Université de Lille

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LILLE

Ecole Doctorale Mathématiques, Sciences du Numérique et de leurs Interactions

Département Informatique

Unsupervised STDP-based Feature Learning for Video Analysis with Spiking Neural Networks

---

Apprentissage non supervisé basé sur le STDP pour l'analyse vidéo avec des réseaux neuronaux impulsionnels

---

Mireille EL ASSAL

Soutenu le : 6 février 2024

Devant le jury composé de :

Pr. Jean MARTINET Université Côte d'Azur	**Rapporteur**
Pr. Bernard GIRAU Université de Lorraine	**Rapporteur**
Dr. Ewa KIJAK Université de Rennes	**Examinatrice**
Pr. Laurent GRISONI (Président) Université de Lille	**Examinateur**
Pr. Ioan Marius BILASCO Université de Lille	**Directeur de thèse**
Dr. Pierre TIRILLY Université de Lille	**Co-encadrant de thèse**

Région Hauts-de-France

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

IRCICA

6 3 1

Ce projet bénéficie du soutien de la Région Hauts-de-France

Abstract

A substantial amount of visual data is publicly released on a daily basis, with a significant portion of this data comprising videos. This has rendered video analysis an important endeavor in the computer vision field. Among the various video analysis tasks, Human Action Recognition (HAR) holds significant importance due to its applications across numerous domains, such as surveillance, human-machine interaction, autonomous vehicles, healthcare, security, and military sectors. Deep convolutional neural networks currently stand as the state-of-the-art for HAR. However, these networks come with high computational costs, which limit their usage on energy-constrained devices. They also rely generally on supervised learning, which necessitates vast amounts of labeled data for training. Spiking neural networks (SNNs) are models that process the information in the form of low-energy spikes, instead of regular values. These third generation neural network can overcome the bottlenecks of traditional Artificial Neural Networks (ANNs), when implemented on neuromorphic hardware, such as the widespread energy efficiency problem. However, supervised SNN training methods, like ANN-to-SNN conversion and spiking back-propagation, have their own limitations, such as the requirement of a large amount of labeled data for training. On the other hand, SNNs can leverage unsupervised learning rules, such as the Spike Timing-Dependent Plasticity (STDP) rule, reducing their dependency on labeled data. Despite these advantages, unsupervised SNNs still face challenges in reaching the performance levels of ANNs on complex data. Thus, understanding how an STDP-based SNN can efficiently learn spatio-temporal features becomes crucial in the pursuit of enhancing their performance for human action recognition tasks. This thesis covers knowledge in computer vision and motion modeling, as well as SNN topics. In this manuscript, our main objective is to learn spatio-temporal features and perform video analysis with SNNs in an unsupervised manner using the STDP learning rule. We investigate ways to close the performance gap between SNNs and their non-spiking counterparts when processing spatio-temporal data. Therefore, the first contribution in this manuscript is to study the feature extraction capabilities of an STDP-based Convolutional Spiking Neural Network (CSNN) with different static representations of motion. Motion modeling methods are introduced, categorized into frame-based and shot-based representations, and processed using a 2D CSNN. This produces a clear baseline of the capability of these models to extract spatio-temporal features from different types of motion representations. Our second contribution is to present the first STDP-based

3D CSNN model that can extract spatio-temporal features naturally from videos, without requiring extra motion modeling steps. This model outperforms 2D CSNNs for video analysis, especially with longer videos. Then, in our third contribution, we explore the possibility of reducing the number of parameters of these networks by attempting spiking separated spatial and temporal convolutions (S3TCs). This not only reduces the computational cost of these networks even further, but also potentially reduces the complexity for implementing these networks on neuromorphic hardware. S3TCs outperform 3D CSNNs, and produce a higher spiking activity at the output, which potentially reduces the severity of the spike vanishing problem. Our fourth contribution introduces spiking STDP-based two-stream CSNNs. Two-stream methods are effective spatio-temporal feature extraction methods, with state-of-the-art performance on HAR tasks in the non-spiking domain. Therefore, we use spiking spatial and temporal streams based on CSNNs to obtain spatio-temporal features. This produces an assessment of the capability of these unsupervised STDP-based models to extract effective spatio-temporal features in the spiking domain.

Résumé

Chaque jour, une quantité importante de données visuelles est rendue publique, dont une grande partie est constituée de vidéos. L'analyse vidéo est donc devenue une tâche importante dans le domaine de la vision par ordinateur. Parmi les différentes tâches d'analyse vidéo, la reconnaissance des action humaines (HAR) revêt une importance significative en raison de ses applications dans de nombreux domaines, tels que la surveillance, les interactions homme-machine, les véhicules autonomes, la santé, la sécurité et le secteur militaire. Les réseaux neuronaux convolutionnels profonds constituent actuellement l'état de l'art en matière de reconnaissance des actions humaines, mais leur coût de calcul élevé limite leur utilisation sur les appareils à faible consommation d'énergie. En outre, ils reposent exclusivement sur l'apprentissage supervisé, qui nécessite de grandes quantités de données étiquetées pour leurs formation. Les réseaux neuronaux à impulsions (SNN) sont des modèles qui traitent les informations sous forme d'impulsions à faible énergie, au lieu de valeurs numériques. Ces derniers peuvent surmonter les goulots d'étranglement des réseaux neuronaux artificiels (RNA) traditionnels tels que le problème de l'efficacité énergétique, lorsqu'ils sont mis en œuvre sur du matériel neuromorphique. Toutefois, les méthodes d'apprentissage supervisé des SNN, telles que la conversion ANN-SNN et la rétropropagation à impulsions, ont leurs propres limites, notamment la nécessité d'une grande quantité de données étiquetées pour l'apprentissage. D'autre part, les SNN peuvent tirer parti de règles d'apprentissage non supervisées, telles que la règle de plasticité fonction du temps d'occurrence des impulsions (STDP), ce qui réduit leur dépendance aux données étiquetées. Malgré ces avantages, les SNN non supervisés doivent encore relever des défis pour atteindre les niveaux de performance des ANN sur des données complexes. Ainsi, comprendre comment un SNN basé sur la STDP peut apprendre efficacement les caractéristiques spatio-temporelles devient crucial dans la poursuite de l'amélioration de leur performance pour les tâches d'HAR. Cette thèse couvre les connaissances en vision par ordinateur et en modélisation du mouvement, ainsi que les sujets relatifs aux SNN. Dans cette thèse, notre objectif principal est d'apprendre des caractéristiques spatio-temporelles et effectuer une analyse vidéo avec des SNN de manière non supervisée en utilisant la règle d'apprentissage STDP. Nous étudions les moyens de combler l'écart de performance entre les SNN et leurs homologues non impulsionnels lors du traitement des données spatio-temporelles. Par conséquent, la première contribution de cette thèse est d'étudier les capacités d'extraction de caractéristiques d'un réseau neuronal

convolutif à impulsion (CSNN) basé sur la STDP avec différentes représentations statiques du mouvement. Les méthodes de modélisation du mouvement sont introduites, catégorisées en représentations basées sur les trames ou basées sur les séquences, et traitées à l'aide d'un CSNN 2D. On obtient ainsi un référentiel clair de la capacité de ces modèles à extraire des caractéristiques spatio-temporelles à partir de différents types de représentations du mouvement. Notre deuxième contribution est de présenter le premier modèle CSNN 3D basé sur la STDP qui peut extraire des caractéristiques spatio-temporelles naturellement à partir de vidéos, sans nécessiter d'étapes supplémentaires de modélisation du mouvement. Ce modèle est plus performant que les CSNN 2D pour l'analyse vidéo, en particulier pour les vidéos plus longues. Ensuite, dans notre troisième contribution, nous explorons la possibilité de réduire le nombre de paramètres de ces réseaux en essayant des convolutions spatiales et temporelles séparées (S3TC). Cela permet non seulement de réduire davantage le nombre de paramètres entraînables de ces réseaux, mais aussi de réduire potentiellement la complexité matérielle pour leur mise en œuvre sur du matériel neuromorphique. Les S3TC sont plus performants que les CSNN 3D et produisent une activité plus élevée à la sortie, ce qui réduit potentiellement l'ampleur du problème de disparition des impulsions. Notre quatrième contribution présente des CSNN à deux flux basés sur la STDP. Les méthodes à deux flux sont des méthodes efficaces d'extraction de caractéristiques spatio-temporelles, avec des performances de pointe sur les tâches HAR dans le domaine traditionnel. Par conséquent, nous utilisons des flux spatiaux et temporels à impulsions basés sur des CSNN pour obtenir des caractéristiques spatio-temporelles. Cela permet d'évaluer la capacité de ces modèles non supervisés basés sur les STDP à extraire des caractéristiques spatio-temporelles efficaces dans le domaine des impulsions.

Acknowledgements

I am grateful to the Université de Lille and the Region Hauts-de-France for generously funding my PhD journey. Their support has been instrumental in enabling the successful completion of this endeavor.

My heartfelt gratitude extends to the CRIStAL laboratory, where I embarked on this academic journey. The environment fostered at the laboratory has played an indispensable role in shaping my research pursuits. Furthermore, CRIStAL has generously funded my last months of my PhD, allowing me to conclude my academic pursuit in good conditions.

I extend my sincere appreciation to the IRCICA institution for providing me with a conducive workspace throughout my research journey.

I am grateful to my thesis director, Dr. Ioan Marius Bilasco, and co-director, Dr. Pierre Tirilly, for their guidance, mentorship, and invaluable insights. Their expertise and constant support have been crucial in shaping my research and academic growth. I am particularly thankful to them for their patience and expert solutions to the numerous challenges I encountered along the way. Their wisdom and encouragement have been the cornerstone of this thesis.

I would like to express my sincere gratitude to the members of my thesis committee for their time, expertise, and feedback throughout the evaluation process. Their thoughtful insights and constructive criticism have greatly enriched the quality of this work. I am honored to have had the opportunity to benefit from their guidance and expertise, and I appreciate their commitment to ensuring the academic rigor of my thesis.

A special note of gratitude goes out to my colleagues and the members of Team FOX. The past three years have been enriched by their camaraderie, making this journey both delightful and enlightening.

I am deeply appreciative of my husband, Thibault Desprez, whose unwavering support has been a steady anchor throughout the ups and downs of this journey. His presence has provided the strength to navigate even the most challenging moments, and for that, I am truly grateful.

I extend my thanks to my family, my parents, Antoine El Assal and Tamam Sleem, as well as my sisters Madeline and Marwa, for their unconditional support. Most significantly, my engineer father Antoine, for his stimulating discussions and invaluable insights that have influenced my intellectual endeavors.

My appreciation also goes to my in-laws, Michelle Bourlet and Jean-Marc Desprez, as well as my sister-in-law Dr. Laura Desprez, for their support, kindness, and for teaching me the nuances of French culture and language, which was crucial in team meetings, teaching sessions, and french conferences throughout these three years.

Finally, I would like to thank my dear grandfather Shukri EL ASSAL, who valued education to the highest degree and always wished for me and my sisters to grow up to be strong independent educated women. May his memory live on in everything that we accomplish.

In closing, this journey would not have been possible without the collective support, encouragement, and understanding of all those mentioned above and countless others who have contributed to my growth and success. Your presence has made this academic endeavor both meaningful and memorable. Thank you.

Contents

Part I

Introduction

Chapter 1

Introduction

In recent years, there has been an increasing surge in visual data across various domains, with a large portion constituting video data. A study conducted by Ericsson revealed that videos accounted for approximately 70 percent of all global mobile network traffic in 2023 [1]. Video analysis is an important field within computer vision, especially considering the difficulty for humans to analyze the vast amount of video data generated every day. Video analysis involves understanding the content of videos (e.g., identifying specific persons or objects), and has applications in various domains like smart surveillance, autonomous vehicles, robotics, human-computer interaction, content recommendation, and video editing.

One of the key tasks in video analysis is Human Action Recognition (HAR), which aims to identify human actions in videos. HAR faces challenges such as variations in motion characteristics, similarities between different actions, and changing environmental settings. Other important tasks in video analysis include object tracking, which focuses on tracking targets within videos; event detection, which automatically identifies specific events in videos, not necessarily related to humans; and video segmentation, which divides videos into different segments or regions.

Motion modeling, which is computationally representing the movement of objects in a sequence of images or videos, is particularly essential for HAR, as it helps capture movement patterns specific to action classes. Developing computational models for motion is crucial for HAR applications.

There are numerous different approaches for video analysis, and they can be separated into two main groups: hand-crafted methods and learning-based methods [2]. Hand-crafted methods rely on the careful selection of some interesting features from the video. This careful selection requires human engineering. Learning methods can extract features automatically from videos, unlike hand-crafted methods. They are

often based on deep neural networks made of numerous layers. The current state-of-the-art methods for video analysis in computer vision are deep learning solutions [2], which include models like Convolutional neural networks (CNNs) [3] and two-stream methods [4].

A major drawback of these deep learning solutions, is that they are known for their high computational costs, especially in complex tasks and when dealing with large datasets, raising environmental concerns. The prolonged use of powerful GPUs contributes to pollution, as highlighted by studies like [5, 6]. Moreover, this high computational cost makes it challenging to implement these models on energy-constrained devices, because they can quickly deplete battery life. While there is a move toward more energy-efficient algorithms and edge computing to mitigate these concerns, energy efficiency remains a significant challenge as the demand for computer vision applications continues to grow, and sustainability becomes a key consideration in technology development. In the era of the Internet of Things (IoT), where sensors are integrated into physical devices, IoT applications are being used in various sectors such as smart cities, agriculture, and retail. Consequently, reducing the computational expenses associated with computer vision methods that are often used with IoT applications becomes a pressing need. This reduction can facilitate the implementation of these applications on a wider range of devices.

Another drawback of these deep learning computer vision models is that they heavily rely on large amounts of labeled data for supervised training, which necessitates human involvement, demanding considerable resources and time for collection and annotation. It is essential to highlight the orders of magnitude required for effective training of deep learning computer vision models in well-established tasks, such as image classification or object detection. These orders of magnitude range from tens of thousands to several million labeled instances, as seen in datasets like ImageNet [7].

To address these issues, alternative methods with reduced computational costs are recommended. This led to a shift towards alternative computational models. In this context, Spiking Neural Networks (SNNs) have emerged as a promising approach for video analysis. SNNs present an opportunity to overcome the limitations posed by traditional techniques, opening new avenues for more energy-efficient and environmentally conscious video analysis methods [8]. The main source of inspiration behind SNNs is the brain, which only consumes around 20 W of power and performs extremely complex computations [9]. Therefore, the neurons of these third generation neural networks mimic some features of the spiking neurons in the brain [10]. It is important to note that these networks only mimic some aspects of the brain, and do not perform as well as the brain with complex computations. Unlike traditional

Artificial Neural Networks (ANNs), which have continuous-valued activations, SNNs process the information in the form of impulses that are referred to as spikes. These networks hold the potential to perform video analysis tasks like HAR at a lower cost.

There exist various neuromorphic hardware platforms facilitating the implementation of SNNs, including TrueNorth [11], SpiNNaker [12], BrainScaleS-2 [13], Tianjic [14], and Loihi [15]. While digital chips like Loihi enable STDP training, analog neuromorphic hardware stands out as particularly compelling for STDP applications. They take up significantly less area and consume less energy compared to digital chips [16], and their capability of in-memory computation circumvents the necessity for external memory. Memristor-based analog technology, in particular, holds significant promise for STDP-based learning, due to their inherent ability to emulate synaptic plasticity.

SNNs are energy efficient when implemented on ultra-low power neuromorphic hardware [12, 17], which makes them attractive candidates for applications on devices with limited energy, such as real-time video analysis on edge devices. However, current SNN training methods that achieve high classification rates, like ANN-to-SNN [18] conversion and surrogate gradient-based methods [19], still have certain limitations, like needing large amounts of labeled data, and using global computations that are difficult to implement with ultra-low power neuromorphic hardware. SNNs trained in an unsupervised manner with the Spike Timing-Dependent Plasticity (STDP) rule use local computations that give them the advantage of being easier for efficient neuromorphic hardware implementation. Moreover, the unsupervised nature of these networks can be leveraged to reduce the amount of labeled data needed for training.

STDP-based training also facilitates on-chip learning, which promotes user data privacy. Edge device users are becoming more and more aware of the importance of not sharing their personal information to external servers. An example of this is the voice privacy problem, which has risen from the fact that our voices contain sensitive data about us [20]. The same applies to face recognition applications, that raise biometric security concerns [21]. Also, more and more laws are being issued to protect user data; a famous one is the General Data Protection Regulation (GDPR) issued by the European Union. With the rise of IoT applications, which are finding their way into every home, like with home automation applications, on-chip learning has become a requirement. Therefore, designing technologies that permit on-chip learning and processing in an efficient and effective manner has to be addressed.

The energy efficiency that results from processing information using spikes has made Dynamic Vision Sensor (DVS) [22] cameras the most recent trend for providing spiking input data to SNNs. SNNs trained with event data captured from DVS

cameras are frequently discussed in the existing literature [23–25]. These cameras can be used to create spiking datasets tailored for video analysis tasks, such as the DVS Gesture Dataset [26] which is recorded with a DVS camera for spiking hand gesture recognition. Additionally, SNNs can be supplied with input from micro-Doppler data from radars [27], sensor data from wearable devices [28] and sensors from smartphones [29] [30]. We find that not all applications of SNNs necessitate the use of sensors or DVS cameras. Frame-based applications with SNNs are also essential due to the vast volume of frame-based video data being generated daily. Frame-based video analysis with SNNs becomes imperative for cost-effective analysis.

In this manuscript, we explore the potential of spiking neural networks trained in an unsupervised manner with the Spike Timing-Dependent Plasticity (STDP) learning rule for HAR. We investigate the strengths and limitations of STDP-based SNNs in capturing and exploiting spatio-temporal information. Additionally, we develop novel methodologies and architectures to process video data with minimal energy and supervision if implemented on neuromorphic hardware.

1.1 Motivation

Video analysis is useful in numerous applications, and is expected to gain even more prominence in the upcoming years. Some of these applications can be implemented on energy-constrained devices, which raises energy and computation limitation concerns. For instance, applications used on mobile devices often need to send requests to deep models stored on remote servers because of computation limitations. However, if computations could be performed locally, it would not only reduce computational costs, but also enhance security and user privacy [31].

As previously mentioned, the current state-of-the-art methods used in computer vision are neural networks. These networks present challenges when it comes to energy efficiency. Due to their high energy consumption, these models heavily rely on powerful GPUs for quicker operation. Nevertheless, deploying them on devices with limited energy poses significant challenges [8, 31]. Additionally, concerns about the environmental impact of prolonged GPU usage [5, 6] have surfaced. These concerns illustrate the impact of carbon emissions related to the energy required for training a neural network model on the environment. Furthermore, these models are trained using supervised learning methods, which means that they require large amounts of labeled data for training. This led to the exploration of alternative methods with lower computational costs, such as SNNs that are implementable on a wider range of devices.

By opting for SNNs over traditional Deep Neural Networks (DNNs) in video analysis, the following advantages can be achieved:

- Energy efficiency and local computations: Spiking neural networks are more energy-efficient compared to their DNN counterparts [8]. This is due to the fact that these networks process the information in the form of low energy action potentials, in addition to the sparse nature of spiking activity. This reduces computational requirements and can enable the implementation of low-power neuromorphic hardware for real-time video analysis applications. STDP-based SNNs can conduct their training and perform their computations locally [31]. This could allow the training of the SNN to be done in a wide range of devices, including smartphones, which results in advantages, like promoting more security.

- Unsupervised learning: SNNs can be trained using unsupervised Hebbian learning methods such as STDP [32]. This eliminates the need for a large amount of labeled data, which otherwise would require costly human intervention.

1.2 Major challenges

Video analysis using SNNs is a relatively new research field. Due to its relative novelty, this topic has limited existing research, particularly in the context of unsupervised STDP-based feature learning for video analysis. Consequently, this domain presents several significant challenges that need to be addressed.

- Challenge 1: Video data processing for motion modeling — STDP-based SNNs have demonstrated their capability to extract pertinent spatial information from static images [31]. These models could also be used for video analysis by sequentially processing individual frames. However, this approach would disregard the motion information between successive frames, necessitating the development of methods capable of modeling the motion inherent in videos, similar to what has been done with other computer vision models [33]. This is essential to enable spatio-temporal processing using the currently available 2D SNN architectures. However, as a result of the novelty of STDP-based SNNs for video analysis, the information available concerning the suitable input information and motion modeling for effective processing of videos by these models is insufficient.

- Challenge 2: Network architecture design for motion modeling — Designing effective network architectures for SNNs remains an open challenge. There has been limited work concerning computer vision architectures for video analysis in the spiking domain; some examples are [34], [35], and [36]. However, there is currently not enough research addressing spatio-temporal computer vision architectures for video analysis with unsupervised STDP learning, even though these networks promote user information security, energy efficiency and a lower dependence on labeled data. Designing network architectures for SNNs requires considering the encoding/decoding mechanisms, learning algorithms, and hardware constraints specific to spiking neural models [37].

