016], and MongoDB to manage and store a large quantity of data, we still have to find a way to support pre-processing of this data which takes time and resources when executed for the first time (which is the case in VS2N). This support can be done by either proposing new ways to process the data effectively or by reducing the amount of data kept from the network activity by defining the objectives of a visual analysis from the beginning.

6.2.2 CSNN and hardware-based networks

In this manuscript, our work mainly focuses on STDP-based networks using simple datasets. However, the visual analysis of more complex networks such as convolutional spiking neural networks (CSNN) represents more challenges since deeper

networks require more computation to analyze data flow and adapted visualizations. Moreover, we can group the learning mechanism in SNN into two categories: local and global learning rules. Therefore, in this manuscript, we focus on local learning rules (STDP) and the analysis of global learning rules, such as Surrogate Gradient Learning [Neftci et al., 2019] is a great opportunity to learn more about this learning approach in SNN, which we did not discuss in our work. Furthermore, since CNN performs better than SNN in different tasks, some works focused on converting a trained CNN to a SNN [Rueckauer et al., 2017, Xing et al., 2019] to get a trained network with high accuracy and low power consumption. This approach makes it possible to avoid training SNN. The visual analysis of such a network should help us learn more about the difference between the activity in this type and a trained SNN. Moreover, the implementation of SNN in neuromorphic architectures is another approach to accelerate the learning phase and deploy prototypes. The visual analysis of hardware-based networks is possible since we can extract the network activity and use it the same way we did with the network activity obtained from the simulator. Finally, Since the appearance of bio-inspired sensors (like event-based cameras), other types of datasets appeared which contain only spikes like the N-MNIST [Orchard et al., 2015], and any data collected from those sensors. Since our experiments are conducted mainly on images in this manuscript, the analysis of other types of input data and the network response to it can help us better understand our data and compare the network behavior with a similar network with static images as input.

6.2.3 Evaluation metrics of SNN

Usually, network accuracy is one of the main metrics of neural networks, sometimes it is the only one used to evaluate a network, we can observe this in academia in recent years (see Figure 6.1).

This type of research that seeks to improve the accuracy while disregarding the cost (referred to also as Red AI [Schwartz et al., 2020]) also contributes to the carbon emissions issue. Furthermore, with the increase in network complexity, the need for more power to run, the lack of interpretability, and the discrimination issues that appeared in some applications. Therefore, we need to revise our metrics to evaluate a neural network, particularly SNN. In addition, other metrics such as energy consumption, interpretability level, and accuracy can change how we assess neural networks. For example, a network with state-of-the-art accuracy but requires enormous energy to run or lacks a minimum level of interpretability is not better

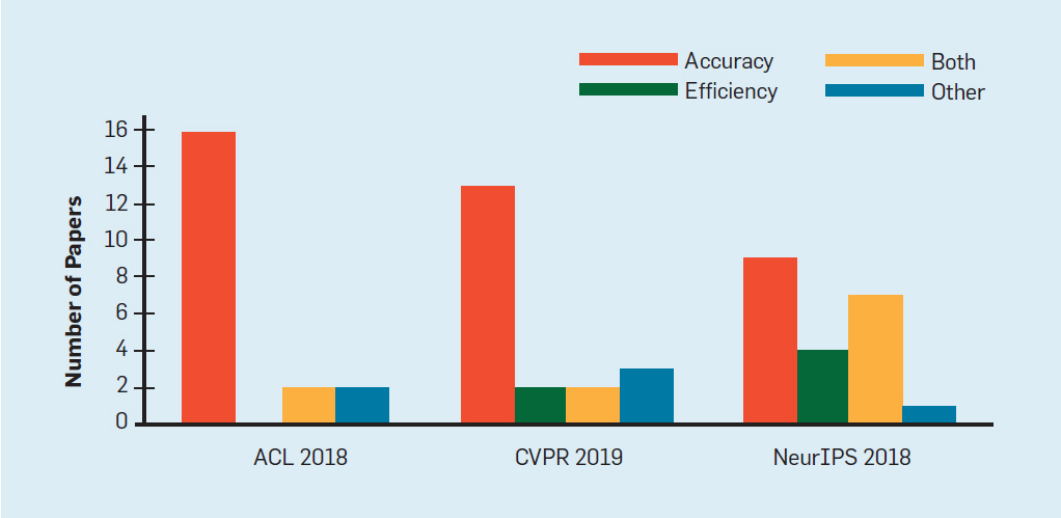

Figure 6.1: The proportion of papers that target accuracy, efficiency, both or other from a random sample of 60 papers from top AI conferences [Schwartz et al., 2020].

than another network with less accuracy but requires less energy to run or a network that we can easily interpret its results.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

[Abbott, 1999] Abbott, L. F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5):303–304.