- Challenge 3: Network complexity and hardware limitations — Video analysis tasks often require modeling complex spatio-temporal relationships. Balancing the network complexity to achieve high accuracy without sacrificing computational efficiency is crucial for practical implementation. This is because hardware connectivity architectures have some resource constraints (i.e., the total number of neurons/core may not exceed 1,024 with Loihi chip [15]). Implementing SNNs for video analysis on resource-constrained hardware platforms can be challenging due to the computational requirements of spatio-temporal data processing. Adapting SNN architectures and training algorithms to anticipate the accommodation of hardware limitations is an important consideration.

- Challenge 4: The spike vanishing problem — It is factual that deeper ANN architectures can learn more complex features [38]; however, the case of multilayer STDP-based SNNs remains an open research challenge [31]. This is attributed to the fact that multiple spikes are aggregated within neurons to trigger a smaller amount of output spikes, resulting in diminished spiking activity in subsequent layers. This loss of spike information is known as the spike vanishing problem.

Overcoming these challenges will lead to more robust, efficient, and accurate spiking neural network-based video analysis systems, which brings us one step closer to unlocking the full potential of SNNs in analyzing visual data.

1.3 Contributions

This section provides a summary of the contributions presented in this manuscript, which address the challenges of video analysis with SNNs discussed in Section 1.2. These contributions consist of:

- Video analysis with 2D Convolutional Spiking Neural Networks (CSNNs) using various static representations of motion: As previously mentioned, 2D STDP-based CSNNs exist for images, so we take advantage of this model and use it for video analysis. The representation of the input data is very important for the learning process of neural networks. Therefore, we take on the task of motion modeling to extract static representations of motion from videos. We divide these representations into two groups, which are the frame-based and the shot-based representations. Then, we analyze the behavior of SNNs with these representations in order to better understand their performance. This contribution presents a comparative analysis of the performance of these spiking networks with different static representations of motion, incorporating novel static representations of motion. This analysis addresses Challenge 1 mentioned in Section 1.2 and sheds light on the efficacy of the introduced representations

- Extending 2D CSNNs into 3D CSNNs: 3D CSNNs are fully spiking solutions that can extract effective spatio-temporal features from videos. To the best of our knowledge, this contribution is the first work that presents 3D CSNNs trained with STDP in an unsupervised manner. We compare the performance of this network to that of a 2D CSNN, and we conclude that the performance of 3D CSNNs is superior. This contribution addresses Challenge 2 mentioned in Section 1.2.

- Spiking Separated Spatial and Temporal Convolutions (S3TC): 3D spiking convolutional kernels can be separated into 2D and 1D kernels that account for the spatial and temporal dimensions, respectively. We compare the performance of 3D and separated spiking kernels and conclude that separated spiking convolutions can improve the performance of spiking models while reducing the number of parameters, promoting potentially easier hardware implementation and lower computational cost. S3TC can also increase the spiking activity at the output of a layer, therefore decreasing the severity of the spike vanishing problem as well. This contribution addresses the Challenges 3 and 4 mentioned in Section 1.2.

- Spiking two-stream methods with STDP-based convolutional SNNs: An effective method for video analysis is the use of two-stream methods [4]. Yet, to the best of our knowledge, this method has not been explored with unsupervised STDP in the spiking domain. Therefore, we transpose two-stream methods to the spiking domain, and we use five different temporal stream configurations to

better understand the performance of these networks. This contribution also addresses Challenge 2 mentioned in Section 1.2.

1.4 Outline

The structure of this manuscript is as follows.

Chapter 2 introduces video analysis, outlining its tasks, challenges, and presenting some widely-used algorithms. It discusses state-of-the-art techniques and methodologies in video analysis, emphasizing the difficulties associated with capturing temporal dynamics.

Chapter 3 offers a comprehensive understanding of essential concepts and principles of SNNs. This chapter also highlights the advantages of SNNs over traditional neural networks and provides an overview of SNNs, along with a concise presentation of spiking neurons, training approaches, and neural coding techniques used for converting pixels into spikes.

Chapter 4 introduces the baseline architecture, and model choices carefully selected from the literature to be used in the contributions.

In Chapter 5, we perform video analysis using STDP-based 2D CSNNs, and evaluate its performance on two benchmark HAR datasets.

Chapter 6 presents our proposed 3D CSNN for video analysis, where we show the advantages of this fully spiking model over the already existing 2D CSNN. We evaluate the performance of 3D CNNs on two benchmark HAR datasets.

In Chapter 7, we improve the performance and reduce the number of parameters of a fully spiking solution by introducing our spiking separated spatial and temporal convolutions model (S3CT). We present comprehensive experimental results that compare between S3TC networks and the previously mentioned 3D CSNNs using a performance evaluation on three benchmark datasets.

Chapter 8 presents spiking two-stream methods, where we transpose the effective two-stream model for video analysis into the spiking domain, and evaluate its performance on four benchmark HAR datasets.

Finally, Chapter 9 concludes the thesis by summarizing the contributions, highlighting the implications of our work, and outlining potential directions for future research.

Part II

Background

The objective of this part is to provide a comprehensive foundation for the contributions, and it is divided into three chapters. It provides an introduction to video analysis in Chapter 2, describes the specificities of SNNs in Chapter 3, and explains the choices that are a basis for our research pursuit in Chapter 4.

In Chapter 2, we provide an overview of video analysis. We describe some of its diverse tasks, focusing particularly on Human Action Recognition (HAR). We then present the challenges inherent to HAR, along with a discussion of pertinent HAR datasets. Following this, we introduce the general components of a video analysis system, accompanied by an introduction to feature extraction methods, split into two principal categories: hand-crafted techniques and deep learning approaches. The chapter is concluded with a summary, where we highlight some of the effective deep learning architectures employed for HAR. Specifically, we find Convolutional Neural Networks (CNNs) and two-stream methods, to be suitable for implementation in the spiking domain within the scope of this study.

In Chapter 3, we present the fundamental concepts and theories related to SNNs. These include an exploration of diverse spiking neuron types, methods to transform data into spikes, training mechanisms of SNNs, and their applications within the domain of video analysis. An understanding of these concepts serves as the foundation for our subsequent chapters, where we take a deeper plunge into the specific aspects of SNNs for video analysis.

Finally, Chapter 4 presents the mechanisms that we have carefully selected from the literature to lay the foundations of our research. These mechanisms encompass, among others, our chosen spiking neuron model, the preferred method for transforming video data into spikes, and the SNN learning rule that we will need to train our network. These choices establish the groundwork for our research.

Chapter 2

Background: Video Analysis

Video analysis is the process of understanding the content within a video, which involves the extraction of valuable information from the video data, including spatio-temporal patterns. This task is fundamental in the field of computer vision and has diverse applications in creating intelligent systems that have a variety of capabilities, including interpreting and responding to human actions. The utility of video analysis is widespread, serving as an important technology in various domains, including smart surveillance, healthcare, and autonomous vehicles.

While the human brain effortlessly processes visual data, video analysis presents a challenge for machines. This is because machines interpret visual information represented as pixel matrices, which can be influenced by various factors like lighting, viewpoint, and a variety of patterns within an object or a class of objects (e.g. shape, color). These factors can significantly alter the values within these matrices, thereby complicating the task of visual analysis.

As the volume of visual data demanding analysis continues to rise, it has become impractical to rely solely on human effort for processing such large quantities of videos. Consequently, there has been a compelling need to develop video analysis algorithms that can be useful for a number of tasks. Some of these tasks are the following:

- Human Action Recognition (HAR): Human action recognition is the task of identifying human actions within a video [39]. This task can be used in many applications, like smart surveillance systems, human-machine interaction, and robotics for characterizing human behavior [40], and has witnessed an increasing interest throughout the years [39–42]. This task presents challenges, some of which are variations in the motion characteristics of actions (e.g., jogging can occur at different speeds), inter-class similarities (e.g., fast jogging might

resemble running, while slow jogging might resemble walking), and environmental settings (e.g., dynamic backgrounds which make it more difficult to localize the human) [39].

- Person/object tracking: Object tracking involves following specific targets within a sequence of video frames, serving various applications such as surveillance [43]. This task can include a series of sub-tasks, like object detection. One approach involves tracking by detection, which employs a detection algorithm to identify target locations and then links them together; this method heavily relies on the accuracy of the object detection algorithm [44]. Other methods leverage the exploitation of temporal variations in video sequences for end-to-end tracking, as seen in recent studies like [45].

- Event detection: Event detection involves the automatic identification of specific salient occurrences in videos that, unlike HAR, do not necessarily involve humans [46]. Some examples of these salient events could be natural disasters [47], social media posts [47], and violent sequences in movies [48].

- Video segmentation: Video segmentation is the task of dividing a video into different segments or regions, often with the goal of identifying and isolating distinct objects, scenes, or actions within the video [49]. Video segmentation also includes motion segmentation, which is the task of identifying the independently moving regions in a video and separating them from each other, or from the background motion [50], and background modeling for foreground detection [51].

We are specifically interested in modeling motion for HAR because of its diverse range of applications across various domains [40], [52]. HAR requires capturing the movement patterns present in a video, because human actions are characterized by distinct motion sequences. This is referred to as motion modeling, and it requires developing computational models that describe how objects in the video move over time, like the one modeled in Figure 2.1.

In the subsequent sections of this chapter, we will discuss the challenges associated with HAR, and explore some currently available HAR datasets in Section 2.1. Following this, we describe the elements of a standard video analysis system in Section 2.2. This system can employ either hand-crafted feature extraction techniques or feature learning methods, which are explained in Sections 2.3 and 2.4 respectively. Finally, we provide in Section 2.5 a summary and a conclusion, which bring to focus the pertinent models that inspire our contributions in subsequent chapters.

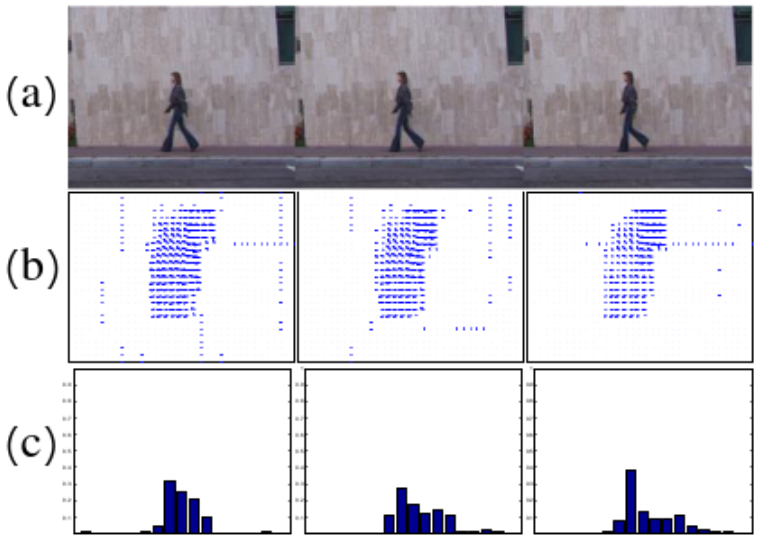


Figure 2.1: Motion modeling with optical flow, (a) a person walking, (b) the optical flow vectors that result from the motion, (c) the histogram of oriented optical flow feature trajectories. Figure from [53].

2.1 Challenges & datasets in HAR

Understanding the challenges of HAR datasets is crucial for developing accurate models that can analyze human activities in various environments. We begin by explaining what constitutes a video. A video is a sequence of frames captured one after the other in a timely manner, which creates the illusion of continuous motion. Each frame is a matrix of pixels, where each pixel has an intensity. Therefore, a video can be seen as a 4D tensor of size $l_w \times l_h \times l_c \times l_{td}$ where l_w and l_h are the width and height of the frames, l_c is their number of channels (e.g. $l_c = 1$ for grayscale frames, $l_c = 3$ for RGB frames), and l_{td} is the temporal depth of the tensor i.e., the number of frames in the video sample.

There are certain visual data challenges that are shared by both image and video data, which can affect the interpretability of the data. These challenges arise from the fact that visual data is perceived by computers as pixel values, which can vary depending on factors like the acquisition device, viewpoint, and illumination. Changing the viewpoint, such as capturing the same scene from different angles, results in significant changes in pixel values. Similarly, variations in illumination, such as capturing an image at night versus during the day, further contribute to the fluctuations in these values.

Another layer of complexity is added to these challenges with video analysis, as variations in visual data can occur not only between individual frames but also across different frames within the video sequence (e.g., viewpoint and illumination can change within the same video).

In the context of action classification, HAR presents specific additional challenges. As the number of distinct classes or class samples increases, the potential for overlap between classes in pixel space also rises. Consequently, accurately distinguishing and classifying similar actions becomes more difficult. HAR challenges also encompass motion variations, scenery changes, diverse settings, and interpersonal differences among video subjects. For a HAR algorithm to be considered effective, it must be robust to these variations and deliver consistent performance even in the presence of noise.

These challenges are reflected in the existing HAR datasets. There are numerous publicly available frame-based video datasets used for HAR, and several of them are detailed in Table 2.1 along with their challenges in Table 2.2. Some of them are early and simple datasets, like the KTH [54] and Weizmann [55] datasets. The KTH dataset [54] contains 600 videos of 25 subjects, performing 6 actions in 4 scenarios, and the Weizmann dataset [55] contains 90 videos of 9 subjects performing 10 actions. Other datasets are slightly more complex. For example, the Inria XMAS dataset [56] is made up of 11 actions and 1148 sequences with different actors, cameras, and viewpoints, which brings additional complexity. Another challenging dataset is the UCF sports action dataset [57]. This dataset contains 150 videos of 10 actions, and is made up of realistic videos, and has numerous challenges such as variations in settings, the presence of other individuals, objects, and animals within the videos, and varying viewpoints. Some other well-known action recognition datasets are the UCF101 [58] dataset, which includes a wide range of human actions, the HMDB51 [59] dataset, which consists of videos of 51 action classes from different sources, the Hollywood-2 [60] dataset, which contains 12 actions and 10 classes of scenes distributed over 3669 video clips from movies, and the Kinetics [61] dataset, which is a large-scale action recognition dataset comprising over 500,000 video clips covering 600 human action classes with at least 600 video clips for each action class. Among the wide variety of frame-based video datasets used for HAR, the challenges associated with the eight prominent datasets introduced here are summarized in Table 2.2. These challenges encompass a range of factors, such as setting variations (e.g., indoor and outdoor recordings), the presence of similar actions (e.g., jogging and running), different viewpoints captured from various angles, subject occlusion caused by objects or subjects partially obstructing the action, the complexity of managing large-scale datasets with numerous distinct classes, and the complexity of dealing with realistic

videos sourced from environments outside the controlled settings of a laboratory, including movie scenes and sports events. The progress in dataset complexity reflects the advancements of HAR systems, and the available datasets offer increasing challenges to evaluate new paradigms in HAR with different levels of maturity.

Summary	# Actions	# Videos	# Subjects	Resolution	Year
KTH [54]	6	600	25	160×120	2004
Weizmann [55]	10	90	9	180×140	2005
IXMAS [56]	11	1148	10	48×64	2006
UCF Sports [57]	10	150	-	-	2009
Hollywood-2 [60]	12	3669	-	-	2009
HMDB51 [59]	51	6849	-	320×240	2011
UCF101 [58]	101	13,320	-	320×240	2012
Kinetics [61]	600	500,000	-	-	2017

Table 2.1: The characteristics of human action recognition datasets.

Challenges	setting variations	similar actions	different viewpoints	subject occlusion	large-scale dataset	realistic videos
KTH [54]	✓	✓				
Weizmann [55]		✓				
IXMAS [56]		✓	✓			
UCF Sports [57]	✓	✓	✓			✓
Hollywood-2 [60]	✓	✓	✓	✓	✓	✓
HMDB51 [59]	✓	✓	✓	✓	✓	✓
UCF101 [58]	✓	✓	✓	✓	✓	✓
Kinetics [61]	✓	✓	✓	✓	✓	✓

Table 2.2: The challenges of human action recognition datasets.

2.2 A video analysis system for HAR

A video analysis system consists of several essential blocks shown in Figure 2.2: preprocessing, feature extraction, and feature classification. The input to the system

is a video, which undergoes pre-processing as an optional step to prepare it for subsequent analysis tasks. Pre-processing may involve tasks like noise removal or reduction, data normalization, and problematic artifact removal. Pre-processing can also involve methods that extract motion from video data to highlight motion-related patterns, which facilitates the extraction of relevant temporal information crucial for subsequent analysis tasks.

An example of this is pre-processing with optical flow, which involves analyzing the motion between two successive frames in an image sequence or video. Optical flow provides motion vectors, from one frame to the next. Two main types of optical flow are sparse optical flow, which focuses on specific interest points [62], and dense optical flow, which considers motion for all points in the frame [63].

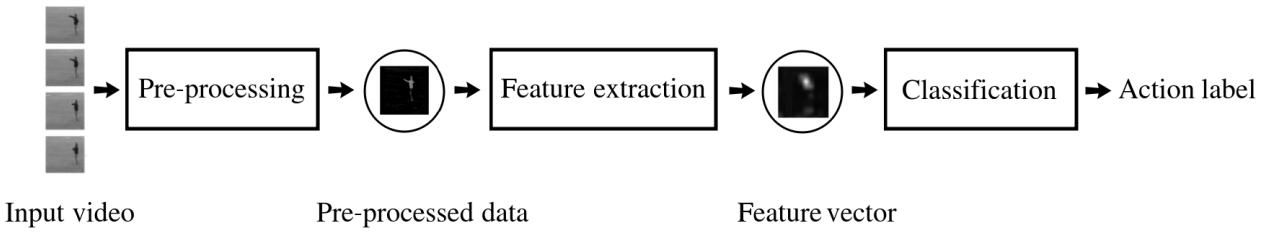


Figure 2.2: General stages of video analysis.

Unlike the pre-processing step, which is optional, the feature extraction step is essential for video analysis. The relevant extracted features are distinctive patterns, resulting in output feature vectors that serve as inputs to a classifier. These features mitigate the challenges of HAR by building representations that are invariant to a number of factors like illumination, translation, viewpoint, etc. The classifier assigns appropriate labels to the data, which allows the distinguishing of different activities within the video. This classifier is typically a supervised machine learning model.

Feature extraction methods can vary in the literature, and include hand-crafted as well as deep learning approaches. The choice of a particular approach depends on various factors, including the complexity of the task at hand, the available computational resources, and the specific constraints of the application, such as the need for robust performance in the presence of noisy or limited data.

2.3 Hand-crafted methods

Hand-crafted methods require human expertise in designing the feature extraction process, and they rely on the manual engineering of the features needed for a partic-

ular task. Feature extraction with hand-crafted methods is generally separated into multiple stages, as shown in Figure 2.3. First, a feature detector is required in order to detect regions of interest, which are key points where unique characteristics or patterns are present in the visual data. Detecting them makes it possible to extract relevant information. Then, descriptors are used to represent the visual information from this region of interest in a quantifiable manner. The output feature vectors are then transformed into a single feature vector, facilitating the classification process by providing an informative representation of the essential visual cues. Then, a classifier is used to perform the action classification.

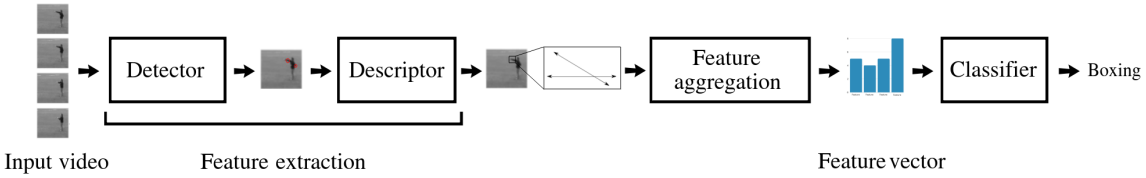


Figure 2.3: Stages of video analysis with hand-crafted methods.

2.3.1 Detectors

There are numerous options for the detection of points or regions of interest that are used for video analysis. These detectors are usually inspired from spatial detectors used for images, and have been extended for video analysis. Some of them are sparse and detect only points of interest, which have significant variations in space and time [64], while others are dense [65], which sample the appearance and motion variations in all regions over the entire video. Dense sampling of local features has proven better effectiveness in classification tasks [65, 66]. Some pertinent hand-crafted detectors, varying in sparsity and density, are listed here:

- Space-Time Interest Points (STIPs): Space-time interest points [64] are an extension of spatial interest points, and they represent the points of significant local variations in both space and time. The detection of these points is based on Harris and Forstner detectors, where the Harris detector [67] is a corner detector. With STIP, spatio-temporal corners are detected with the 3D Harris detector [64], where a spatio-temporal corner is an image region containing a spatial corner whose velocity vector is changing [68]. However, this method is not effective when the motion is subtle [68].

- 3D Scale-Invariant Feature Transform (SIFT-3D): The Scale Invariant Feature Transform (SIFT) computes scale-invariant coordinates of local patterns

(blobs, corners) from image data. A Difference-of-Gaussians operator is applied to an image to obtain SIFT interest points. This operation involves creating copies of the same image smoothed with Gaussian kernels of different standard deviations, calculating the differences between these images, and then selecting the extremas (peaks) of these values as SIFT interest points. These interest points are scale invariant as they are local extremas in the scale space, allowing their scales to be normalized or adjusted. Additionally, they are rotation invariant, as a principal orientation is calculated for each SIFT interest point [69]. SIFT-3D is a spatio-temporal extension of SIFT, and it is also made up of a detector and a descriptor. It detects key points and extracts descriptors from different scales and orientations in a 3D space [70]. SIFT-3D can be employed to identify distinctive points in videos that can be used to track motion and describe the appearance of objects or body parts, which is useful for human action recognition.

- Enhanced Speeded-Up Robust Features (ESURF): ESURF [71] is an enhanced version of the Speeded-Up Robust Features (SURF) [72] algorithm. It detects key points and generates descriptors that are robust to changes in scale, rotation, and lighting. The detection of interest points is done using the determinant of the Hessian matrix. This process involves convolving the image with a series of box filters at different scales, and then computing the Hessian matrix using the derivatives of these convolutions. The determinant of the Hessian matrix is used to measure the local changes around a point, and interest points are detected at locations where the determinant of the Hessian matrix is maximized. ESURF retains the core principles of SURF, but it incorporates further optimizations and adjustments to enhance its performance in various applications, and can be applied to extract features from video frames.

- Trajectories: Trajectories permit tracking a given spatial point over time, which captures the local motion information from videos [65]. The Kanade–Lucas–Tomasi (KLT) tracker [73] is a tracker which includes a detector, and can be used to extract trajectories. It is slightly denser than STIP [65] because it only detects features in the spatial dimension. This method detects points of interest that are identified using a corner detection algorithm like the Harris corner detector or the Shi-Tomasi corner detector [74]. Then, the optical flow at these points of interest between two consecutive frames is computed using the Lucas-Kanade algorithm, and these motion vectors are linked from frame to frame in order to track corners. The KLT tracker is also paired with the Harris3D interest points in some work [75]. This tracker is also used in [76] to capture long-duration

trajectories.

- Dense Trajectories: Dense trajectories were introduced in [65]. They sample feature points on a dense grid in each frame and track them using a dense optical flow algorithm, instead of the sparse one used in the KLT tracker. This provides a good coverage of the video features and smoothness constraints, which allows relatively robust tracking of fast and irregular motion patterns.

- Cuboids: Cuboids were introduced in [68]. With cuboids, some subtle movements that are ignored by the STIP method can be detected. The detection of motion with cuboids requires first the definition of a response function, similarly to other detection methods. This response function is calculated by applying separable linear filters, which are a 2D Gaussian smoothing kernel, applied only along the spatial dimensions, and 1D Gabor filters applied in the temporal dimension [68]. This response function detects interest points when it reaches a local maxima.

2.3.2 Descriptors

With hand-crafted methods, the detectors mentioned above require descriptors, which can represent and quantify the visual information around the detected key point. Some descriptors that can be used for spatio-temporal feature extraction extend the work done for spatial feature extraction, like 3D Histograms of Oriented Gradients (HOG-3D) [77], 3D Scale-Invariant Feature Transform (SIFT-3D) [70] and Enhanced Speeded-Up Robust Features (ESURF) [71].

- 3D Histogram of Oriented Gradients (HOG3D): Histogram of oriented gradients (HOG) [78] generates a compact representation of features within an image. Gradients represent changes in pixel values and can highlight edges and textures. This technique involves computing gradient vectors at each position around the key-point. Gradient vectors are then organized into a histogram, with the histogram's bins representing specific orientation ranges. The magnitudes are aggregated within these bins, creating a condensed representation of the pattern around the key-point. HOG3D is the extension of HOG which calculates the distribution of gradients in a 3D space. In the context of human action recognition, videos are considered as spatio-temporal volumes and the key HOG concepts can be generalized to 3D [77]. HOG3D captures not only the appearance but also the motion characteristics in the volume around a key-point. Similar to HOG, HOG3D computes gradient vectors in

spatial and temporal dimensions. These computed gradients are then organized into a three-dimensional histogram, with the bins representing specific spatio-temporal orientation ranges [77].

- Histogram of Optical Flow (HOF): HOF [79] is an optical flow-based descriptor capable of extracting motion features from sequences of images. HOF is a histogram of orientations of the optical flow vectors [80]. This descriptor has the advantage of being insensitive to variations in lighting and clutter.

  HOF can highlight motion patterns and dynamics in videos. It is effective for recognizing actions that involve specific motion directions or patterns [79].

- Motion Boundary Histograms (MBH): MBH [65] is another descriptor that has been introduced for action recognition. This descriptor computes the derivatives separately for the horizontal and vertical components of the optical flow. Motion boundaries represent the locations where changes in motion occur. With this descriptor, information like camera motion is removed by disregarding constant motion, while information like motion boundaries are kept. Motion boundaries provide a robust descriptor; this makes MBH an effective descriptor that significantly outperforms other hand-crafted descriptors for HAR [65].

After the extraction of descriptors, the set of feature vectors needs to be transformed into a single vector that can be fed into a machine learning model. A typical approach that can be used is the bag-of-features approach, also known as the bag-of-visual-words representation [81]. This method involves the construction of a visual vocabulary containing a set of learned visual words that characterize the distinctive visual elements present within the data. These visual words encapsulate the dominant patterns found in the descriptors. By employing the bag-of-words approach, the data dimensionality is further reduced, resulting in a simplified yet informative representation of the visual content. Then this data is introduced it into a classifier [65].

2.4 Learning methods

Unlike hand-crafted methods that require expert engineering to design appropriate features for recognition, deep learning methods offer the capability to learn features directly and automatically from visual data. Deep learning models use trainable feature extractors followed by trainable classifiers, allowing for end-to-end learning.

In this manuscript, the focus on neural networks is driven by their exceptional ability to automatically capture complex patterns and hierarchical representations from complex data, making them well-suited for action recognition tasks.

Artificial neural networks (ANNs) are loosely inspired by the biological neural networks in the human brain. They consist of layers of artificial neurons. These neurons process and transmit information through synapses, where synapses carry strengths referred to as weights. A neuron is a computational unit that takes inputs from one or multiple sources. It applies a linear combination of these inputs using weights and then applies an activation function to produce an output, as shown in Figure 2.4 and represented in Equation 2.1:

$$y = f_a \left(\sum_{i=1}^{\text{nb}} W_i X_i + b \right) \tag{2.1}$$

where y is the output value, f_a is the activation function, nb is the number of inputs, W is the weight vector of the synapse, X is the input vector, and b is a bias value that allows the model to fit the data better.

The activation function of a neuron is used to introduce non-linearity to the model, enabling it to learn complex patterns and relationships in the data. The most famous examples are the sigmoid and Rectified Linear Unit (ReLU) activation functions. The sigmoid function squashes its input to a range between 0 and 1, while the ReLU function outputs the input directly if it is positive and zero otherwise.

A typical ANN is made up of multiple layers, usually an input and an output layer, with one or more hidden layers in between. The network is considered as Fully Connected (FC) if each neuron in one layer is connected to all the neurons in the subsequent layer, as shown in Figure 2.5. This figure shows a Feed-Forward (FF) neural network that transmits the information in a forward direction from the input layer through one or more hidden layers, ultimately reaching the output layer. The weights associated with the connections between neurons are adjusted during the training process, used to optimize the network's performance.

A fundamental algorithm used in training ANNs, known as back-propagation [82], which is short for "back-propagation of errors". It is a supervised learning rule that optimizes the network's parameters by calculating the error through a comparison of the predicted output with the ground truth and minimizing a cost function that measures the discrepancy between them. Ground truth refers to the known correct output corresponding to the provided input. The back-propagation algorithm uses a forward pass, where each neuron in the network computes the output to send to the next layer, and a backward pass where the error computed at the output of the

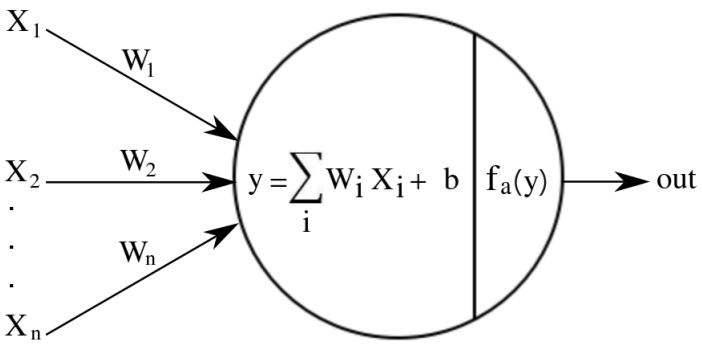


Figure 2.4: A neuron.

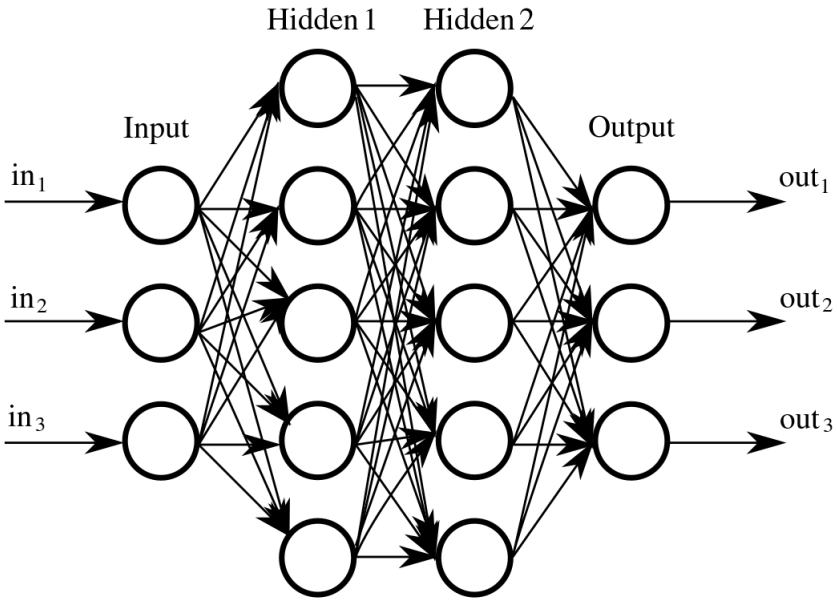


Figure 2.5: A neural network.

network is back-propagated to optimize the network weights starting from the output layer up till the input layer. The weight updates are determined by the gradient of the error with respect to the network's weights. This process is repeated multiple times in order to reduce the error and try to reach a global minimum of the cost function.

The feature learning process involves updating weights over multiple epochs, where an epoch represents one complete iteration through the entire training dataset, to learn significant patterns and representations from the input data. As the network progresses through the epochs, these learned features gradually align with the patterns present in the input data, enabling the network to refine the representations that are used for classification. When local minimum is reached, the network is considered as trained, and the parameters of the network are preserved. It is important to note that while some neurons serve as feature learners and are trained as feature extractors, others function as classifiers.

The architecture of the network plays a role in feature extraction, where the arrangement and connections of the layers of the network influence the nature of the learned features. For instance, deeper architectures can often extract more complex features [83], while shallower architectures tend to capture simpler and more fundamental patterns, as shown in Figure 2.6.

Video analysis requires the network to take the temporal relationship between the video frames into account. Many deep architectures have been designed for video classification tasks. These models have been successful in learning and understanding patterns, dynamics, and temporal dependencies in sequences of data. In this section, we explore three prevalent topologies often used in HAR: Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and two-stream networks.

2.4.1 Convolutional neural networks

Convolutional Neural Networks (CNNs) are feed forward networks that have proven to be effective with visual data analysis like image classification or segmentation [38, 85], as well as video analysis [86]. They operate based on the principle of data locality, which involves using data elements from nearby locations to extract relevant features and patterns among neighboring pixels. A discrete 2D convolution computes the amount of overlap between an input image A and a convolutional kernel B denoted as $A * B$ [31] as shown in Figure 2.7, and represented by Equation 2.2:

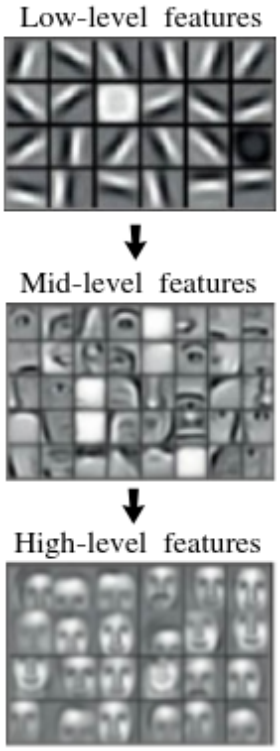


Figure 2.6: A CNN extracts more sophisticated, higher-level features as it goes deeper [84].

$$S(x,y) = \sum_{m=-\left\lfloor \frac{f_w}{2} \right\rfloor}^{\left\lfloor \frac{f_w}{2} \right\rfloor} \sum_{n=-\left\lfloor \frac{f_h}{2} \right\rfloor}^{\left\lfloor \frac{f_h}{2} \right\rfloor} A(x+m, y+n) \cdot B\left(\left\lfloor \frac{f_w}{2} \right\rfloor + m, \left\lfloor \frac{f_h}{2} \right\rfloor + n \right) \tag{2.2}$$

where A is an input image of size $l_w \times l_h$, B is a 2D convolutional kernel of size $f_w \times f_h$, and (x, y) is a location in the input image.

The convolution process begins by placing the filter, also known as a convolutional kernel, over the top-left corner of the input data, initiating the sliding process with a step size l_{stride} across the entire input. This kernel is positioned in a way that aligns its elements with the corresponding pixel values within the chosen region. For each pixel, a dot product is computed between the values of the kernel and the image pixel values it covers. This multiplication and summation process results in a single numerical value, which is the response to the kernel, as explained in Figure 2.7. Repeating this procedure across the entire input data yields an output feature map. Each value in this map corresponds to the result of the dot product operation performed at a specific location. In essence, each position in the output feature map represents the degree of similarity between the pattern of the kernel and the patterns present in the input data in the corresponding region. Padding pixels represented by l_{pad} can be added at the border of the image to ensure that the filter adequately covers the boundaries, preventing information loss and maintaining spatial dimensions in the output feature map.

A CNN contains one or more convolutional layers, and each layer has multiple filters which correspond to neurons. Neurons corresponding to the same filter are duplicated at all locations and share weights, allowing them to collectively cover the entire visual field and mimic the sliding over the input of the convolution operation.

Performing a convolution operation with f_k 2D kernels will result in a feature map vector made up of a set of f_k feature maps according to Equations 2.3, 2.4, 2.5 from [31]:

$$l_w(i) = \frac{l_w(i-1) + 2l_{\text{pad}} - f_w}{l_{\text{stride}}} + 1 \tag{2.3}$$

$$l_h(i) = \frac{l_h(i-1) + 2l_{\text{pad}} - f_h}{l_{\text{stride}}} + 1 \tag{2.4}$$

$$l_c(i) = f_k \tag{2.5}$$

where l_w and l_h are the new width and height of the output feature maps, l_c is the channel dimension of the output feature map vector, l_{pad} represents the padding, and l_{stride} represents the stride.

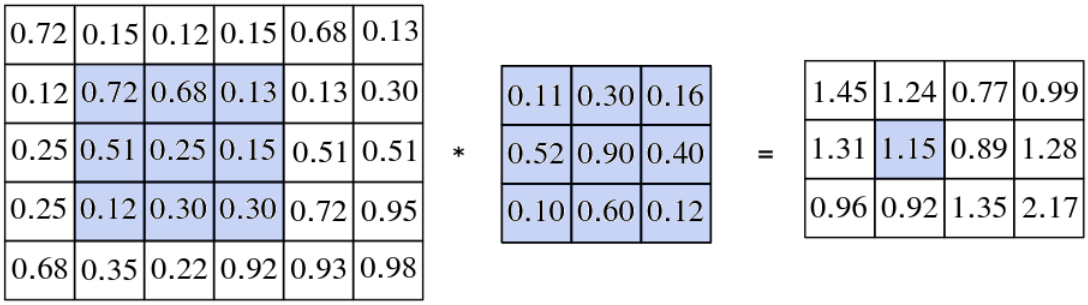


Figure 2.7: A 2D convolution operation.

Thus, through this convolutional operation, we compute responses from the input data that measure the presence of certain patterns or characteristics, enabling the subsequent layers of the neural network to learn more complex patterns by composing the patterns detected previously. CNNs take advantage of the nature of visual data, where neighboring pixels have meaningful relationships. These relationships can be characterized by metrics such as pixel adjacency and connectivity [87].

Compared to fully connected networks, the number of trainable parameters is reduced due to the sharing of parameters, which involves the application of the same set of weights across the entire input sample. This fewer number of parameters renders CNNs more computationally efficient and enables them to handle larger inputs. Additionally, it serves to limit the effects of overfitting, where the network closely learns the training data and struggles to generalize. Moreover, this parameter sharing property allows CNNs to be trained effectively with a substantial yet practical amount of training data.

2D convolutions are widely used for image analysis, effectively capturing spatial patterns and features within individual frames. However, when applied directly to video data, 2D convolutions exhibit limitations in effectively processing temporal information, as they do not inherently account for the sequential changes between frames. Consequently, this approach fails to capture the dynamic changes and temporal dependencies present in video data, resulting in incomplete representations of the underlying motion patterns.

To overcome these limitations, 3D convolutions [88] were introduced as a natural extension of 2D convolutions, incorporating an additional temporal dimension to

handle spatio-temporal information. 3D convolutions enable the kernel to slide across three dimensions, including width, height, and time, in contrast to 2D convolutions that slide in two dimensions only, as shown in Figure 2.8. 3D convolutions facilitate the extraction of both spatial and temporal features simultaneously. This enhancement allows 3D CNNs to effectively capture motion patterns in video data, leading to better representations for tasks such as action recognition. It is considered as the most natural way to deal with spatio-temporal information by some authors [89].

In summary, neurons in convolution layers are connected to a subset of neurons in the previous layer. A convolutional layer is made up of a set of f_k convolutional kernels that perform the convolution operation on the input visual data in order to extract meaningful patterns. These kernels can be 2D of size $f_w \times f_h$ or 3D of size $f_w \times f_h \times f_{td}$ depending on the nature of the visual input (i.e., image or video) and considering grayscale data where the number of channels is one, where f_w, f_h, and f_{td} represent the filter width, height and temporal depth.

A $3D$ convolution operation with f_k filters will also result in a set of f_k feature maps according to Equations 2.3, 2.4, 2.5, but these feature maps also have an extra dimension l_{td}, for time represented by Equation 2.6:

$$l_{td}(i) = \frac{l_{td}(i-1) + 2l_{\text{pad}} - f_{td}}{l_{\text{stride}}} + 1 \tag{2.6}$$

CNNs are primarily composed of sequential convolutional layers, with pooling layers in between to downsample the data, as shown in Figure 2.9. These pooling layers also improve spatial invariance, and add more non-linearity [31]. As with FC layers, the stacking of convolutional layers enables the learning of hierarchical patterns with increasing complexity. Each convolutional layer processes the input data to extract simple visual features in the initial layers and progressively learns more complex visual patterns as the data passes through deeper layers. This hierarchical pattern learning is made possible by the ability of the convolutional layers to capture local patterns in the initial layers and combine them to recognize more complex structures in subsequent layers. Then, the feature maps at the output of the last layer can be linearized into a one-dimensional array, which can subsequently be introduced into FC layers that act as a classifier.

As mentioned previously, a 3D convolution has the advantage of dealing explicitly with temporal data using its third dimension dedicated to time. However, these models also have several issues. For instance, they have more trainable parameters than 2D models, which consequently increases their computational cost and makes the optimization of these parameters more difficult. As a result, these models can be energy-consuming, which renders running them on devices with limited energy very

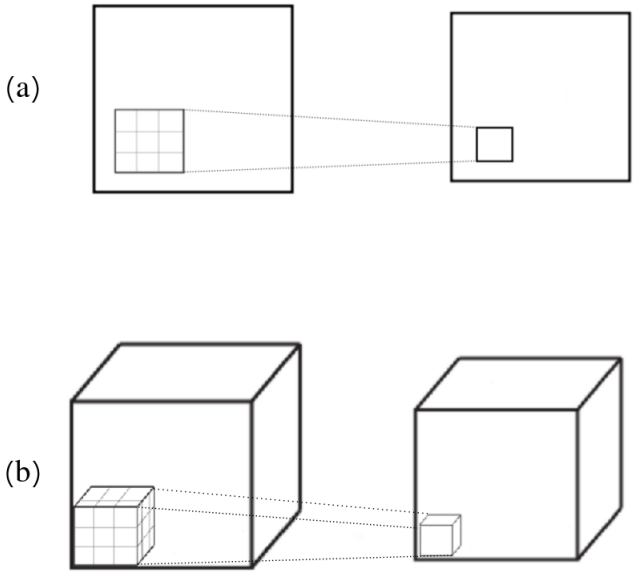


Figure 2.8: A (a) 2D convolution and (b) 3D convolution.

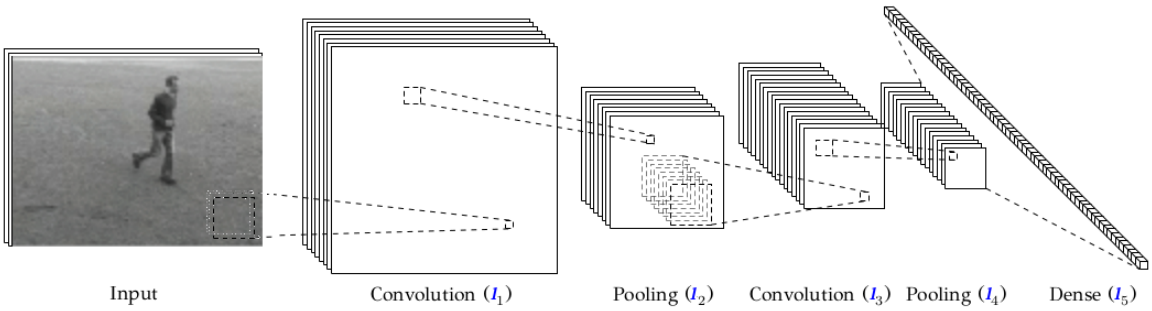


Figure 2.9: Example of a CNN topology from [31].