[Ali et al., 2016] Ali, S. M., Gupta, N., Nayak, G. K., and Lenka, R. K. (2016). Big data visualization: Tools and challenges. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pages 656–660.

[Avilov et al., 2020] Avilov, O., Rimbert, S., Popov, A., and Bougrain, L. (2020). Deep Learning Techniques to Improve Intraoperative Awareness Detection from Electroencephalographic Signals. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*, pages 142–145.

[Babadi, Baktash and F. Abbott, L., 2016] Babadi, Baktash and F. Abbott, L. (2016). Stability and Competition in Multi-spike Models of Spike-Timing Dependent Plasticity. *PLOS Computational Biology*, 12(3):e1004750.

[Bekolay et al., 2014] Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., Choo, X., Voelker, A., and Eliasmith, C. (2014). Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7.

[Benjamin et al., 2014] Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. M., Alvarez-Icaza, R., Arthur, J. V., Merolla, P. A., and Boahen, K. (2014). Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE*, 102(5):699–716.

[Bi and Poo, 1998] Bi, G. Q. and Poo, M. M. (1998). Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type. *Journal of Neuroscience*, 18(24):10464–10472.

[Bichler et al., 2012] Bichler, O., Querlioz, D., Thorpe, S. J., Bourgoin, J.-P., and Gamrat, C. (2012). Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Networks*, 32:339–348.

[Boulet et al., 2017] Boulet, P., Devienne, P., Falez, P., Polito, G., Shahsavari, M., and Tirilly, P. (2017). N2S3, an Open-Source Scalable Spiking Neuromorphic Hardware Simulator. report, Université de Lille 1, Sciences et Technologies ; CRIStAL UMR 9189.

[Brette, 2015] Brette, R. (2015). Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain. *Frontiers in Systems Neuroscience*, 9.

[Brette and Goodman, 2011] Brette, R. and Goodman, D. F. M. (2011). Vectorized algorithms for spiking neural network simulation. *Neural Computation*, 23(6):1503–1535.

[Brette et al., 2007] Brette, R., Rudolph, M., Carnevale, T., et al. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[Burkitt, 2006] Burkitt, A. N. (2006). A Review of the Integrate-and-fire Neuron Model: I. Homogeneous Synaptic Input. *Biological Cybernetics*, 95(1):1–19.

[Carreira-Perpinan and Idelbayev, 2018] Carreira-Perpinan, M. A. and Idelbayev, Y. (2018). "Learning-Compression" Algorithms for Neural Net Pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8532–8541.

[Chen et al., 2018] Chen, R., Ma, H., Xie, S., Guo, P., Li, P., and Wang, D. (2018). Fast and Efficient Deep Sparse Multi-Strength Spiking Neural Networks with Dynamic Pruning. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

[Cho et al., 2019] Cho, S.-G., Beigné, E., and Zhang, Z. (2019). A 2048-Neuron Spiking Neural Network Accelerator With Neuro-Inspired Pruning And Asynchronous

Network On Chip In 40nm CMOS. In *2019 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4.

[Cun et al., 1990] Cun, Y. L., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems 2*, pages 598–605. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Davies et al., 2018] Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y., Wild, A., Yang, Y., and Wang, H. (2018). Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1):82–99.

[Davison et al., 2009] Davison, A., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., and Yger, P. (2009). Pynn: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2.

[Dhar, 2020] Dhar, P. (2020). The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2(8):423–425.

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

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

[Doshi-Velez and Kim, 2017] Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv: Machine Learning*.

[Dragly et al., 2017] Dragly, S.-A., Mobarhan, M. H., Solbrå, A. V., Tennøe, S., Hafreager, A., Malthe-Sørenssen, A., Fyhn, M., Hafting, T., and Einevoll, G. T. (2017). Neuronify: An Educational Simulator for Neural Circuits. *eNeuro*, 4(2):ENEURO.0022–17.2017.

[Elbez et al., 2018] Elbez, H., Benhaoua, K., Devienne, P., and Boulet, P. (2018). Visualization Techniques in SNN Simulators. In *3rd International Conference on Multimedia Information Processing, CITIM'2018*, Mascara, Algeria.