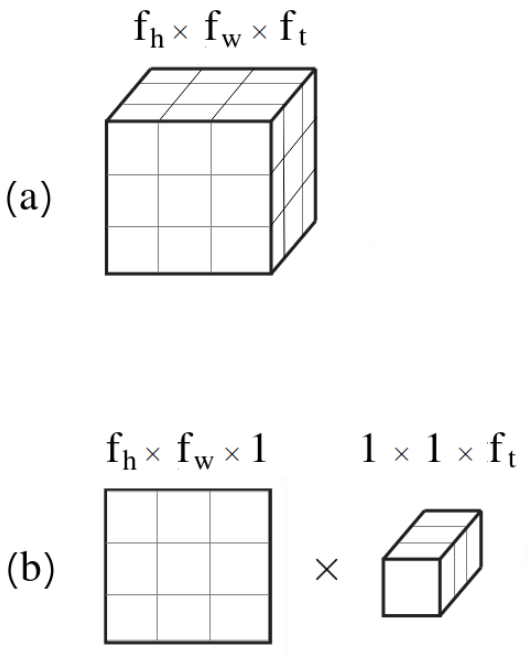


Figure 2.10: Factorizing (a) a 3D convolutional kernel into (b) separated spatial (2D) and temporal (1D) convolutions.

challenging. This, in addition to the environmental concerns of running powerful GPUs for extended periods of time [5,6], initiated the search for alternative methods that have a lower computational cost, and are easier to train, among which are separable convolutions [90]. With separable convolutions, a single large convolution kernel is separated into two or more smaller kernels, as shown in Figure 2.10: the complex 3D kernel of Figure 2.10(a), which can slide along both spatial and temporal dimensions, is factorized into a 2D kernel in Figure 2.10(b), that slides along the spatial dimensions only to extract spatial data first, followed by a 1D kernel which slides along the temporal dimension, and extracts temporal features, such as motion.

Separable convolutions adopted in networks like MobileNets [90] and Xception [91] have succeeded in decreasing the number of parameters of these networks while maintaining their performance. Moreover, gains in accuracy have been recorded when factorizing a 3D convolution into a 2D spatial convolution and a 1D temporal convolution [92]. In [92], the authors attribute this gain in accuracy to additional nonlinearities added by the separated convolutions compared to using a 3D convolution. They argue that these nonlinearities render the model capable of representing more complex functions. They also add that 2D and 1D filters are easier to optimize

than 3D filters.

In this section, we have observed that CNNs can process spatial and temporal features via 2D and 3D convolutions. Other models, such as Recurrent Neural Networks (RNNs), are also capable of extracting these features, and handling sequential data to capture temporal dependencies. We will further examine this topic in the next section.

2.4.2 Recurrent neural networks

Recurrent Neural Networks (RNNs) are networks with cyclic connections, which can process sequential data by maintaining hidden states that retain information over time. This can make them suitable for human action recognition tasks, because human movements are encoded in a sequence of successive samples in time, and RNNs can exploit the temporal correlations in the data [93]. An RNN cell has a feedback connection so that it can keep track of its previous output, and is made up of input, output and hidden nodes. A single RNN node is illustrated in Figure 2.11. This node takes two inputs, which are the current input x_t and the preceding hidden state h_{t-1}, and updates its hidden state h_t and output y_t as indicated by Equations 2.7 and 2.8 from [93], respectively:

$$h_t = \mathrm{f_a}(W_{xh}X_t + W_{hh}h_{t-1} + b_h) \tag{2.7}$$

$$y_t = \mathrm{f_a}(W_{hy}h_t + b_y) \tag{2.8}$$

where X is the input vector, W_{xh}, W_{hh}, and W_{hy} are the weights for the input-to-hidden connection, hidden-to-hidden recurrent connection, and hidden-to-output connection, respectively; b_h and b_y are the biases for the hidden and output states, respectively, and $\mathrm{f_a}$ is the activation function associated with each node.

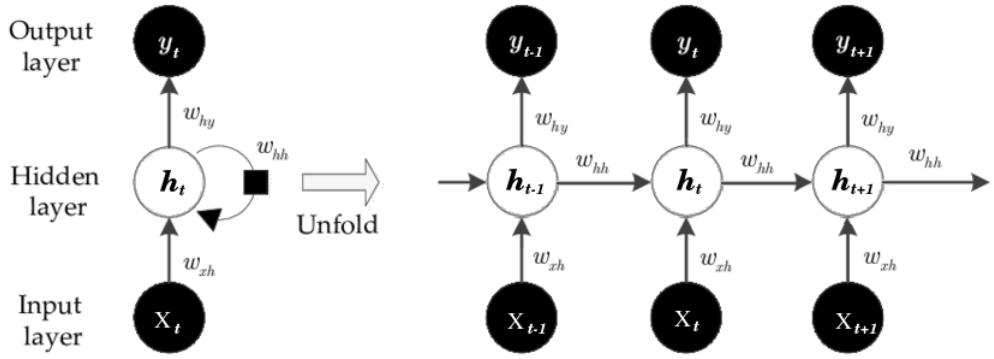


Figure 2.11: An RNN cell from [94]

.

Backpropagation Through Time (BPTT) [95] is a supervised training algorithm adapted from the classical backpropagation algorithm for RNNs. This algorithm involves unfolding the node over time, as illustrated in Figure 2.11, treating it as a deep feedforward network, and applying standard backpropagation to update the weights of the model. In standard RNNs, the gradients of the loss function become extremely small as the network undergoes backpropagation through time. As a result, the updates to the parameters of the model become negligible, leading to halted learning.

Long Short-Term Memory (LSTM) networks [96] are a variant of RNNs that have been constructed to address the vanishing gradient problem with vanilla RNNs. LSTMs have a more complex architecture than vanilla RNNs, incorporating memory cells, input gates, output gates, and forget gates. These gates permit the LSTM to store, retrieve, and forget information over long sequences [93, 97].

Gated Recurrent Units (GRUs) were introduced as a less complex alternative to LSTMs that can also capture long-term temporal dependencies [98, 99]. These networks have less gating mechanisms than LSTMs, and they aim to strike a balance between model complexity and performance.

Reservoir computing [100] is another common recurrent architecture that consists of an input layer, a reservoir, and a readout (output) layer. The reservoir is made up of a group of randomly interconnected neurons, which enable the transformation of the input into a higher-dimensional space. In the output layer, a linear classifier is adequate for distinguishing the various states of the reservoir [101].

In order to use RNNs for HAR, some work suggests to pre-process the HAR videos with pose estimation algorithms capable of extracting the landmarks of the

person [102]. Then, these landmarks are sent into an LSTM. However, these methods are not ideal when there is some object in the context, because pose estimation algorithms usually only focus on the landmarks of the human and discard spatial information. Therefore, other methods have emerged, like combining RNNs and CNNs. One method to conserve the temporal component in a sequence of images is training CNNs on this sequence of static images and extract local features from the input sequence, then use recurrent neural networks (RNNs) in order to capture their temporal dependencies [103]. These models can be used to detect patterns in HAR videos.

2.4.3 Two-stream networks

Two-stream networks are able to extract different types of features from the same video sample, as shown in Figure 2.12. These networks are inspired by the visual processing system in the human brain, which has a ventral and a dorsal stream that extract different visual information. The ventral stream is called the "vision-for-perception" pathway, while the other is the "vision-for-action" pathway [104]. Inspired by this, two-stream methods extract appearance information by using a designated spatial stream, while the motion information is extracted by a temporal stream. Then, the information extracted from both streams is concatenated and sent into a classifier.

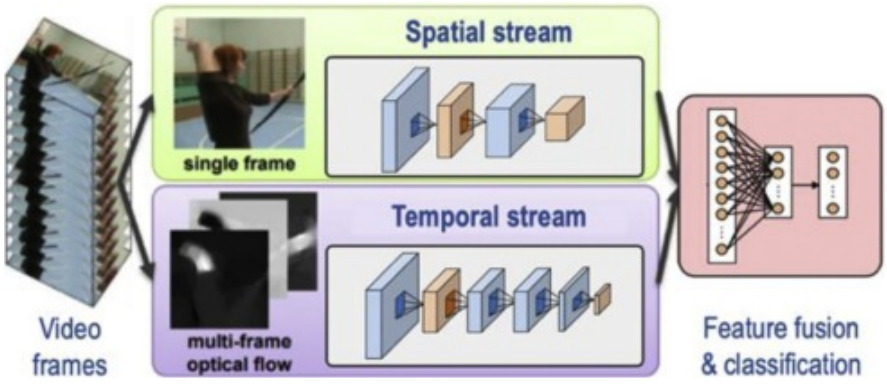


Figure 2.12: General two-stream architecture from [105].

These networks have shown good performance with video analysis, even when using limited training data [4]. The spatial stream in these models typically extracts

spatial information from a single video frame [4]. However, recent literature indicates a shift towards favoring the processing of multiple frames through the spatial stream to better represent spatial information [106–109], as opposed to relying solely on a single frame. For example, in [109], the authors select 10 RGB frames and apply average pooling on these frames to merge them into one frame. This frame is then processed by the spatial stream to extract spatial features. The preference for using multiple frames instead of relying solely on a single one in the spatial stream is because using only one frame might not capture the intricate spatial context essential for accurate classification. By extracting multiple spatial feature maps through the spatial stream, the resulting output encapsulates the rich complexity of spatial information, enhancing the capacity of the model for effective classification.

Some authors use RNNs, LSTMs and 3D CNNs as streams in two-stream network architectures [110–112]. They show a benefit in using spatio-temporal streams that already extract spatio-temporal features. Some work also uses pose or skeleton information, like in [111], where the authors show that two-stream methods that are based on 3D CNNs can be improved by adding a stream for pose estimation. In the case of action recognition, actions can manifest significant variations in appearance and motion across different instances of a video. By modeling spatial and temporal aspects separately, two-stream networks can better handle these variations, which leads to better performance.

2.5 Summary and conclusion

Deep learning methods, particularly CNNs, have demonstrated superior performance in human action recognition (HAR) compared to traditional hand-crafted features [113]. While there is ongoing debate about the comparative efficacy of CNNs, RNNs, and LSTMs, literature suggests that CNNs tend to outperform the latter, especially in scenarios involving longer sequences of data [114].

	KTH	Weizmann	UCF101
3D CNN [3]	90.20%	-	-
3D CNN [115]	94.90%	97.2%	-
3D CNN + LSTM [116]	94.39%	-	-
RNN + HOG3D [117]	93.28%	-	-
Two-Stream [118]	**98.83%**	**99.10%**	-
3D CNN [88]	-	-	85.20%
DB-LSTM [119]	-	-	91.21%
Two-Stream [4]	-	-	88.0%
Two-Stream [120]	-	-	**92.5%**

Table 2.3: Performance of deep learning methods on benchmark datasets.

The comparative analysis presented in Table 2.3 underscores the effectiveness of these deep learning methods in human action recognition tasks. Notably, 3D CNNs emerge as models capable of extracting spatio-temporal features by incorporating the time dimension, avoiding the added complexity of gates, feedback loops, and memory associated with RNNs and LSTMs. Additionally, Two-stream methods, as depicted in Table 2.3, showcase state-of-the-art performance on certain HAR datasets. These methods offer a compelling approach by integrating spatial and temporal information, making them effective spatio-temporal architectures for addressing the challenges of HAR. Their capability to capture both appearance and motion aspects inherent in video data contributes to an overall improvement in recognition accuracy. Nevertheless, it is essential to acknowledge that all these methods possess inherent limitations associated with ANNs. Some of these limitations were introduced in Chapter 1 and are reiterated here:

- Vanishing or exploding gradients: Gradients used for weight updates may become extremely large or small during supervised training, leading to slow convergence or instability [121], [122].
- Limited memory and computational resources: Traditional ANNs demand substantial computational resources and memory, posing challenges for large-scale implementation [123], [124].
- Dependency on large labeled datasets: ANNs require large labeled datasets for effective training, necessitating costly human intervention for labeling.

- Environmental concerns: The use of powerful GPUs for extended periods raises environmental concerns related to carbon footprint and energy consumption [5], [6].

These limitations have prompted a search for alternative methods with lower computational costs and reduced susceptibility to these issues. Consequently, transferring some of these methods to the spiking domain could mitigate the severity of these limitations.

Chapter 3

Background: Spiking Neural Networks

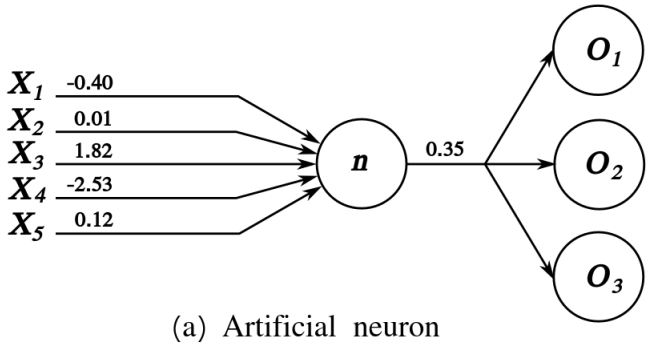
Spiking neural networks (SNNs) are third generation neural networks that process the information in the form of low-energy spikes instead of continuous numerical values like with non-spiking ANNs, as shown in Figure 3.1. A regular neuron processes information by applying a linear combination of its inputs, each modulated by weights that act as magnifiers, amplifying or attenuating the input based on their respective strengths. Then, the neuron applies an activation function to produce an output, as explained in Chapter 2 on page 26.

In contrast, SNNs are event-driven with sparse activation, which means that the neurons of these networks process events spread through time, called spikes, and that only a subset of neurons in the network are active at any given time. This prevents unnecessary calculations when there is no significant input, and thus increases computational efficiency. These networks also allow parallelizing their training and inference processes, hence speeding them up. The advantages of SNNs over ANNs are efficient information processing when implemented on dedicated hardware, and event-based asynchronous computations [8, 125, 126].

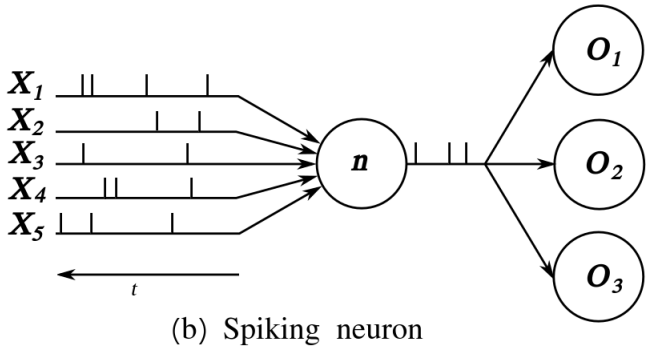
3.1 Principles of spiking neurons

3.1.1 Spiking neuron models

SNNs use spiking neurons to communicate with spike trains, where a spike represents an electrical pulse also known as an action potential. These neurons have a membrane potential and a firing threshold. They receive and accumulate spikes over time, which



(a) Artificial neuron



(b) Spiking neuron

Figure 3.1: A comparison between an artificial neuron and a spiking neuron. The structure is common between ANNs and SNNs, However, (a) an artificial neuron receives numerical values to compute an output numerical value, while (b) a spiking neuron receives sparse events called spikes and generates a sequence of output spikes.

incrementally raises their membrane potential, until their threshold is crossed. This prompts the neuron to fire an output spike, and its membrane potential goes back to its resting state, as shown in Figure 3.2. Then, the neuron enters a refractory period during which its membrane potential is unresponsive to incoming spikes.

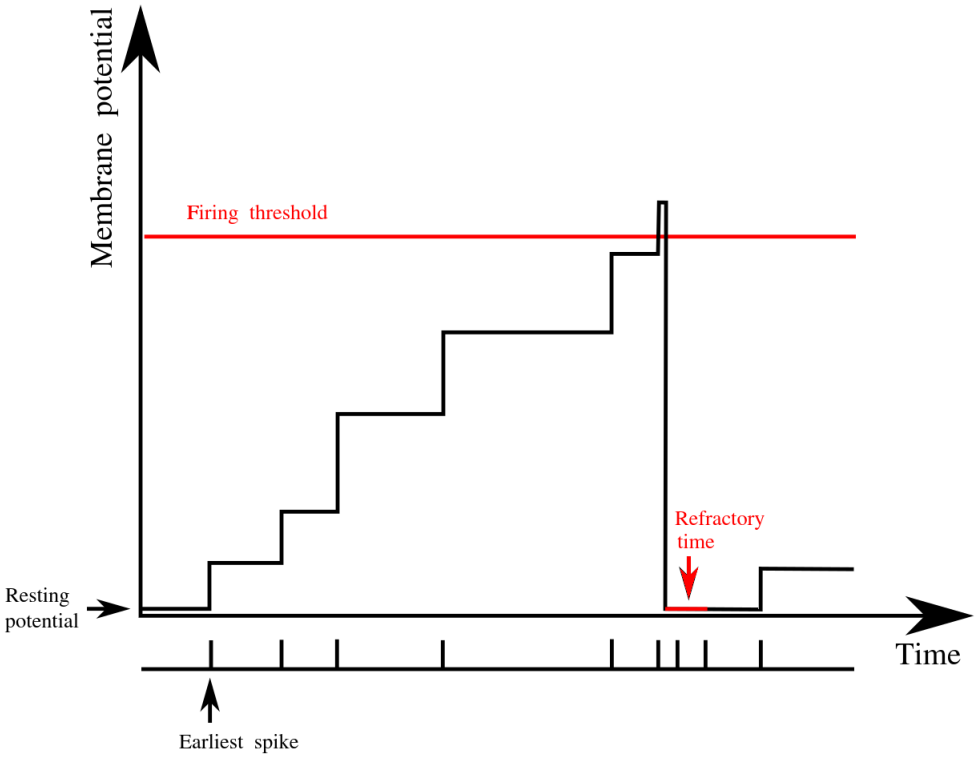


Figure 3.2: The evolution of a basic spiking neuron's membrane potential in response to an incoming spike train.

There are many types of spiking neuron models that vary in biological plausibility and complexity. The simpler neurons are easier to implement with neuromorphic hardware, while the more complex models promote more biological integrity. One of the earliest and most important models in computational neuroscience [127] is the Hodgkin-Huxley model [128], which is relatively complex and extremely expensive to implement [127]. To understand this model, it is necessary to look at how a spike is generated in a biological neuron. The membrane of a biological neuron has sodium ions (Na^+) and potassium ions (K^+) on both its interior and exterior surfaces. These ions carry positive charges, and typically, there is a greater concentration of positive charge on the exterior, maintaining the cell membrane's resting state at a

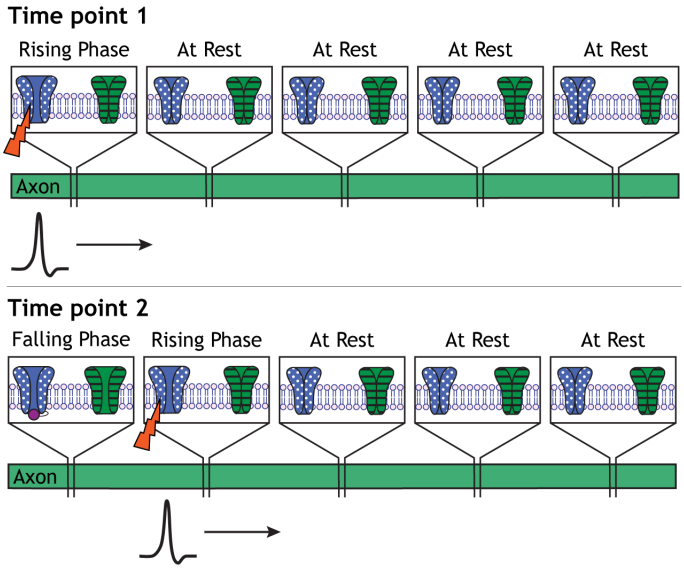


Figure 3.3: The action potential moves down an axon like a wave, opening voltage-gated channels along the length of the axon. The dotted, blue channels represent voltage-gated sodium channels; the striped, green channels represent voltage-gated potassium channels. From [129], this figure is licensed under a Creative Commons Attribution Non-Commercial Share-Alike (CC-BY-NC-SA) 4.0 International License.

resting potential of -70mv. These ions traverse potassium and sodium pumps, which further increase the positive charge on the exterior surface of the membrane, by allowing more sodium ions to exit than potassium ions to enter. After a certain sodium construction on the outside, sodium gates open as a result of the potential increase, and permit the excess sodium that is accumulated on the exterior, to flow into the interior of the cell, generating an electrical potential difference. This, in turn, increases the interior potential and triggers the opening of potassium gates, enabling positive potassium ions to flow outward, restoring the resting potential. This electrical potential difference propagates along the axon of the cell through the action of sodium and potassium gates that are along the axon. This allows the propagation of an action potential, as shown in Figure 3.3.