[Elbez et al., 2021] Elbez, H., Benhaoua, M. K., Devienne, P., and Boulet, P. (2021). Vs2n : Interactive dynamic visualization and analysis tool for spiking neural networks. In *2021 International Conference on Content-Based Multimedia Indexing (CBMI)*, pages 1–6.

[Elbez et al., 2022] Elbez, H., Benhaoua, M. K., Devienne, P., and Boulet, P. (2022). Progressive compression and weight reinforcement for spiking neural networks. *Concurrency and Computation: Practice and Experience*.

[Eppler et al., 2009] Eppler, J. M., Helias, M., Muller, E., Diesmann, M., and Gewaltig, M.-O. (2009). PyNEST: a convenient interface to the NEST simulator. *Frontiers in Neuroinformatics*, 2.

[Falez et al., 2018] Falez, P., Tirilly, P., Bilasco, I. M., Devienne, P., and Boulet, P. (2018). Mastering the Output Frequency in Spiking Neural Networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

[Falez et al., 2019] Falez, P., Tirilly, P., Marius Bilasco, I., Devienne, P., and Boulet, P. (2019). Multi-layered Spiking Neural Network with Target Timestamp Threshold Adaptation and STDP. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

[Few, 2009] Few, S. (2009). *Now You See It: Simple Visualization Techniques for Quantitative Analysis*. Analytics Press, Oakland, CA, USA, 1st edition.

[Few, 2017] Few, S. (2017). Data visualization effectiveness profile. *Perceptual Edge*, 10:12.

[FitzHugh, 1961] FitzHugh, R. (1961). Impulses and Physiological States in Theoretical Models of Nerve Membrane. *Biophysical Journal*, 1(6):445–466.

[Forum, 1994] Forum, M. P. (1994). Mpi: A message-passing interface standard. Technical report, University of Tennessee, USA.

[Frankle and Carbin, 2019] Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

[Freitas et al., 2002] Freitas, C. M. D. S., Luzzardi, P. R. G., Cava, R. A., Winckler, M., Pimenta, M. S., and Nedel, L. P. (2002). On Evaluating Information Visualization Techniques. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '02, pages 373–374, New York, NY, USA. ACM.

[Frézal and Félix, 2015] Frézal, L. and Félix, M.-A. (2015). C. elegans outside the Petri dish. *eLife*, 4:e05849.

[Fu et al., 2016] Fu, J., Sun, J., and Wang, K. (2016). SPARK – A Big Data Processing Platform for Machine Learning. In *2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII)*, pages 48–51.

[Furber et al., 2014] Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The SpiNNaker Project. *Proceedings of the IEEE*, 102(5):652–665.

[Gerstner et al., 1993] Gerstner, W., Ritz, R., and van Hemmen, J. L. (1993). Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns. *Biological Cybernetics*, 69(5):503–515.

[Gewaltig and Diesmann, 2007] Gewaltig, M.-O. and Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430.

[Ghazi and Gangodkar, 2015] Ghazi, M. R. and Gangodkar, D. (2015). Hadoop, MapReduce and HDFS: A Developers Perspective. *Procedia Computer Science*, 48:45–50.

[Goodman and Brette, 2008] Goodman, D. and Brette, R. (2008). Brian: a simulator for spiking neural networks in Python. *Frontiers in Neuroinformatics*, 2:5.

[Gupta et al., 2017] Gupta, A., Thakur, H. K., Shrivastava, R., Kumar, P., and Nag, S. (2017). A Big Data Analysis Framework Using Apache Spark and Deep Learning. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 9–16.

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1135–1143, Montreal, Canada. MIT Press.

[Harris et al., 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[Hassibi et al., 1994] Hassibi, B., Stork, D. G., and Wolff, G. (1994). Optimal Brain Surgeon: Extensions and performance comparisons. In *Advances in Neural Information Processing Systems 6*, pages 263–270. Morgan-Kaufmann.

[Hazan et al., 2018] Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., and Kozma, R. (2018). BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python. *Frontiers in Neuroinformatics*, 12:89.

[He et al., 2019] He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4335–4344.

[Hebb, 1949] Hebb, D. O. (1949). *The organization of behavior; a neuropsychological theory*. The organization of behavior; a neuropsychological theory. Wiley, Oxford, England.

[Hines and Carnevale, 1997] Hines, M. L. and Carnevale, N. T. (1997). The NEURON simulation environment. *Neural Computation*, 9(6):1179–1209.

[Hinton et al., 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [cs, stat]*.

[Hodgkin and Huxley, 1952] Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544.

[Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.

[Huang et al., 2018] Huang, Q., Zhou, K., You, S., and Neumann, U. (2018). Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, 9(3):90–95.

[Huttenlocher, 1979] Huttenlocher, P. R. (1979). Synaptic density in human frontal cortex - developmental changes and effects of aging. *Brain Research*, 163(2):195–205.

[Izhikevich, 2003] Izhikevich, E. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572.

[J, 2008] J, Y. (2008). Simbrain: A visual framework for neural network analysis and education. *Interactive Educational Media for the Neural and Cognitive Sciences, Brains, Minds & Media*.

[Kahng et al., 2018] Kahng, M., Andrews, P. Y., Kalro, A., and Chau, D. H. (2018). ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97.

[Kasiński et al., 2009] Kasiński, A., Pawłowski, J., and Ponulak, F. (2009). 'SNN3DViewer' - 3D Visualization Tool for Spiking Neural Network Analysis. In *Computer Vision and Graphics*, Lecture Notes in Computer Science, pages 469–476, Berlin, Heidelberg. Springer.

[Kelley, 1960] Kelley, H. J. (1960). Gradient Theory of Optimal Flight Paths. *ARS Journal*, 30(10):947–954.

[Kernighan, 1984] Kernighan, P. (1984). Hoc - A User Extendable Interactive Language.

[Kim et al., 2018] Kim, J., Park, S., and Kwak, N. (2018). Paraphrasing Complex Network: Network Compression via Factor Transfer. *NeurIPS*.

[Kohonen, 1982] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

[Kosslyn, 2006] Kosslyn, S. M. (2006). *Graph Design for the Eye and Mind*. Oxford University Press.

[Krause et al., 2014] Krause, J., Perer, A., and Bertini, E. (2014). INFUSE: Interactive Feature Selection for Predictive Modeling of High Dimensional Data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1614–1623.

[Lazzaro et al., 1989] Lazzaro, J., Ryckebusch, S., Mahowald, M. A., and Mead, C. A. (1989). Winner-take-all networks of O(N) complexity. In *Advances in neural information processing systems 1*, pages 703–711. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lee et al., 2016] Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience*, 10.

[Legenstein et al., 2008] Legenstein, R., Pecevski, D., and Maass, W. (2008). A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback. *PLOS Computational Biology*, 4(10):e1000180.

[Li et al., 2017] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

[Li and Zhou, 2017] Li, M. and Zhou, Q. (2017). Industrial Big Data Visualization: A Case Study Using Flight Data Recordings to Discover the Factors Affecting the Airplane Fuel Efficiency. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 853–858.

[Liu et al., 2015] Liu, B., Wen, W., Chen, Y., Li, X., Wu, C.-R., and Ho, T.-Y. (2015). EDA Challenges for Memristor-Crossbar based Neuromorphic Computing. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 185–188, Pittsburgh, Pennsylvania, USA. Association for Computing Machinery.

[Liu et al., 2019] Liu, P., Wang, J., Sangaiah, A. K., Xie, Y., and Yin, X. (2019). Analysis and Prediction of Water Quality Using LSTM Deep Neural Networks in IoT Environment. *Sustainability*, 11(7):2058.

[Liu et al., 2018] Liu, Z., Xu, J., Peng, X., and Xiong, R. (2018). Frequency-domain dynamic pruning for convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[Luo et al., 2019] Luo, J.-H., Zhang, H., Zhou, H.-Y., Xie, C.-W., Wu, J., and Lin, W. (2019). Thinet: Pruning cnn filters for a thinner net. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(10):2525–2538.

[Maitrey and Jha, 2015] Maitrey, S. and Jha, C. K. (2015). MapReduce: Simplified Data Analysis of Big Data. *Procedia Computer Science*, 57:563–571.

[Marder and Goaillard, 2006] Marder, E. and Goaillard, J.-M. (2006). Variability, compensation and homeostasis in neuron and network function. *Nature Reviews Neuroscience*, 7(7):563–574.

[Markram et al., 1997] Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science (New York, N.Y.)*, 275(5297):213–215.

[Marks, 2017] Marks, S. (2017). Immersive visualisation of 3-dimensional spiking neural networks. *Evolving Systems*, 8(3):193–201.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Merolla et al., 2011] Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., and Modha, D. S. (2011). A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *2011 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4.

[Merolla et al., 2014] Merolla, P. A., Arthur, J. V., Alvarez-Icaza, et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673.