This Hodgkin-Huxley model is represented using four equations which reproduce a neurons' membrane current (I) in Equation 3.1, sodium current I_{Na} in Equation 3.2, potassium current I_{K} in Equation 3.3, and leakage current I_l in Equation 3.4:

$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + \bar{g}_{\mathrm{K}} n^4 (V_m - V_K) + \bar{g}_{\mathrm{Na}} m^3 h (V_m - V_{\mathrm{Na}}) + \bar{g}_l (V_m - V_l) \tag{3.1}$$

$$I_{\mathrm{Na}}(t) = \bar{g}_{\mathrm{Na}} m(V_m)^3 h(V_m)(V_m - E_{\mathrm{Na}}) \tag{3.2}$$

$$I_{\mathrm{K}}(t) = \bar{g}_{\mathrm{K}} n(V_m)^4 (V_m - E_{\mathrm{K}}) \tag{3.3}$$

$$I_l = g_l (V_m - V_l) \tag{3.4}$$

where V_m is the membrane potential, $\bar{g}_{\mathrm{K}}$ is the maximal potassium conductance, n is the activation gating variable for potassium, responsible for opening the potassium gate, V_{K} is the potassium potential, $\bar{g}_{\mathrm{Na}}$ is the maximal sodium conductance, m is the activation gating variable for sodium, responsible for opening the sodium gate, V_{Na} is the sodium potential, V_l is the leakage potential that account for the natural permeability of the membrane to ions, I_l is the leakage current, C_m is the membrane capacitance, h is the inactivation gating variable responsible for closing the sodium gate, E_{Na} is the sodium reversal potential, E_{K} is the potassium reversal potential, and g_l is the maximal leakage conductance. These equations explain how the excitability of a neuron is affected by the changes in ion channel conductance and gating variables.

The FitzHugh-Nagumo model [130] is a simplified version of the Hodgkin-Huxley model, which provides insights into neuronal excitability by modeling the activation and deactivation dynamics of a spiking neuron using nullclines (i.e., a curve or set of points in a phase space). An external stimulus I_{ext} affects the membrane potential V_m and the linear recovery variable y. The recovery variable reflects the return to the baseline state of specific ion channels within a neuron membrane, y represents sodium channel reactivation and potassium channel deactivation. When I_{ext} crosses a certain threshold, spikes are generated, then V_m and y go back to their resting values. This model is represented by the following equations of a cubic nullcline in Equation 3.5 and a linear nullcline in Equation 3.6:

$$\frac{dV_m}{dt} = V_m - \frac{V_m^3}{3} - y + RI_{\mathrm{ext}} \tag{3.5}$$

$$\tau \frac{dy}{dt} = V_m + a - by. \tag{3.6}$$

where V_m is the membrane potential, y is a linear recovery variable associated with a recovery process, I_{ext} is the external input current, R is a constant (typically set

to 0.1) that represents resistance, τ is a small positive constant that determines the timescale separation between V_m and y, and a and b are constants that control the nullclines of the system.

The Hindmarsh-Rose models [131] are designed to capture a broader range of neuronal behaviors and can replicate more realistic spiking patterns than the FitzHugh-Nagumo model. This model is described by the following equations, which reproduce the membrane potential of a neuron V_m in Equation 3.7, the variable y in Equation 3.8, associated with the rate of transport of sodium and potassium ions and referred to as the spiking variable, and the slow adaptation variable z in Equation 3.9, which models the slow adaptive process, corresponding to an adaptation current that increases with every spike, leading to a decrease in the firing rate:

$$\frac{dV_m}{dt} = y - aV_m^3 + bV_m^2 - z + I_{\text{ext}}, \tag{3.7}$$

$$\frac{dy}{dt} = c - dV_m^2 - y, \tag{3.8}$$

$$\frac{dz}{dt} = r[s(V_m - V_r) - z] \tag{3.9}$$

where V_m represents the membrane potential, a and b are constants affecting the cubic and quadratic terms, and I_{ext} is an external current, c and d are constants that influence the dynamics of the y variable, r is a positive constant, s is the sigmoid function often used to capture the slow adaptation, and V_r is the resting potential.

Izhikevich's model [132] is a simple and computationally efficient spiking neuron model that can replicate a wide range of spiking patterns found in biological neurons. This model is represented by the following Equations 3.10 and 3.11:

$$\frac{dV_m}{dt} = 0.04V_m^2 + 5V_m + 140 - y + I_{\text{ext}}, \tag{3.10}$$

$$\frac{dy}{dt} = a(bV_m - y) \tag{3.11}$$

where V_m represents the membrane potential, y is a recovery variable, I_{ext} is an external input current, a is a timescale constant that determines the rate of recovery, and b influences the sensitivity of the recovery variable to changes in the membrane potential.

he integrate-and-fire (IF) model [133] is the simplest neuron model to date, and despite its simplicity, it is effective in performing visual pattern learning tasks [31]. This model integrates incoming spikes and fires an action potential when its threshold is reached, and it is represented by the following equations:

$$\begin{aligned} c_m V_m(t) &= V_r + \sum_{i \in \mathcal{E}} W_i f_s(t - t_i) \\ V_m(t) &\leftarrow V_r \text{ when } V_m(t) \geq V_{\text{th}}(t) \end{aligned} \tag{3.12}$$

$$f_s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.13}$$

where c_m is the membrane capacitance, V_m is the membrane potential, $\mathcal{E}$ represents the set of incoming spikes, W_i is the weight of the synapse carrying the i-th spike, t_i represents the timestamp of the i-th spike, and f_s is the kernel of spikes.

A slightly more complex model is the leaky integrate-and-fire (LIF) model [134], which includes a leak term to gradually decrease the membrane potential over time in the absence of input spikes, as shown in Figure 3.4. This leak simulates the passive ionic currents across the neuron membrane, and makes this neuron model closer to biology. This model is represented by the following equation from [31]:

$$\begin{aligned} \tau_{leak} \frac{dV_m}{dt} &= [V_m(t) - V_r] + r_m \sum_{i \in \mathcal{E}} W_i f_s(t - t_i) \\ V_m(t) &\leftarrow V_r \text{ when } V_m(t) \geq V_{\text{th}}(t) \end{aligned} \tag{3.14}$$

where $\tau_{leak} = r_m c_m$ the time constant that controls the shape of the leak, r_m is the membrane resistance, c_m is the membrane capacitance, and f_s is the kernel of spikes (Equation 3.13).

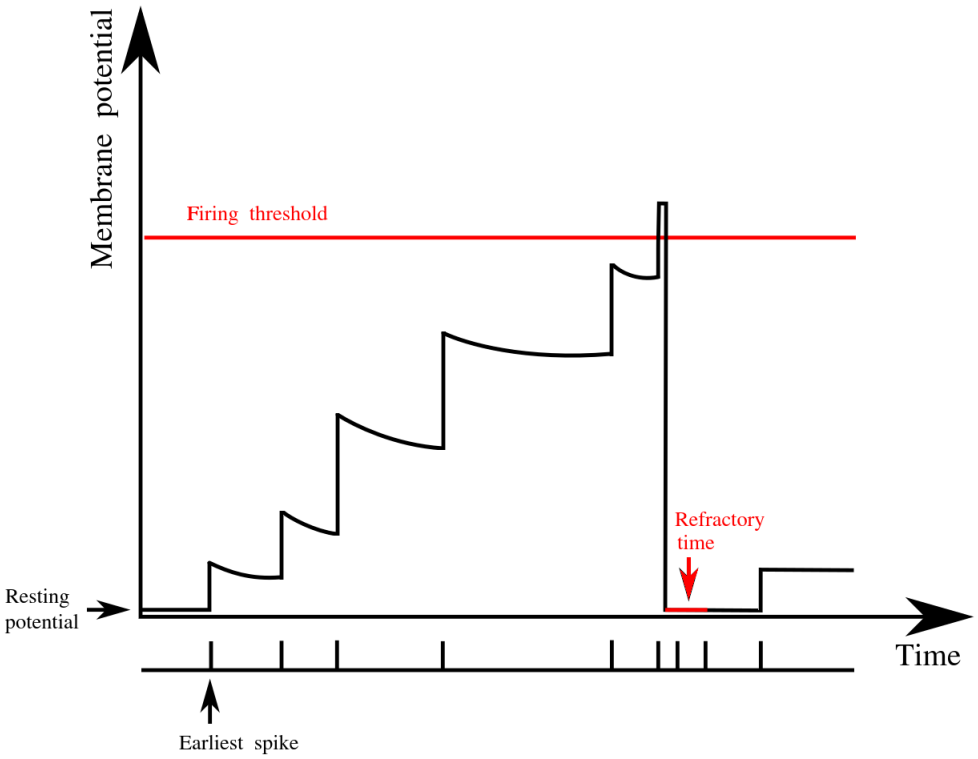


Figure 3.4: The evolution of a LIF spiking neuron's membrane potential in response to an incoming spike train.

A slightly more biologically plausible neuron model is the Current-based Leaky Integrate-and-Fire (CUBA-LIF) model. This spiking neuron accounts for the temporal dynamics of the post-synaptic current. Similarly to LIF, it shows a decay in the membrane potential in the absence of spiking activity. However, it also has another exponentially decaying term, which is the synaptic current. The dynamics of this neuron for pre-synaptic neurons j connected to a post-synaptic neuron i are presented in Equations 3.15 and 3.16 from [135]:

$$\tau_s \frac{dI_i(t)}{dt} = -I_i(t) + \sum_j W_{ij} \sum_{t_{j,m}<t} \delta(t - t_{j,m}) \tag{3.15}$$

$$\tau_m \frac{dV_{m_i}(t)}{dt} = -(V_{m_i}(t) - V_r) + I_i(t) \tag{3.16}$$

where τ_m is the time constant for the membrane potential $V_m(t)$, τ_s is the time constant for the synaptic current $I(t)$, $\delta(t)$ is the Dirac function representing the

kernel of spikes, $t_{j,m<t}$ is the firing time of the $m-th$ spike generated by the $j-th$ presynaptic neuron, synaptic weights between neurons i and j are denoted as W_{ij}, and V_r is the resting potential.

3.1.2 Neural coding

Spiking neural networks process data in the form of spikes. Therefore, there is a need to transform input data into spike trains. In this manuscript, we focus on independent input data presented to the spiking neurons one sample at a time. Each sample is shown to a neuron for a given time frame considered the exposition period also known as $t_{\text{exposition}}$ in [31]. All neural codings can be generalized to a continuous input setting by adjusting the time frame accordingly. There are many neural coding schemes that permit transforming input values into spikes. We list the major ones below:

- Rate coding: This method represents input intensity $f_{\text{in}}(x)$ through Poisson spike trains obtained using a Poisson process: $f_{\text{in}}(x) = t \sim \text{Poisson}(x)$, where t is the spike train and the frequency of spikes corresponds to the input value $x \in [0, 1]$.

  The rate coding scheme is constrained by a prolonged processing period and slow information transmission [136], which is due to the fact that obtaining a precise assessment of the encoded values requires the neurons to integrate numerous spikes across each input connection. Therefore, this leads to the incorporation of latency within the network, as each neuron requires adequate time to assimilate several spikes before generating new ones.

- Temporal coding: This method represents input intensities by the timestamps of spikes. There are multiple variants of temporal coding, which include:

  - Rank order coding [137], which implicates that the order of spike occurrences represents the encoded input value. This coding scheme relies on the ranking of spike timings across multiple neurons to encode information. The more strongly activated neurons tend to fire earlier than weakly activated ones, and the resulting spike pattern encodes the information about the input sample [138]. This neural coding can be represented in a way that depicts its impact directly on the neuron, as done in [138]. Alternatively, this encoding can also be represented in a manner independent of the neuron model, as demonstrated in Equation 3.17:

$$f_{\text{in}}(x_i) = r(x_i) \times \frac{1}{n} \tag{3.17}$$

where x_i is an input value, $X = (x_1, x_2, \ldots, x_n)$ is the set of incoming connections, and $r(x_i)$ is the rank of x_i in X.

- Latency coding [139]: This coding represents the value using the firing timestamp of the neuron. This signifies that earlier timestamps are assigned to inputs with higher intensities, while inputs with lower intensities are assigned later timestamps. This encoding is represented by Equation 3.18 from [31]:

$$f_{\text{in}}(x) = (1.0 - x) \times t_{\text{exposition}} \tag{3.18}$$

where f_{in} is the resulting spike timestamp, $x \in [0, 1]$ is the input value, and $t_{\text{exposition}}$ represents the presentation duration for one sample. This equation keeps the spike timestamp inversely proportional to the value of the input. As the input intensity increases, the spike timestamp decreases, resulting in earlier spiking. This reflects the concept of latency coding, where the stronger stimuli result in faster neural responses.

A variant of latency coding is Time-To-First Spike (TTFS) coding [140], which is similar in spirit to latency coding, but produces spikes using a different, possibly non-linear function of the input [136].

- Phase coding: This coding involves the encoding of information by the relative timing of neurons' spikes with respect to the phase of an ongoing oscillatory rhythm. This phenomenon has been observed in the hippocampus and olfactory system [136, 141]. In [142], a simple phase coding scheme is presented by converting input values into their binary representations, where a bit of value "1" signals the presence of a spike in a given phase. For encoding pixel intensity ranging from 0 to 255, one can use 8 phases. Each bit position corresponds to a phase. So, for an input value, the number of spikes generated equals the number of ones in its binary representation and each spike is assigned to the phase corresponding to its originating a bit in the binary representation. Over a period, the weight carried by a spike changes periodically according to the phase, as shown in Figure 3.5, and this change is defined by Equation 3.19 from [136].

$$W_s(t) = 2^{-[(1+\text{mod}(t-1,\phi))]} \tag{3.19}$$

where $W_s(t)$ represents the spike weight at time t, ϕ represents the phase, and mod represents the modulus function.

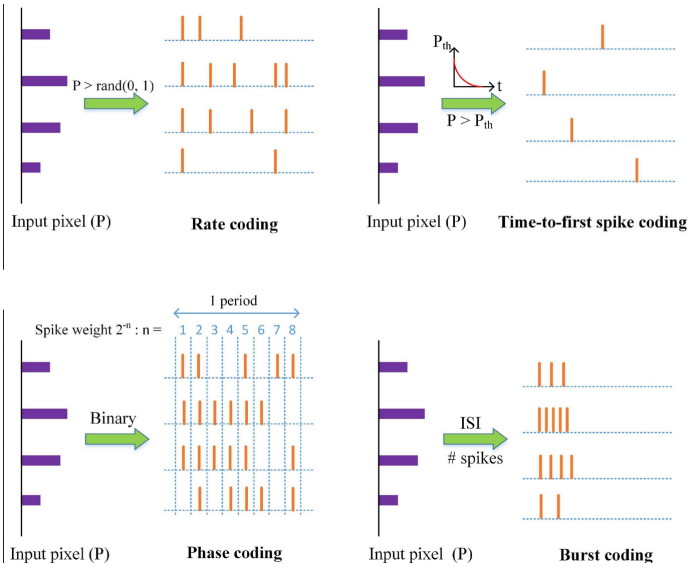


Figure 3.5: Different types of neural coding from [136]. This figure shows how an input pixel is encoded into spikes in four different ways. Rate coding, time-to-first-spike coding, phase coding, and burst coding.

- Burst coding: This coding scheme enables robust and effective information processing by sending a burst of spikes instead of one spike [143]. Burst coding carries the information in the number of spikes (N_s) and the Inter-Spike Interval (ISI) in the burst [136]. The ISI represents the distance between the spikes in the burst. This scheme is represented by the Equations 3.20 and 3.21 from [136].

$$N_s(x) = \lceil N_{\max} x \rceil \tag{3.20}$$

$$\mathrm{ISI}(x) = \begin{cases} \lceil -(T_{\max} - T_{\min})x + T_{\max} \rceil & \text{if } N_s > 1 \\ T_{\max} & \text{otherwise} \end{cases} \tag{3.21}$$

where $N_{\max}$ is the maximum number of spikes, $x \in [0, 1]$ is the input value, $\lceil . \rceil$ is the ceiling function, and $T_{\min}$ and $T_{\max}$ are the minimum and maximum intervals, respectively. The ISI is confined in $[T_{\min}, T_{\max}]$.

3.2 Training spiking neural networks

The spiking neurons of SNNs are connected via synapses, which are fundamental units for information processing with neural networks. Synaptic plasticity is the

ability of synapses to change their strength (weight) according to the spiking activity patterns and the employed learning rules. SNNs incorporate the concept of synaptic plasticity in order to emulate the learning functionalities of biological neurons. The weights of the synapses directly affect the firing ability of the neuron: a low synaptic weight reduces the contribution of the input in prompting the post-synaptic neuron to fire when a spike is received, while a high synaptic weight increases this contribution. SNNs are trained by modulating synaptic weights and other parameters, like neuron thresholds, in order to learn significant patterns from input data. The most common SNN training models featured in the literature are based on one of the following three methods:

- ANN-to-SNN conversion [144], which trains a regular ANN and then transfers the learned weights to an SNN,
- Spiking back-propagation [145, 146], which is a supervised technique for direct SNN training,
- Hebbian learning [8, 32, 147], which is a biologically plausible learning rule similar to the learning in the brain. This type of rule enables the use of unsupervised learning with SNNs.

3.2.1 ANN to SNN conversion

ANN-to-SNN conversion is the process of converting an ANN into a SNN while conserving the network's functionality and performance. This method requires training a regular ANN using back-propagation or any other training algorithm. Then the trained weights of this ANN are converted to be used with a spiking neural network. This conversion usually consists of scaling the weights and adjusting them to the suitable range for the chosen spiking neuron model [144, 148].

Although competitive results are reached with this method, it still requires training a regular ANN first. Therefore, this method has many of the same bottlenecks that SNNs should help avoid. These methods also usually require complicated post-conversion processes like weight normalization and threshold balancing [149].

3.2.2 Spiking back-propagation

Spiking back-propagation is a supervised training rule specifically designed for training SNNs. It is a variant of the back-propagation algorithm [82]. This algorithm optimizes the weights of the network by computing the error, which is based on the

cost function of the desired output and the predicted output. This error is back-propagated, starting from the output layer and moving backward through the layers of the network. This classical back-propagation approach can only be applied to neurons that have continuous and differentiable activation functions and deal with continuous values. This method cannot be applied to SNNs without modification, because one cannot calculate the gradient of a spike, since spikes are not differentiable [150]. Spiking back-propagation is an SNN training approach that adapts the traditional back-propagation algorithm to accommodate the spiking behavior.

There are some approaches to solve this problem, such as considering membrane potentials of spiking neurons as differentiable signals [150] and using surrogate or approximate gradients [8, 145, 146, 151]. In [151], the authors train a spatio-temporal SNN using a supervised Spatio-Temporal Back-Propagation (STBP) algorithm with surrogate gradients as approximate derivatives of spike activity. The STBP algorithm allows error propagation in both the spatial and temporal domains. Spiking back-propagation could be used with recurrent architectures like LSTM-based SNNs [152].

Spike-based back-propagation is successful in training SNNs, but it is computationally expensive and requires a large amount of data for training. The computational demands of this method does not promote efficient on-chip learning [153].

3.2.3 Hebbian learning

Hebbian plasticity is a principle that indicates that the strength of a synapse between two neurons is based on the correlation of the activities of these neurons. In other words, as stated by Donald O. Hebb: "Neurons that fire together, wire together [154]". Hebbian learning is frequently used to train SNNs, and this training can be supervised [155], unsupervised [156], or self-supervised [157].

Hebbian learning is based on the change of synaptic plasticity according to correlations between a neuron's pre-synaptic (input) and post-synaptic (output) activities [154]. There are only few synaptic plasticity rules used for SNNs. The primary synaptic plasticity rule used for training SNNs is Spike Timing-Dependent Plasticity (STDP). Another plasticity rule is the Spike-Driven Synaptic Plasticity (SDSP) [158], but it has very few applications with SNNs [159]. Depending on the choice of hyperparameters, this learning rule can result in inconsistent impact on the learning process, with the same incoming input. With SDSP, some parameter settings may lead to spurious learning [159].

In this section, we only discuss STDP and its variants, then we explain the concepts of neuronal inhibition and homeostasis, which are required to apply STDP effectively over a layer of neurons.

3.2.3.1 STDP

STDP is a biologically plausible Hebbian learning rule [160] that can be used in supervised learning [126], or unsupervised learning [31,32]. In this manuscript, we focus on unsupervised learning paradigms with STDP, as they do not require supervision and can be implemented on ultra-low power hardware.