[Miguel Grinberg, 2018] Miguel Grinberg (2018). *Flask Web Development, 2nd Edition*. O'Reilly Media, Inc.

[Miikkulainen, 2010] Miikkulainen, R. (2010). Topology of a Neural Network. In *Encyclopedia of Machine Learning*, pages 988–989. Springer US, Boston, MA.

[Moghar and Hamiche, 2020] Moghar, A. and Hamiche, M. (2020). Stock Market Prediction Using LSTM Recurrent Neural Network. *Procedia Computer Science*, 170:1168–1173.

[Morris and Lecar, 1981] Morris, C. and Lecar, H. (1981). Voltage oscillations in the barnacle giant muscle fiber. *Biophysical Journal*, 35(1):193–213.

[Mozafari et al., 2019] Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., and Masquelier, T. (2019). SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks With at Most One Spike per Neuron. *Frontiers in Neuroscience*, 13:625.

[Neftci et al., 2019] Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks. *IEEE Signal Processing Magazine*, 36(6):51–63.

[Noh et al., 2015] Noh, H., Hong, S., and Han, B. (2015). Learning Deconvolution Network for Semantic Segmentation. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1520–1528, USA. IEEE Computer Society.

[Nowke et al., 2013] Nowke, C. et al. (2013). VisNEST: Interactive analysis of neural activity data. In *2013 IEEE Symposium on Biological Data Visualization (BioVis)*, pages 65–72.

[Orchard et al., 2015] Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. *Frontiers in Neuroscience*, 9.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

[Pehle and Pedersen, 2021] Pehle, C. and Pedersen, J. E. (2021). Norse - A deep learning library for spiking neural networks.

[Pfister and Gerstner, 2006] Pfister, J.-P. and Gerstner, W. (2006). Triplets of Spikes in a Model of Spike Timing-Dependent Plasticity. *Journal of Neuroscience*, 26(38):9673–9682.

[Qin et al., 2018] Qin, X., Luo, Y., Tang, N., and Li, G. (2018). DeepEye: An automatic big data visualization framework. *Big Data Mining and Analytics*, 1(1):75–82.

[Querlioz et al., 2011] Querlioz, D., Bichler, O., and Gamrat, C. (2011). Simulation of a memristor-based spiking neural network immune to device variations. In *The 2011 International Joint Conference on Neural Networks*, pages 1775–1781.

[Rathi et al., 2019] Rathi, N., Panda, P., and Roy, K. (2019). STDP-Based Pruning of Connections and Weight Quantization in Spiking Neural Networks for Energy-Efficient Recognition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(4):668–677.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

[Rueckauer et al., 2017] Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification. *Frontiers in Neuroscience*, 11.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

[Sabahi et al., 2016] Sabahi, F., Omair Ahmad, M., and Swamy, M. N. S. (2016). An unsupervised learning based method for content-based image retrieval using hopfield neural network. In *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*, pages 1–5.

[Sandler et al., 2018] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

[Sansen et al., 2017] Sansen, J., Richer, G., Jourde, T., Lalanne, F., Auber, D., and Bourqui, R. (2017). Visual Exploration of Large Multidimensional Data Using Parallel Coordinates on Big Data Infrastructure. *Informatics*, 4(3):21.

[Saunders et al., 2019] Saunders, D. J., Patel, D., Hazan, H., Siegelmann, H. T., and Kozma, R. (2019). Locally connected spiking neural networks for unsupervised feature learning. *Neural Networks*, 119:332–340.

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

[Schwartz et al., 2020] Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12):54–63.

[Senk et al., 2018] Senk, J., Carde, C., Hagen, E., Kuhlen, T. W., Diesmann, M., and Weyers, B. (2018). VIOLA—A Multi-Purpose and Web-Based Visualization Tool for Neuronal-Network Simulation Output. *Frontiers in Neuroinformatics*, 12.

[Sezavar et al., 2019] Sezavar, A., Farsi, H., and Mohamadzadeh, S. (2019). Content-based image retrieval by combining convolutional neural networks and sparse representation. *Multimedia Tools and Applications*, 78(15):20895–20912.

[Shahsavari and Boulet, 2017] Shahsavari, M. and Boulet, P. (2017). Parameter Exploration to Improve Performance of Memristor-Based Neuromorphic Architectures. *IEEE Transactions on Multi-Scale Computing Systems*, 4(4).