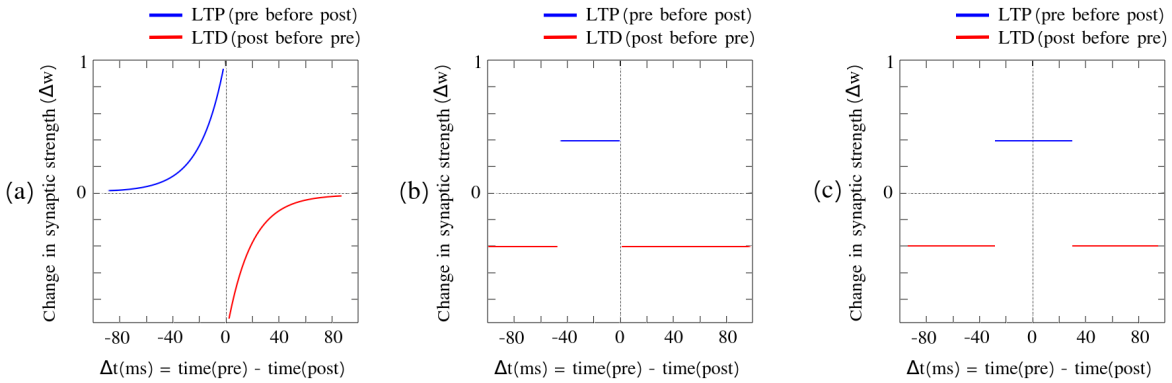


Figure 3.6: (a) The biological STDP rule, (b) The multiplicative STDP rule, (c) The mirrored STDP rule.

The STDP learning rule is used to train SNNs for many tasks, including object recognition [156], as well as action classification [161]. STDP adjusts the synaptic weights based on the relative timing of spikes [162]. Specifically, it strengthens or weakens the synapse between neurons depending on the local correlations of spike timings: if a pre-synaptic spike is fired briefly before a post-synaptic spike, the synapse is strengthened, and this strengthening is referred to as Long-Term Potentiation (LTP "pre before post"); conversely, if a post-synaptic spike is fired briefly before a pre-synaptic spike, the synapse is weakened, which is referred to as Long-Term Depression (LTD "post before pre") [156, 161–164]. This enables the network to learn patterns by reinforcing connections that conveyed spikes involved in generating the output spike, while weakening connections that did not participate and, therefore, are unrelated to the pattern to be recognized. There are some variations of STDP which are mentioned and explained in [147], including:

- Biological STDP: This learning rule [9] is the basic type of STDP observed in biology. Its update function is represented in Figure 3.6(b). This rule applies LTP and LTD to the synaptic weight depending on the difference in firing time between the pre-synaptic and post-synaptic neurons, and the amplitude of the

weight update depends on how far in time the input spike is from the output spike. This learning rule is represented in Equation 3.22 [31]:

$$\Delta_w = \begin{cases} \eta_w e^{-\frac{t_{\text{post}} - t_{\text{pre}}}{\tau_{\text{STDP}}}}, & \text{if } t_{\text{pre}} \leq t_{\text{post}} \\ -\eta_w e^{-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_{\text{STDP}}}}, & \text{otherwise} \end{cases} \tag{3.22}$$

where t_{pre} and t_{post} are the timestamps for input and output spikes respectively, η_w is the learning rate and τ_{STDP} is the time constant responsible for the STDP learning window.

There are some approximations of this rule, which are the additive, and the multiplicative STDP:

- Additive STDP (Add-STDP): The additive STDP depends only on the sign of the time difference between the pre- and post-synaptic neuron spike. However, the absence of weight dependence makes it unstable [165]. Therefore, additive STDP requires hard boundaries to secure the stability of weight dynamics. This results in the weights being closer to the upper and lower bounds, leading to mostly binary weights (0 or 1), due to a saturation effect [31, 147]. The weight update for this learning rule is represented in Equation 3.23:

$$\Delta_w = \begin{cases} \eta_{w+}, & \text{if } \delta t \leq 0 \\ -\eta_{w-}, & \text{otherwise} \end{cases} \tag{3.23}$$

  where $\delta t = t_{\text{pre}} - t_{\text{post}}$ is the relative timing of the spike pair, η_{w+} is a constant learning rate for LTP and η_{w-} is a constant learning rate for LTD.

- Multiplicative STDP (Mult-STDP): The multiplicative STDP is similar to the additive STDP, but it has weight dependence, making this rule slightly more complex than additive STDP [31]. This counters the saturation effect, but consequently weakens the competition among synapses, because weaker synapses have the potential to catch up due to the proportionally larger impact of multiplication on smaller weights, and can result in weakly skewed weight distributions [165]. In some neuroscience research [165], the weight dependence with multiplicative STDP occurs solely during the depreciation phase, while the potentiation is constant like with additive STDP. In the general case, the weight update with this learning rule is represented in Equation 3.24:

$$\Delta_w = \begin{cases} \eta_{w+} e^{-\beta \frac{w - w_{\min}}{w_{\max} - w_{\min}}}, & \text{if } \delta t \leq 0 \\ -\eta_{w-} e^{-\beta \frac{w_{\max} - w}{w_{\max} - w_{\min}}}, & \text{otherwise} \end{cases} \tag{3.24}$$

where $\delta t = t_{\text{pre}} - t_{\text{post}}$ is the relative timing of the spike pair, η_{w+} is a constant learning rate for LTP and η_{w-} is a constant learning rate for LTD.

- Mirror STDP (M-STDP): The main proposition of M-STDP [166] is that the LTP time window should be centered around the outgoing spike, making the update value symmetric, as shown in the update function represented in Figure 3.6(c). This approach allows for the long-term potentiation (LTP) to happen even when the time difference between the outgoing and incoming spike is negative, rather than long-term depression (LTD), which neglects the causality underlined by Hebbian learning. This type of STDP is useful for training networks that use frequency coding [167].

- Probabilistic STDP (P-STDP): This learning rule introduced by [168] yields weights with a probabilistic interpretation after convergence. With P-STDP, the magnitude of the weight adjustment with LTP is related exponentially to the current value of weight w. This rule has amplification parameters a_+ for LTP and a_- for LDT, whose magnitudes are kept within a ratio of 4/3. This rule is represented by Equation 3.25 from [168]:

$$\Delta_w = \begin{cases} a_+ . \exp^{-w}, & \text{if } t_{\text{pre}} \leq t_{\text{post}} \\ -a_-, & \text{otherwise} \end{cases} \tag{3.25}$$

  where t_{pre} and t_{post} are the timestamps for the input and output spikes, respectively.

- Reversed STDP (Rev-STDP): This type of STDP occurs in top-down synapses in the brain (synapses where post-synaptic activity precedes pre-synaptic activity). With this rule, the order of pre- and post-synaptic spikes is reversed compared to the standard STDP rule [169]. Therefore, LTP and LTD are reversed: if a pre-synaptic spike is fired before a post-synaptic spike within a certain time window, LTD occurs. This rule is used when there is classical feed-forward communication and feed-backward communication (in the opposite direction) together in the same network. This rule results in more stable weight distributions when the learning is biased towards depression [147, 169].

- Triplet STDP (T-STDP): This form of STDP operates on the premise that the mechanisms underlying long-term potentiation (LTP) and long-term depression (LTD) are influenced not only by the correlation between a single pair of post-synaptic and pre-synaptic spikes, but also by the correlation within a triplet,

which can involve either two pre-synaptic and one post-synaptic spikes of the same synapse, or one pre-synaptic spike and two post-synaptic spikes) [170]. Triplet STDP can be used to train SNNs for input patterns consisting of higher-order spatio-temporal correlations than pair-based STDP, which will not be able to distinguish any higher-than-pairwise correlations. However, this type of STDP is useful for rate-based patterns [171].

3.2.3.2 Inhibition

Inhibition refers to the reduction or suppression of the activity of certain neurons. This phenomenon is observed in the brain, and is a central mechanism for the regulation of cortical activity [172]. Inhibition plays a crucial role in shaping the dynamics of the SNN. It balances between excitatory and inhibitory mechanisms of neurons, and can permit the dynamic control of the output of spiking neurons [173]. Winner-Takes-All (WTA) inhibition is a method commonly used during training [174, 175]. This inhibition rule dictates that the first neuron that fires a spike is considered the winner neuron. This neuron sends an inhibitory signal to its peers in order to silence them. This method is effective with unsupervised STDP-based learning [176], it creates a competition between the neurons and ensures that they learn different patterns by preventing neurons from firing output spikes and, consequently, triggering STDP updates for the same input pattern. This allows the network to learn more significant, less correlated features.

3.2.3.3 Homeostasis

WTA inhibition can cause some neurons to overpower other neurons, because some neurons have a tendency to fire more spikes than others. This can cause the network to become stuck in a state where the same few active neurons are firing all the time, leaving the other neurons quiet. In order to avoid this and ensure the stability of the network, a homeostasis mechanism is needed. The adaptive threshold method introduced in [177] is one way to insure the homeostasis of the system. This method follows Intrinsic Plasticity (IP) principles in which, when the neuron is highly active, IP weakens the synapses, and when the activity is low, it strengthens the synapses. Thus, it adjusts the neuron threshold based on its activity level, ensuring that neurons respond appropriately to varying input intensities.

Another method is Spike Frequency Adaptation (SFA) [178], which is a mechanism used to regulate the firing rates of the neurons of the network. SFA helps keep the neuron's activity within a desired range by incorporating an adaptation current.

This current gradually builds up with spiking activity and opposes the input current, which leads to reduced firing rates.

An effective threshold adaptation method is the Leaky Adaptive Threshold (LAT) introduced in [177]. This method adjusts the thresholds of all neurons after one neuron fires: the threshold of the firing neuron is increased, while the thresholds of the other neurons are decreased, to promote a balanced activity among neurons.

Another threshold adaptation method, used alongside LAT in the state-of-the-art STDP-based SNN for image analysis [31], is the target timestamp threshold adaptation method from [83]. This method prompts the neurons to fire at a pre-defined target timestamp $\hat{t}$: this allows them to learn specific patterns while maintaining the homeostasis of the network. The thresholds of all neurons (winners and losers), are adapted each time a neuron fires or receives an inhibitory spike, so that their firing time converges towards this target timestamp.

3.2.4 Hybrid learning

Effectively training SNNs poses challenges within both supervised and unsupervised learning paradigms. The limitations of networks trained exclusively using STDP, coupled with issues like overfitting and unstable convergence behaviors in the spiking back-propagation algorithm, have stimulated the development of hybrid learning algorithms [153].

Hybrid learning represents a novel strategy to train SNNs, it combines the advantages of supervised and unsupervised learning techniques. This approach allows SNNs to extract meaningful patterns from input data through pre-training, subsequently followed by gradient descent-based supervised optimization [153]. Compared to a purely gradient descent-based training devoid of pre-training, the hybrid learning approach demonstrates distinct advantages. It fosters heightened robustness, expedites the training process, and augments the network's capacity for generalization [153, 179].

3.3 SNNs for HAR

SNNs have the potential for energy-efficient spatio-temporal feature learning from frame-based videos, spike streams from DVS cameras, and other sources like wearable devices, as previously mentioned in Chapter 1. In this section, we explore the work concerning spike-based HAR in the literature, primarily focusing on frame-based applications. This emphasis is important, because the application of SNNs should

not be restricted solely to DVS data or sensor data; frame-based videos also necessitate analysis. Frame-based videos are an appropriate input for tasks that require fine-grained spatial analysis, such as object detection or semantic segmentation. Additionally, some important tasks concern pre-recorded videos such as online content filtering, and violent cartoon detection [36]. In this section, we focus on HAR with SNNs based on different learning paradigms.

Unsupervised learning: There are some successful application of SNNs with unsupervised learning based on Gene Regulatory Networks (GRNs) paired with the Bienenstock-Cooper-Munro (BCM) model. Gene regulatory networks are inspired from biology. They can identify the regulatory relationships of genes, which can explain how cells function [180]. In artificial neural networks, GRNs can be used to regulate elements such as the synaptic plasticity of a BCM-based SNN [181]. BCM models the synaptic plasticity of spiking neural networks by simulating intrinsic plasticity through the modulation of a specific threshold. Essentially, in response to post-synaptic activity, the threshold adjusts dynamically. When post-synaptic activity deviates significantly from its average, the threshold decreases or increases accordingly, impacting the ease or difficulty of increasing synaptic weights. Then, this threshold is utilized to update the synaptic weights; if the post-synaptic plasticity surpasses the threshold, the synapse undergoes potentiation; otherwise, it undergoes depression [182]. In [181] the authors use GRN dynamics to regulate the plasticity and meta-plasticity of a BCM model. They use the Covariance Matrix Adaptation (CMA) evolution strategy to automatically generate the parameters of GRN dynamics that yield the optimal performance of the BCM model for specific tasks. They test their network with the KTH dataset and achieve a classification rate of 84%, but they remove the problematic "jogging" class, which is easily confused with the "running" class. It gives the network an unfair advantage and makes it incomparable to other work with this benchmark dataset. They also do not disclose their experimental protocol for the Weizmann dataset.

Another unsupervised learning method for SNNs, which is also based on gene regulatory networks for action recognition, is presented in [183]. They test their GRN-BCM-based SNN model with the KTH and Weizmann datasets and achieve a recognition rate of 84.81% with KTH and 74.44% with Weizmann. However, they do not use the standard protocols either. They use four-fold cross-validation to test the efficiency of their model in behavior recognition. The original video sequences are randomly partitioned into four subsamples. For each run, three-fourths of the videos are used for training and the rest are used for test and validation. This gives an unfair advantage for their network and makes it incomparable to any results we

get using the standard protocol.

Another approach involves a heterogeneous recurrent spiking neural network (HRSNN) introduced in [184] with unsupervised learning for spatio-temporal classification tasks for videos. They also do not use the standard KTH protocol.

Supervised learning: Within supervised learning paradigms, SNNs have made remarkable achievements. In [36] the authors use a deep convolutional SNN called SpikeConvFlowNet to detect violent activities in videos. They use a spiking supervised two-stream architecture with IF neurons. This network is made up of an RGB stream, an optical flow stream, a merging block, and fully-connected layers. SpikeConvFlowNet is based on convolutional blocks, where each block is made up of a 2D convolution operation followed by a 2D average pooling operation. This network is trained using the spiking back-propagation algorithm, and the approximated derivatives for IF neuron activations from [150].

Although there is a scarcity of literature exploring spiking two-stream methods, In [185], the authors introduce a deep two-stream SNN based on a spiking ResNet50 architecture and a recurrent spiking neural network (RSNN) fusion module. They use ANN-to-SNN conversion with their own hybrid conversion method. However, ANN-to-SNN conversion still requires the training to be done with a non-spiking ANN, thus reducing the energy efficiency benefits.

Spiking ResNets are used frequently in the SNN literature, as evident in [186,187]. Another instance is found in [188], where the authors present a spatio-temporal Spiking ResNet (STS-ResNet). This SNN is trained with a hybrid SNN training approach, where the authors pre-initialize the weights of their network with non-spiking pre-training, after which they perform ANN-to-SNN conversion. Then, they use skip connections to simulate a spiking ResNet architecture that they train using the STBP method mentioned in [151]. Both of these methods have the limitations that were discussed in Sections 3.2.1 and 3.2.2.

A supervised method for action recognition with SNNs is introduced in [189], where the authors introduce a novel descriptor that relies on joint entropy of difference in magnitude and orientation of the optical flow vectors in order to model human actions. They compute the optical flow using a Pyramid–Warping–Cost volume Network (PWCNet), and apply their feature descriptor. They aggregate the information across frames for long-term temporal dependency, using a spiking neural network. They obtain classification rates of 74.46% and 86.93% on the HMDB51 and UCF101 datasets, respectively.

Furthermore, an SNN with a Liquid State Machine (LSM) multi-layer architecture is presented in [190]. This network is evaluated on the KTH dataset and achieves a

recognition rate of 92.08% when using layers with 2000 spiking neurons. Layers with fewer neurons gave inferior results. They did not use the standard KTH protocol either.

Recurrent Neural Networks (RNNs) have also been explored in the literature for HAR. In [191], a Temporal Spiking Recurrent Neural Network (TSRNN) is introduced. They use temporal pooling operations, where the pooling is implemented at the level of multiple frames instead of the pixels of the same frame. They use GRUs instead of LSTM cells, and they introduce a novel message passing bridge which allows the RNN to correct its contaminated memory, which refers to information erroneously predicted.

There is limited existing research addressing the issue of using spatio-temporal networks from computer vision in the spiking domain. An example of this is applying 3D convolutions for video analysis with SNNs. In [192], the authors combine 3D convolutional SNNs with RNNs, utilizing the supervised Spike Layer Error Reassignment (SLAYER) training mechanism for network training.

Reinforcement learning: In [161], an SNN trained with Reward-modulated Spike Timing-Dependent Plasticity (R-STDP) is introduced. It is based on reinforcement learning, which optimizes the synaptic weights according to the reward or punishment signal that it receives from the decision layer. This system is simpler and more computationally efficient than supervised learning rules [161]. The authors present a convolutional SNN, and they use gradient filters with four different orientations in their feature extraction layer to extract oriented edges, which results in four feature maps per frame. Their method has recognition rates of 94.44% and 92.50% on the Weizmann and KTH datasets, respectively. However, they use the KTH dataset with a different protocol, in which 70% of the samples are considered as the training set, and the remaining 30% of the samples are used for testing. This also confers an unfair advantage to their results compared to using the standard KTH protocol, rendering their results incomparable to other studies utilizing the same benchmark dataset.

In summary, given the limited existing research, building a strong understanding of the performance of SNNs with video analysis tasks is crucial for driving the progress in promising and exciting directions. However, the field of SNNs lacks standardized evaluation protocols, making it challenging to directly compare and replicate results across different studies. We find that the currently existing work with SNNs that is evaluated on frame-based video analysis benchmark datasets does not follow the standard computer vision protocols for these benchmark datasets [183] [161] [193] [161]. While these methods show potential, deviations from

standardized protocols need to be addressed for fair comparisons and meaningful advancements.

3.4 Summary and conclusion

This chapter explores the dominant neural coding mechanisms, like temporal coding, which is an efficient neural coding method in terms of data transfer speed and energy consumption. Temporal coding offers distinct advantages over rate coding by encoding information within spike timestamps. The high spike count required for rate coding makes temporal coding a more efficient alternative [156]. Additionally, adhering to biological principles, where minimal variation in spike amplitude and duration is observed, underscores the significance of spike timing as the primary carrier of information [156].

This chapter has also explored supervised and unsupervised learning techniques within the SNN framework. While supervised approaches like ANN-to-SNN conversion and spiking backpropagation prioritize achieving high recognition rates, they may overlook key advantages of SNNs, such as potential energy efficiency during training on dedicated hardware. The reliance of ANN-to-SNN conversion on a regular ANN for training limits its advantages compared to a fully spiking model. Similarly, spiking backpropagation, despite being adapted for direct SNN training, presents notable limitations: it demands substantial labeled data for effective training and involves non-local computations, complicating implementation on low or ultra-low power neuromorphic hardware.

In contrast, unsupervised learning techniques conduct direct SNN training, feature local computations, and are more straightforward to implement on neuromorphic hardware [31]. This approach aligns better with the inherent SNN structure and eliminates the need for abundant labeled data for learning significant features. However, despite the potential advantages of unsupervised Hebbian SNN training techniques, they remain unexplored enough for video analysis.

There is a gap in the literature regarding unsupervised spatio-temporal STDP-based models for video analysis, which hold the potential to develop cost-efficient, secure, and easily implementable video analysis systems on neuromorphic hardware. Upon reviewing existing literature, we have noted the significant advantages offered by CNNs in processing visual data. However, we identify a research deficiency concerning STDP-based CSNNs and STDP-based spiking spatio-temporal methods for video analysis in general. As a result, we deem unsupervised STDP-based CSNNs as crucial focal points in our research. Our objective is to extract spatio-temporal features from videos using STDP-based SNNs.

Chapter 4

Foundation Architecture

Unsupervised STDP-based CSNNs are crucial focal points in our research. Our objective is to extract spatio-temporal features from videos with STDP-based SNNs. The first step to achieve this is establishing a foundation model for our CSNNs, which will serve as a basis to handle video analysis in the subsequent contribution chapters. This baseline model is illustrated in Figure 4.1, and elaborated upon in this chapter.

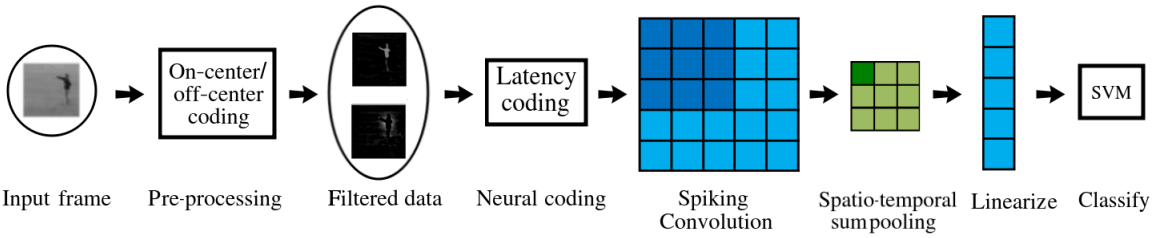


Figure 4.1: General recognition pipeline that serves as a basis in the subsequent contribution chapters.

4.1 On-center/off-center filtering

Pre-processing is often necessary for an STDP-based SNN to effectively learn visual features from frame-based visual information with temporal coding. This is because STDP tends to capture correlations among early spikes; hence, the interesting features should be present in the earlier spikes for the network to learn aspects such as shape or texture patterns. Therefore, STDP-based SNNs need to be fed salient information like edges. Otherwise, the sum of input patterns becomes close to zero

with dark patches, and very high with bright patches, which leads the kernels to converge towards similar or uninformative patterns [31].