[Shi et al., 2019] Shi, Y., Nguyen, L., Oh, S., Liu, X., and Kuzum, D. (2019). A Soft-Pruning Method Applied During Training of Spiking Neural Networks for In-memory Computing Applications. *Frontiers in Neuroscience*, 13.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Son et al., 2018] Son, S., Nah, S., and Lee, K. M. (2018). Clustering Convolutional Kernels to Compress Deep Neural Networks. In *ECCV*.

[Sourikopoulos et al., 2017] Sourikopoulos, I., Hedayat, S., Loyez, C., Danneville, F., Hoel, V., Mercier, E., and Cappy, A. (2017). A 4-fJ/Spike Artificial Neuron in 65 nm CMOS Technology. *Frontiers in Neuroscience*, 11.

[Stimberg et al., 2019] Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife*, 8:e47314.

[Stimberg et al., 2020] Stimberg, M., Goodman, D. F. M., and Nowotny, T. (2020). Brian2GeNN: accelerating spiking neural network simulations with graphics hardware. *Scientific Reports*, 10(1):410.

[Strobelt et al., 2018] Strobelt, H., Gehrmann, S., Pfister, H., and Rush, A. M. (2018). LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):667–676.

[Strukov et al., 2008] Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. (2008). The missing memristor found. *Nature*, 453(7191):80–83.

[Taher and Sammouda, 2011] Taher, F. and Sammouda, R. (2011). Lung cancer detection by using artificial neural network and fuzzy clustering methods. In *2011 IEEE GCC Conference and Exhibition (GCC)*, pages 295–298.

[Tanaka et al., 2019] Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123.

[Thorpe et al., 2001] Thorpe, S., Delorme, A., and Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Networks*, 14(6):715–725.

[Tosi and Yoshimi, 2016] Tosi, Z. and Yoshimi, J. (2016). Simbrain 3.0: A flexible, visually-oriented neural network simulator. *Neural Networks*, 83:1–10.

[Tufte, 2006] Tufte, E. R. (2006). *Beautiful Evidence*. Graphics Press, Cheshire, CT.

[van den Oord et al., 2013] van den Oord, A., Dieleman, S., and Schrauwen, B. (2013). Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

[Wang et al., 2017] Wang, J., Zelenyuk, A., Imre, D., and Mueller, K. (2017). Big Data Management with Incremental K-Means Trees–GPU-Accelerated Construction and Visualization. *Informatics*, 4(3):24.

[Wang et al., 2015] Wang, L., Wang, G., and Alexander, C. A. (2015). Big Data and Visualization: Methods, Challenges and Technology Progress. *Digital Technologies, Digital Technologies*, 1(1):33–38.

[White et al., 1986] White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. (1986). The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 314(1165):1–340.

[Widrow and Hoff, 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. *1960 IRE WESCON Convention Record*, pages 96–104.

[Wu et al., 2018] Wu, J., Chua, Y., Zhang, M., Li, H., and Tan, K. C. (2018). A Spiking Neural Network Framework for Robust Sound Classification. *Frontiers in Neuroscience*, 12.

[Wu et al., 2016] Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. (2016). Quantized Convolutional Neural Networks for Mobile Devices. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4820–4828.

[Wyatt, 2013] Wyatt, D. (2013). *Akka Concurrency*. Artima Incorporation, USA.

[Xing et al., 2019] Xing, F., Yuan, Y., Huo, H., and Fang, T. (2019). Homeostasis-Based CNN-to-SNN Conversion of Inception and Residual Architectures. In *Neural Information Processing*, Lecture Notes in Computer Science, pages 173–184, Cham. Springer International Publishing.

[Yang et al., 2017] Yang, R., Lyu, C., Liu, Y., Zhou, W., Chen, C., Jiang, X., Li, P., Chen, H., Xu, R., and Wang, Y. (2017). Spiking cortical model for geometry invariant and antinoise texture retrieval. In *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 645–650.

[Yu et al., 2018] Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V. I., Han, X., Gao, M., Lin, C.-Y., and Davis, L. S. (2018). Nisp: Pruning networks using neuron importance score propagation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9194–9203.

[Zagoruyko and Komodakis, 2016] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

[Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 818–833. Springer, Cham.

[Zhou et al., 2017] Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. (2017). Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. *arXiv:1702.03044 [cs]*.

[Zhou et al., 2019] Zhou, H., Lan, J., Liu, R., and Yosinski, J. (2019). Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3592–3602.

[Zurowietz and Nattkemper, 2020] Zurowietz, M. and Nattkemper, T. W. (2020). An Interactive Visualization for Feature Localization in Deep Neural Networks. *Frontiers in Artificial Intelligence*, 3.