A commonly used technique is on-center/off-center filtering. This filter is made to mimic cells in the human retina called retinal ganglion cells [194]. The on cells are activated by an increase in the intensity of light, while the off cells are activated by a decrease in the intensity of light [195]. The on-center/off-surround receptive field has a maximum activation when the light stimuli are in the center, and is inhibited when the light stimuli are in the surrounding region. Inversely, the off-center/on-surround receptive field is activated by light stimuli in the surrounding region, while it is inhibited by light stimuli in the center. This filter is useful in applications with CNNs [196], as well as SNNs [31].

Video frames are pre-processed one at a time using this on-center/off-center filter, which is mathematically modeled by a Difference-of-Gaussians (DoG) filter. The edges of each frame are extracted by convolving the input frame with a kernel of differences of Gaussians with different spatial scales, as shown in Equation 4.1 from [31]:

$$\mathrm{DoG}_{s,\sigma_1,\sigma_2}(x,y) = I(x,y) * (G_{s,\sigma_1} - G_{s,\sigma_2}) \tag{4.1}$$

where I is the input image, $*$ is the convolution operator and $G_{s,\sigma}$ is a zero-mean normalized Gaussian kernel of size $s \times s$ and standard deviation σ. This 2D Gaussian kernel is defined as [31]:

$$G_{s,\sigma}(u,v) = \frac{e^{-\frac{u^2+v^2}{2\sigma^2}}}{\sum_{i=-\mu}^{\mu}\sum_{j=-\mu}^{\mu} e^{-\frac{i^2+j^2}{2\sigma^2}}}, u, v \in [-\mu, \mu], \mu = \left\lfloor \frac{s}{2} \right\rfloor. \tag{4.2}$$

The positive and negative values generated from this filtering process are separated into two channels, where the positive one represents the on cells, and the negative one represents the off cells as shown in Equations 4.3 and 4.4 from [31]:

$$x_{\text{on}} = \max(0, \mathrm{DoG}(x,y)) \tag{4.3}$$

$$x_{\text{off}} = \max(0, -\mathrm{DoG}(x,y)) \tag{4.4}$$

4.2 The IF neuron model and latency coding

The most used spiking neuron for computer vision tasks are the IF and LIF neurons [197]. The leak of LIF neurons gradually reduces the membrane potential over time, which may lead to the initial spikes being forgotten and giving more significance to subsequent spikes. This is not advantageous for visual data, as the earliest spikes

could potentially contain the most critical information [139]. Therefore, the spiking neurons most suitable for our work are the integrate-and-fire (IF) neurons [133], described in Chapter 3. These neurons are effective in visual pattern learning, which may not require the more complex mechanisms like bursting or leak adopted by other neuron models. Another major advantage of this model is that it uses fewer hyper-parameters compared to other models, which facilitates hyper-parameter search [31]. We use a simplified version of the equations which represent the IF neuron discussed in Chapter 3 (page 46), where we set $c_m = 1$ and $V_r = 0$, as shown in Equation 4.5:

$$\begin{aligned} V_m(t) &= V_r + \sum_{i \in \mathcal{E}} W_i f_s(t - t_i) \\ V_m(t) &\leftarrow V_r \text{ when } V_m(t) \geq V_{\text{th}}(t) \end{aligned} \tag{4.5}$$

$$f_s(t - t_i) = \begin{cases} 1, & \text{if } t \geq t_i \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

where $\mathcal{E}$ represents the set of incoming spikes, w_i is the weight of the synapse carrying the i-th spike, t_i represents the timestamp of the i-th spike, and f_s is the kernel of spikes.

SNNs that use IF neurons have been used together with multiple different neural coding schemes. Temporal coding schemes have shown the highest classification accuracy, fastest inference speed, and lowest energy consumption with these models [136, 198]. Previous work shows that with visual data analysis, one spike per neuron per sample is effective [31, 199]. Therefore, we have selected temporal latency coding as our preferred neural coding method to transform input videos from pixels into spiking data. We recall the equation of latency coding discussed in the preceding Chapter 3 (page 49):

$$f_{\text{in}}(x) = (1.0 - x) \times t_{\text{exposition}} \tag{4.7}$$

where f_{in} is the resulting spike timestamp for the neuron, $x \in [0, 1]$ is the input value, and $t_{\text{exposition}}$ represents the presentation duration for one sample.

4.3 Spiking 2D convolutions

2D spiking convolutional layers, as introduced in [31], have f_k trainable kernels, each with dimensions $f_w \times f_h$. Each filter is represented by a neuron that establishes connections with $f_w \times f_h$ inputs from the preceding layer, as shown in Figure 4.2. These

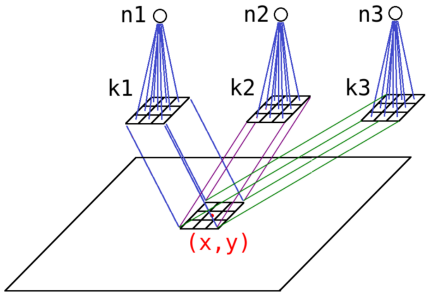


Figure 4.2: The neighborhood of an output neuron at (x, y) is a plane, and n1, n2, and n3 are competing neurons at the same location. The channel dimension is not drawn in this figure.

convolutional filters traverse the spatial dimensions of an input, sized $l_w \times l_h \times l_c$, where l_w, l_h and l_c are the width, height, and channel dimensions, respectively, with a designated stride (representing the step between successive positions). Mathematically, the operation of a spiking 2D convolution with an IF neuron, at a given location (x, y) in the input, can be expressed as detailed in Equation 4.8.

$$V_{m_{x,y,k}}(t) = \sum_{i \in \mathcal{E}} W_{n(x_i),j(y_i),k_i,k} \times f_s(t - t_i) \tag{4.8}$$

where $V_m(t)$ is the neuron membrane potential at time t, x, y and k are the coordinates of the neuron in the width, height, and channel dimensions, respectively, W is the trainable synaptic weight tensor, $n()$ and $j()$ are functions that map the location of the input neuron to the corresponding location in the weight tensor, and f_s is the kernel of spikes represented in Equation 4.6.

4.4 The STDP learning rule

Supervised SNN training techniques, such as ANN-to-SNN conversion and spiking backpropagation, have several limitations, as discussed in Chapter 3. Consequently, we advocate for unsupervised learning through STDP as a more favorable SNN training approach. It helps reduce the need for large amounts of labeled data, since the training of features is done without supervision, although some labeled data is still needed to train the subsequent supervised classifier. Moreover, the locality of this learning rule can facilitate implementation on neuromorphic hardware, as highlighted in [31].

Weights are initialized with a random uniform distribution with $W \sim U(0,1)$. The stability of these weights greatly depends on the type of STDP used [147].

We choose the biological STDP rule [9] for its stability and superior performance compared to other types of STDP when applied to visual tasks [31]. As mentioned in Chapter 3 (page 54), this learning rule is characterized by Equation 4.9 [31]:

$$\Delta_w = \begin{cases} \eta_w e^{-\frac{t_{\text{post}} - t_{\text{pre}}}{\tau_{\text{STDP}}}} & \text{if } t_{\text{pre}} \leq t_{\text{post}} \\ -\eta_w e^{-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_{\text{STDP}}}} & \text{otherwise} \end{cases} \tag{4.9}$$

where t_{pre} and t_{post} are the timestamps for input and output spikes respectively, η_w is the learning rate and τ_{STDP} is the time constant responsible for the STDP learning window.

This equation computes weight updates, which happens when the membrane voltage, discussed in the previous section, $V_{m_{x,y,k}}(t)$ surpasses the threshold $V_{\text{th}}(t)$ of the IF neuron (i.e. when an output spike is fired).

4.5 Threshold adaptation method

In this manuscript, we use WTA inhibition, explained in Chapter 3 (page 56). Therefore, to avoid having a state where one neuron fires all the time and dominates the others, we also need threshold adaptation to maintain the homeostasis of the system.

We use two threshold adaptation methods. The main threshold adaptation method used to maintain the homeostasis of our system is the Leaky Adaptive Threshold (LAT) [177]. We also use the target timestamp threshold adaptation method from [83], which is used in conjunction with LAT in the state-of-the-art 2D CSNN in [31]. This method prompts the neurons to fire at a pre-defined target timestamp $\hat{t}$. This allows biasing the model to learn specific patterns while also contributing to maintaining the homeostasis of the network.

The thresholds of all neurons (winners and losers), are adapted each time a neuron fires or receives an inhibitory spike, so that their firing time converges towards this target timestamp [31]. The thresholds are updated according to Equations 4.10, which represents the target timestamp threshold adaptation, Equation 4.11 for LAT, and 4.12 which shows how the membrane threshold is updated according to both threshold adaptation methods:

$$\Delta^1_{\text{th}} = -\eta_{\text{th}}(t - \hat{t}) \tag{4.10}$$

$$\Delta_{\text{th}}^2 = \begin{cases} \eta_{\text{th}}, & \text{if } t_i = \min\{t_0, ..., t_N\} \\ -\frac{\eta_{\text{th}}}{l_d(n)}, & \text{otherwise} \end{cases} \tag{4.11}$$

$$V_{\text{th}}(t) = \max(\text{th}_{\min}, V_{\text{th}}(t-1) + \Delta_{\text{th}}^1 + \Delta_{\text{th}}^2) \tag{4.12}$$

where t is the timestamp at which the neuron fires, η_{th} is the threshold learning rate, l_d is the number of neurons that are in competition in the layer, t_i is the firing timestamp of neuron i, $\min\{t_0, ..., t_N\}$ is the minimum timestamp, which corresponds to the neuron that fired first, $\text{th}_{\min}$ is the minimum possible threshold value [31], and N the number of neurons in the layer.

4.6 Algorithm

The training algorithm for the 2D CSNN is presented in Algorithm 1. During the training of a 2D spiking convolutional layer, for each input sample, a specific fixed number of patches n_{patches} are randomly selected from the input tensor. These patches are used as input to train the feature learning algorithm, which produces the dictionary of learned features. This strategy prevents the simultaneous update of the same filter at numerous locations, thereby enhancing the convergence of the network, as explained in [31].

In the training process of an SNN, only one patch is considered at a time, hence a single column is activated, which eliminates the need for inter-column communications. Here, a column refers to a collective group of neurons situated at identical positions across different depths of feature maps. Once a neuron in the column fires a spike, it inhibits the other neurons in the column and its weights and threshold are updated. During training, the input spikes that contributed to the firing of the output spike are kept track of. Each synapse that has carried a spike that contributed to firing an output spike is strengthened, while all other synapses of the neuron, that did not contribute, are weakened. This enables learning meaningful patterns by the convolutional kernels. When the training for one patch is done, the parameters including weights and thresholds are replicated across other columns filter by filter [31].

Algorithm 1 Train function for one input sample patch

function TRAIN($input\_spikes, input\_time$)
  $V_m \leftarrow [0, 0, ..., 0]$ ▷ Neuron activations are set to zero
  for all *spike* in $input\_spikes$ **do**
    for f_{indx} from 0 to f_k **do**
      $V_m[f_{indx}] \leftarrow V_m[f_{indx}] + weights[spike.x, spike.y, spike.z, f_{indx}]$
      if $V_m[f_{indx}] < V_{th}[f_{indx}]$ **then**
        skip to next iteration
      end if
      for fn from 0 to f_k **do** ▷ Neuron thresholds converge towards $\hat{t}$
        $V_{th}[fn] \leftarrow V_{th}[fn] - \eta_{th} * (spike.time - \hat{t})$
        if $fn = f_{indx}$ **then**
          $V_{th}[fn] \leftarrow V_{th}[fn] + \eta_{th}$ ▷ Increase V_{th} of the neuron that fired
        else
          $V_{th}[fn] \leftarrow V_{th}[fn] - \eta_{th}/(f_k - 1)$ ▷ Decrease V_{th} of neurons that did not fire
        end if
        $V_{th}[fn] \leftarrow \max(th_{\min}, V_{th}[fn])$ ▷ Ensure a minimum threshold value
      end for
      for x from 0 to f_w **do** ▷ STDP updates: synapses of the neuron that fired
        for y from 0 to f_h **do**
          for k from 0 to l_c **do**
            $pre \leftarrow input\_time[x, y, k]$ ▷ pre synaptic spike timestamp
            $post \leftarrow spike.time$ ▷ post synaptic spike timestamp
            if $pre \leq post$ **then**
              $W[x, y, k, f_{indx}] \leftarrow W[x, y, k, f_{indx}] + \eta_w e^{-\frac{t_{\text{post}} - t_{\text{pre}}}{\tau_{\text{STDP}}}}$ ▷ Apply LTP
            else
              $W[x, y, k, f_{indx}] \leftarrow W[x, y, k, f_{indx}] - \eta_w e^{-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_{\text{STDP}}}}$ ▷ Apply LTD
            end if
          end for
        end for
      end for
      return ▷ WTA inhibition, go to the next patch of spikes
    end for
  end for
end function

4.7 Classifier

The spiking feature maps produced at the output of the network are then converted back into non-spiking representations prior to classification. This conversion is performed because we use a non-spiking classifier in this manuscript. Latency coding was used to generate these spikes, therefore, the inverse of this coding function can generate the output feature maps. Also, the output of the network must take into account the effect of the target timestamp threshold adaptation represented by the parameter $\hat{t}$, as shown in Equation 4.13 from [31]:

$$f_{\text{out}}(t) = \min\left(1.0, \max\left(0.0, 1.0 - \frac{t - \hat{t}}{t_{\text{exposition}} - \hat{t}}\right)\right) \tag{4.13}$$

with f_{out} the converted value, t the spike timestamp from the feature map, which is zero if there is no spike, $t_{\text{exposition}}$ the duration of presentation of the sample, and $\hat{t}$ the target timestamp.

The resulting non-spiking feature maps undergo sum pooling to reduce their dimensions into a standard size. They are then linearized into vectors, and finally introduced into the classifier. We use a Support Vector Machine (SVM) to classify the extracted features. Any other supervised method can be used for the final classification; we chose an SVM because it is standard and effective with default hyperparameters.

4.8 Video processing approaches with SNNs

Considering the lack of an STDP-based CSNN for video feature learning in the current literature, we extend the state-of-the-art STDP-based 2D CSNN [31], originally designed for image feature learning, to be applicable for video analysis. This extension is achievable by processing individual frames and subsequently incorporating an additional step to combine the resulting feature maps, as illustrated in Figure 4.3 (a). This enables elementary video processing with a 2D CSNN without the need for temporal information processing by the CSNN.

Another approach is to concatenate the video frames to preserve the temporal information before converting the sample into spikes, as shown in Figure 4.3 (b), and then processing it with the 2D CSNN. These two approaches do not require changing the architecture of the 2D CSNN.

Transforming the entire video into a 4D tensor of spikes and introducing it into a spatio-temporal version of this CSNN for spatio-temporal processing, as shown in

Figure 4.3 (c) can open many new avenues regarding video analysis.

In this manuscript, we present our contributions, where in the first chapter, we use pre-processing methods to model motion in the spatial domain for video analysis using settings like in Figure 4.3 (a) and (b), then we introduce spatio-temporal SNNs using settings in Figure 4.3 (c), in subsequent chapters.

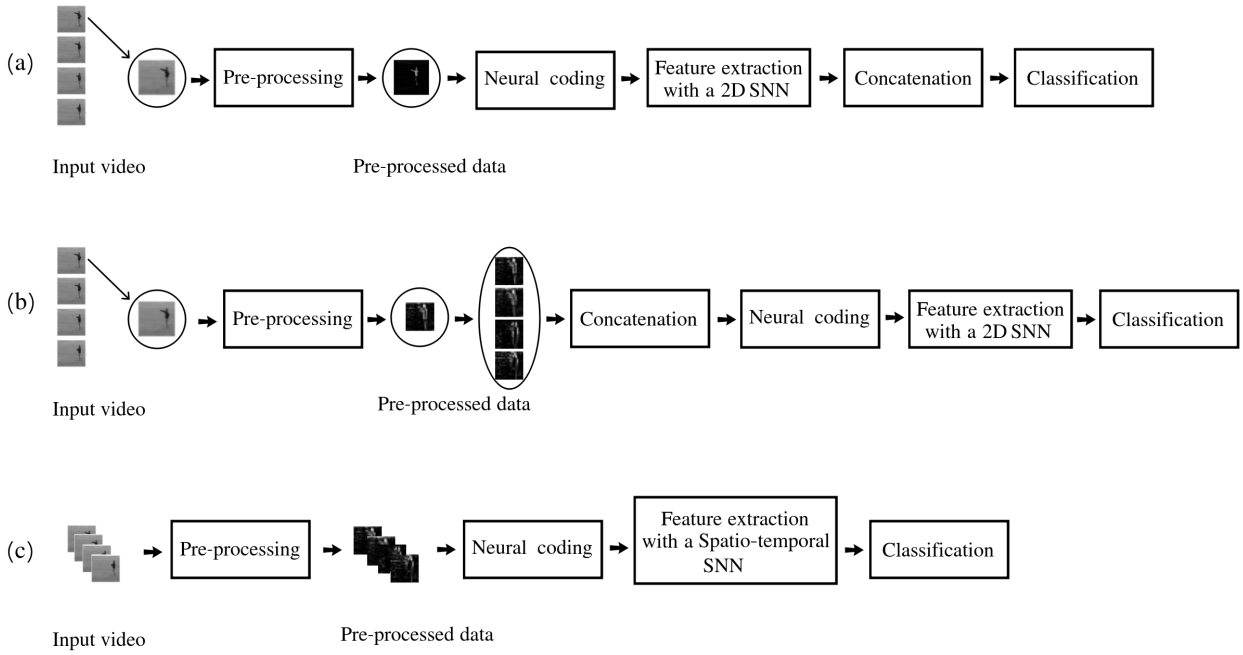


Figure 4.3: Video processing with (a) A 2D SNN with one input frame at a time and feature concatenation at the output of the CSNN. (b) A 2D SNN with concatenated frames as input. (c) A spatio-temporal SNN, with a video tensor as input.

4.9 Datasets with their protocols and implementation details

The KTH [54], Weizmann [55], UCF-sports [57] and IXMAS [56] datasets are used to evaluate the contributions in this manuscript.

For the KTH dataset, the subjects 11, 12, 13, 14, 15, 16, 17 and 18 are used for training, while 02, 03, 05, 06, 07, 08, 09, 10 and 22 are used for testing, and subjects 19, 20, 21, 23, 24, 25, 01, 04 are used for validation, as indicated in the KTH protocol. Experiments with the IXMAS and Weizmann datasets are done using the leave-one-

subject-out strategy. The protocol used for the UCF sports dataset is to randomly select one-third of the videos for each action class for testing, while the remaining two-thirds are used for training.

In our contributions, we conduct the experiments with either three or five runs due to the lengthy processing time required by the simulator, which can extend up to 40 hours. LOSO protocols demand numerous experiments, sometimes totaling up to 230 hours. Another reason is exhaustive hyperparameter tuning, which requires running these large experiments a large number of times.

To shorten the running time of experiments, we take subsets of the video frames (like in [115] and [3]): the specificities of these subsets depend on the experiments, and will be detailed in the related sections. We scale down the frame sizes to half of their original sizes for processing speed reasons. We measure the classification accuracy (in %) on the test set for all experiments.

4.10 Summary

We have opted for an unsupervised STDP-based CSNN composed of Integrate-and-Fire (IF) neurons [133]. These neurons are connected by synapses in a convolutional manner, with trainable weights initialized with a normal distribution and trained with the biological STDP learning rule. To ensure the learning of distinctive patterns, we employ Winner-Takes-All (WTA) inhibition during training. To maintain network homeostasis, we use LAT [177] as the threshold adaptation mechanism. Moreover, we incorporate the target timestamp threshold adaptation [83], which not only contributes to homeostasis but also primarily aims to bias the network towards acquiring a specific type of features. These choices encompass the ones used in the state-of-the-art architecture for STDP-based feature learning with image analysis presented in [31].

We pre-process videos from frame-based datasets with an on-center/off-center filter for the extraction of edges [196]. Then, the response to the filter undergo transformation into spiking input data through temporal latency coding [200]. This prepared data is then fed into the STDP-based SNN, which processes the data, extracts the features, and gives feature maps as an output. These feature maps then undergo sum pooling to reduce their size before classification. We use a supervised classifier to classify the features that are extracted in an unsupervised manner by the SNN.

This foundational architecture serves as the baseline for our forthcoming contributions that aim for video analysis using unsupervised STDP-based SNNs.

Part III

Contributions

Chapter 5

Static Representations of Motion

Spiking neural networks trained with unsupervised STDP for video analysis are not addressed enough in the literature, as discussed in Chapter 3. However, these networks have the potential for efficient low-energy video analysis. Therefore, it is prominent to study the behaviors of these networks with motion data. A 2D CSNN, like the one introduced in Chapter 4, is explicitly designed to process static information, such as images. When we employ a 2D CSNN to analyze video samples, it only extracts appearance information from each individual frame, thereby discarding valuable motion information between the video frames. However, there are handcrafted methods that can preserve the temporal component between video frames, enabling a 2D CSNN to perform video analysis without requiring any changes to its network architecture.

The motion representations presented in this chapter are based on dense optical flow [63] and can be optionally accompanied by fusion techniques [86]. This enables 2D CSNNs to perform video analysis while preserving motion information between frames, thereby providing an assessment of the performance of 2D CSNNs in action recognition.

We begin by a discussion on data fusion techniques [86], where we explain their utility in enabling 2D CSNNs to process videos. Then, we discuss the concept of frame subtraction as an optional pre-processing method that we use primarily to get rid of the static background pixels, which may impact the performance of models that use temporal coding. However, frame-subtraction does not only eliminate static background pixels, it also extracts elementary motion information, therefore feeding the model directly with motion information rather than appearance data. Following that, we introduce methods for extracting motion information from videos and generating static representations that encode this motion. Finally, we evaluate

these static representations using the pipeline introduced in Chapter 4, equipped with a one-layer CSNN. In this chapter, we use a single-layer CSNN because multi-layer CSNNs trained with STDP are still immature and considered an open research problem [83].

The main contributions of this chapter are summarized as follows:

- we use an STDP-based unsupervised 2D CSNN model initially created for image analysis, to learn spatio-temporal patterns for video analysis;
- we implement six handcrafted optical flow-based motion representations, which we categorize into two groups: four frame-based and two shot-based representations. Notably, two of the frame-based methods and both shot-based methods are our distinctive contributions to the field of handcrafted motion modeling;
- we evaluate and compare the performance of the 2D CSNN with these representations on both the KTH and Weizmann action recognition datasets;
- we provide an analysis of the effects of frame subtraction on the performance of the 2D CSNN.

5.1 Fusion techniques

2D CSNNs do not inherently handle motion information. Consequently, it is necessary to preserve this motion, which permits understanding how visual patterns change over time. One approach to achieve this is through data fusion [86]. Fusion enables the analysis of combined information from a subset of video frames by a 2D CSNN, ensuring that temporal information within the sequence is preserved. There are different types of fusion techniques that can be applied in conjunction with neural networks: early, late, and slow fusion [86]. Early fusion is done by fusing the input frames together into one representation, before processing the sample, thus presenting the entire video clip, which implicitly contains motion, to the 2D CSNN in the spatial domain. On the other hand, late fusion occurs after the 2D CSNN has processed the frames and generated feature maps: these feature maps are fused together to create one representation. Slow fusion is a mix of early and late fusion, such that deeper layers of the network process more global information in both spatial and temporal dimensions [86], but this method is not applicable with our single-layer architecture.

The fusion methods discussed in this section are applicable with different types of input data. They can be applied to raw video frames, or frames that have been pre-processed to encode motion.

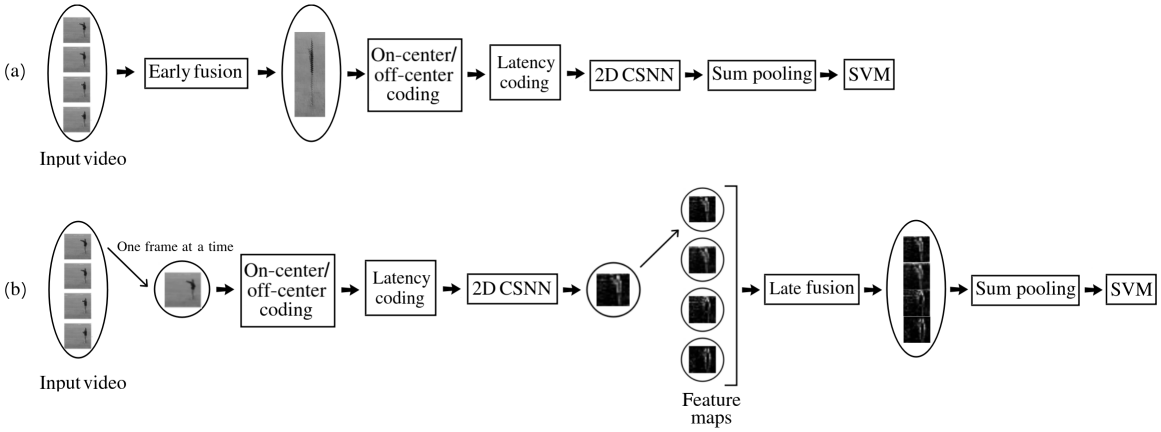


Figure 5.1: (a) Early fusion and (b) Late fusion.

5.1.1 Early fusion

The early fusion method presented in [86] extends the convolution across time in the first layer of the network. This combines information across multiple frames immediately at the pixel level, and requires the convolution kernel to contain a temporal dimension, as explained in [86]. In this chapter, we are interested in processing a video sample successfully without altering the architecture of the 2D CSNN, or its kernel dimensions. To achieve this, we have implemented our early fusion method as a pre-processing step to the video samples, as shown in Figure 5.1 (a). This approach transforms the video sample into a single frame before any processing occurs within the CSNN. This intertwined representation is intended to permit the 2D CSNN kernel to learn local motion patterns across the sample. The consecutive frames are concatenated to form one large frame in a row-by-row manner, as shown in Figure 5.2 (a) and represented by Equation 5.1, or a column-by-column manner, as shown in Figure 5.2 (b) and represented by Equation 5.2:

$$I_o(x, r) = I^n(x, y) \text{ with } r = y \times l_{td} + n \tag{5.1}$$

$$I_o(c, y) = I^n(x, y) \text{ with } c = x \times l_{td} + n \tag{5.2}$$

where I^n is the input frame of width l_w, height l_h, and index n, l_{td} is the total number of input frames, I_o is the output frame of width l_w and height $l_h \times l_{td}$ for Equation 5.1, and of width $l_w \times l_{td}$ and height l_h for Equation 5.2, r is the row index, c is the column index, and $x \in [0, l_w - 1]$ and $y \in [0, l_h - 1]$ are the pixel coordinates in the horizontal and vertical dimensions in the original frame, respectively.

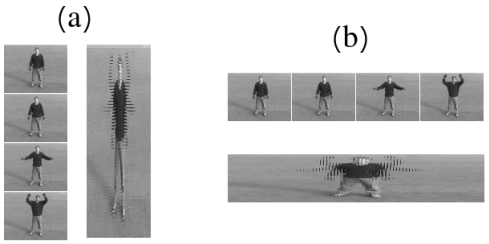


Figure 5.2: Early fusion: multiple input frames are fused together (a) row by row, or (b) column by column.

5.1.2 Late fusion

Late fusion is implemented as an independent post-processing technique that does not change the network architecture of the 2D CSNN, as shown in Figure 5.1 (b). The video is processed frame by frame, by the CSNN, and the resulting output feature maps are fused together. This fusion can be done in multiple ways. One way is simply concatenating the output feature maps and linearize the resulting map before sending it as one large linear vector into the classifier. Another method for implementing late fusion involves sum pooling the output feature maps in the temporal dimension, after they have been transformed back into regular values from spikes, before reaching the classifier.

5.2 Frame subtraction

A pre-processing method that discards irrelevant edges, like the ones in the background, could be useful with CSNNs that use DoG filtering and temporal coding. A straightforward approach to achieve this is to focus on extracting edges in motion and discarding static pixels in HAR videos. This can be done by subtracting each pair of consecutive frames.

This provides positive or negative values depending on pixel variations, and removes the stationary spatial information, as shown in Figure 5.3 (a). The latency coding that we use requires input values to be positive, but the on-center/off-center filter introduced in Equation 4.1 handles negative inputs natively. Still, there are different ways to represent the values resulting from the difference in pixel illumination. The pixel difference between successive frames is represented by Equation 5.3:

$$P_d^n(x,y) = I^n(x,y) - I^{n+1}(x,y) \tag{5.3}$$

The output frame that results from frame subtraction can be handled as follows:

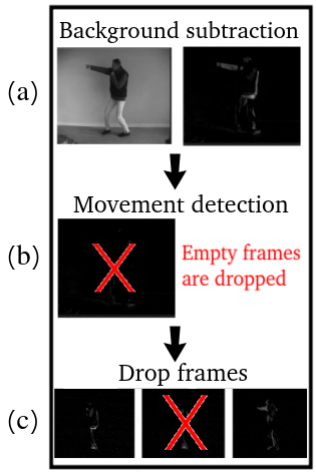


Figure 5.3: (a) Frame subtraction. (b) Empty frame elimination. (c) Dropping some frames in order to capture a full action in a smaller video clip.

- take the absolute value of the resulting values:

$$I_o^n(x,y) = |P_d^n(x,y)|, \tag{5.4}$$

- separate the positive and negative values into two different channels, and take the absolute value of the negative ones:

$$I_o^n = \begin{cases} I_o^n(x,y,c1) = P_d^n(x,y) \text{ if } P_d^n(x,y) \geq 0 \\ I_o^n(x,y,c1) = 0 \text{ if } P_d^n(x,y) \leq 0 \\ I_o^n(x,y,c2) = 0 \text{ if } P_d^n(x,y) > 0 \\ I_o^n(x,y,c2) = |P_d^n(x,y)| \text{ otherwise,} \end{cases} \tag{5.5}$$

- retain the negative values as they are, to be handled by the on-center/off-center filter, which would produce edges of regions in motion:

$$I_o^n(x,y) = P_d^n(x,y) \tag{5.6}$$

where I^n is the input frame of index $n \in [0, l_{td} - 2]$, I_o^n is the output frame resulting from frame subtraction, $c1$ and $c2$ are channels of I_o^n, P_d^n is the pixel difference value represented by Equation 5.3, and (x, y) are the pixel coordinates.

Frame subtraction is an optional pre-processing step. However, it changes the nature of the data that the subsequent process will receive from spatial information

to raw motion information, as the values after subtraction are pixel changes. It can be followed by movement detection for empty frame elimination, as shown in Figure 5.3 (b), in order to avoid processing empty frames. This is done by setting a threshold value on the sum of pixels in the frame that results from subtraction. If its sum of pixels is lower than the threshold θ_m, then frame I_o^n is dropped, as shown in Equation 5.7:.

$$\text{Drop } I_o^n \text{ if } \sum_x \sum_y I_o^n(x, y) < \theta_m. \tag{5.7}$$

Finally, this method can include systematical frame dropping of two frames between each two selected frames, regardless of whether the frame is empty or not, as shown in Figure 5.3 (c), in order to shorten the video and have a more compact representation of the action in a shorter clip.

5.3 Optical flow-based motion representations

Optical flow can estimate the motion between two consecutive frames in a video, as mentioned in Section 2.3. In this section, we use Farneback's dense optical flow [63] and explore six static video representations. We then evaluate the performance of a 2D CSNN with these representations. We split these representations into two categories, which are frame-based and shot-based representations.

Frame-based representations process a video clip, focusing on modeling the motion between each pair of consecutive frames at a time. This results in processed individual frames that are static representations of motion. Taken individually, a single frame does not contain enough motion to represent an entire action. This is because the difference between some actions, such as clapping and waving, are subtle and require the consideration of multiple frames for proper differentiation. Therefore, fusion methods are required in conjunction with frame-based representations to incorporate a greater number of frames as input for a 2D CSNN. Figure 5.4 shows a classification pipeline using a static representation of motion with early or late fusion. The resulting representation after fusion contains enough motion data to represent the entire clip.

Shot-based representations extract motion information from more than two frames at a time, and transform this motion into a static representation of the entire video clip. Therefore, fusion methods are not needed with these representations, as they inherently involve some type of early fusion, as illustrated in Figure 5.5.

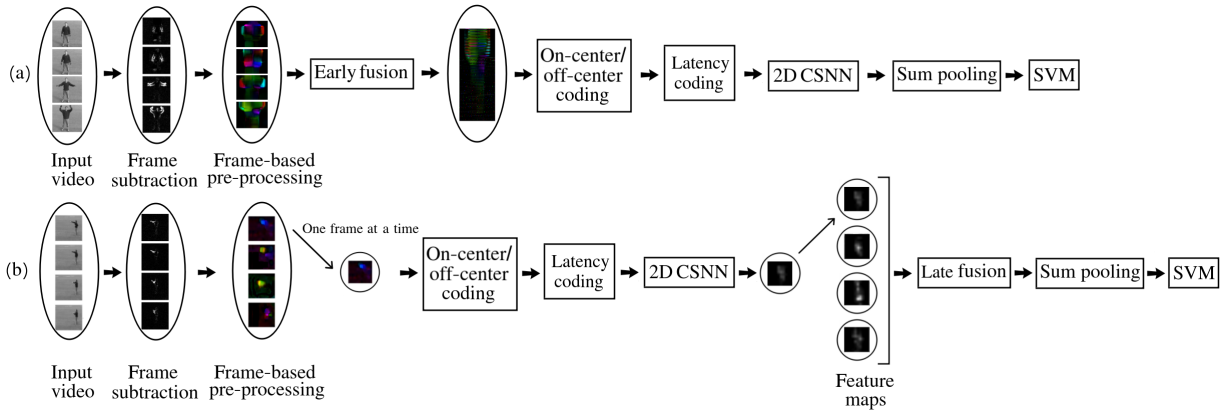


Figure 5.4: A frame-based representation of an HAR video taken as input to a 2D CSNN. (a) Early fusion. (b) Late fusion.

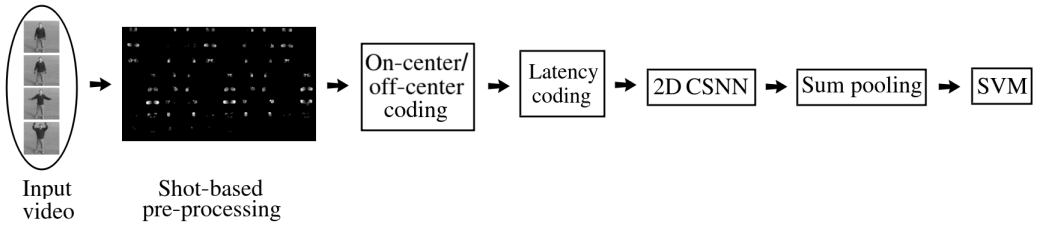


Figure 5.5: A shot-based representation of an HAR video taken as input by a 2D CSNN.

In the rest of this section, we detail four frame-based and two shot-based static representations of motion.

5.3.1 Frame-based methods

Displacement (DXDY) This representation is done by taking the raw dense optical flow displacement in the horizontal and vertical directions, and storing them into a two-channel frame, as shown in Equation 5.8, and illustrated in Figure 5.6 (B). The negative values are dealt with by the on-center/off-center filter.

$$\begin{aligned} I_o^n(x, y, c1) &= \mathrm{OF}_x^n(x, y) \\ I_o^n(x, y, c2) &= \mathrm{OF}_y^n(x, y) \end{aligned} \tag{5.8}$$

where I_o^n is the output frame with channels $c1$ and $c2$, n is the frame index, OF_x and

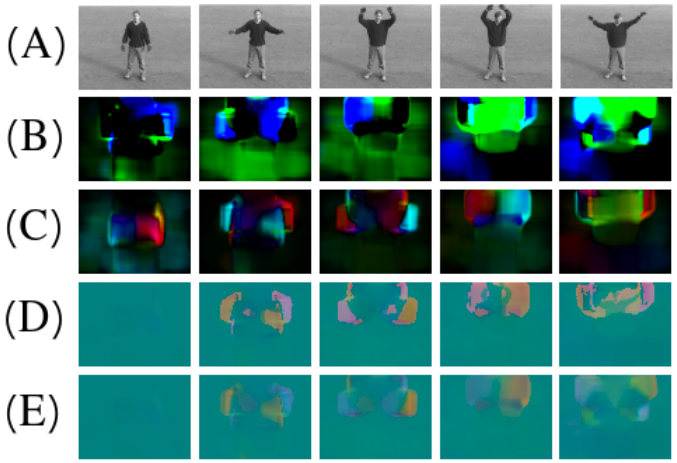


Figure 5.6: A waving action. (A) The original frames. (B) The DXDY representation (in RGB, R: 0, G: OF_x^n, B: OF_y^n). (C) The OA representation (in HSV, H: orientation, S: 0, V: amplitude). (D) The CC representation (in RGB, R: OF_x^n, G: OF_y^n, B: the moving part of the original grayscale image). (E) The CCOA representation (in RGB, R: OF_x^n, G: OF_y^n, B: the average of the optical flow RGB values).

OF_y are the horizontal and vertical components of the optical flow vectors, respectively, and (x, y) are the pixel coordinates.

Orientation and Amplitude (OA) With this representation, the optical flow displacement in the horizontal and vertical directions are computed as the magnitude and orientation of optical flow, as defined in Equations 5.9 and 5.10:

$$\text{magnitude}(x, y) = \sqrt{\text{OF}_x(x, y)^2 + \text{OF}_y(x, y)^2} \tag{5.9}$$

$$\text{orientation}(x, y) = \arctan\left(\frac{\text{OF}_y(x, y)}{\text{OF}_x(x, y)}\right) \tag{5.10}$$

where $\text{OF}_x(x, y)$ is the horizontal component of the optical flow vector at position (x, y), and $\text{OF}_y(x, y)$ is its vertical component. Since orientation data is periodic, it is difficult to apply latency coding to it. Thus, optical flow is displayed in the HSV color space, using orientation as the hue component and magnitude as the value component, then converted into the RGB color space, as shown in Figure 5.6 (C).

Composite Channels (CC) This representation joins spatial and temporal information. It includes the optical flow displacement OF_x and OF_y that are placed sepa-

rately in the first two channels. Then we incorporate the grayscale information of the moving parts from the original image into the third channel. We compute the moving parts of the subject by subtracting pixels of successive frames $I^n(x,y) - I^{n+1}(x,y)$ at every location (x, y). If the difference is zero, then this pixel at this location is stationary, and is set to 0 in the output channel. If this difference is non-zero, then this pixel is not stationary, and is set to $I^n(x,y)$ in the third channel of the CC frame. This way, we keep the gray scale spatial information of the moving part of the subject. Therefore, CC collects the spatial and temporal information in the same frame, as shown in Figure 5.6 (D), and expressed in Equation 5.11. The negative values are also handled by the on-center/off-center filter.

$$I_o^n = \begin{cases} I_o^n(x,y,c1) = \mathrm{OF}_x^n(x,y) \\ I_o^n(x,y,c2) = \mathrm{OF}_y^n(x,y) \\ I_o^n(x,y,c3) = \begin{cases} \mathrm{I}^n(x,y), \text{ if } I^n(x,y) - I^{n+1}(x,y) \neq 0 \\ 0, \text{ otherwise,} \end{cases} \end{cases} \tag{5.11}$$

where I^n is the input frame at index n, I_o^n is the output frame at index n with channels $c1$, $c2$ and $c3$, and OF_x and OF_y are the horizontal and vertical components of optical flow vectors, respectively. This frame-based method will allow an assessment of the importance of the added spatial information for HAR with CSNNs.

Composite Channels with Orientation and Amplitude (CCOA) This representation is another version of the composite channels representation, but we replace the gray scale information in the third channel with the average of the three (RGB) channels of the OA representation, as shown in Figure 5.6 (E), and represented in Equation 5.12.

$$I_o^n = \begin{cases} I_o^n(x,y,c1) = \mathrm{OF}_x^n(x,y) \\ I_o^n(x,y,c2) = \mathrm{OF}_y^n(x,y) \\ I_o^n(x,y,c3) = \begin{cases} \frac{\mathrm{OA}^n(x,y,R)+\mathrm{OA}^n(x,y,G)+\mathrm{OA}^n(x,y,B)}{3}, \text{ if } I^n(x,y) - I^{n+1}(x,y) \neq 0 \\ 0, \text{ otherwise,} \end{cases} \end{cases} \tag{5.12}$$

where I^n is the input frame of index n, I_o^n is the output frame with channels $c1$, $c2$ and $c3$, and OA^n is the RGB frame of optical flow amplitude and orientation, computed from Equations 5.9 and 5.10, as stated earlier. This method, compared to CC, allows us to assess the importance of spatio-temporal composite channels, instead of optical flow vectors only.

5.3.2 Shot-based methods

Edge grid (EG) This method groups the movement information of a video into a composite grid. We start by extracting the optical flow vectors from each two consecutive frames, as shown in Figure 5.7 (B). Then, the horizontal and vertical components of optical flow vectors OF_x^n and OF_y^n are combined into a single channel by retaining their maximum values $I_{OF}^n(x,y) = \max(\mathrm{OF}_x^n(x,y), \mathrm{OF}_y^n(x,y))$. Simultaneously, the edges of each frame are extracted using the Canny edge detection method [201], as shown in Figure 5.7 (C); we note I_{canny}^n the response of the Canny detector for frame I^n. The optical flow channel and the response of the Canny detector are multiplied pixel by pixel, so that edge locations are supplied with movement information in the resulting response: $I_{\text{moving\_edge}}^n(x,y) = I_{OF}^n(x,y) \times I_{\text{canny}}^n(x,y)$. This process is applied to $3.p.q$ frames to produce $3.p.q$ $I_{\text{moving\_edge}}^n$ frames, where p and q are the desired vertical and horizontal dimensions of the grid.

Then, EG cells $I_{o_{i,j}}$ are formed by regrouping three consecutive $I_{\text{moving\_edge}}^n$ images into a single, 3-channel frame, as shown in Figure 5.7 (D). Each EG cell ($I_{o_{i,j}}$) has the same size as the $I_{\text{moving\_edge}}$ images. EG cell construction is expressed in Equation 5.13:

$$I_{o_{i,j}}(x,y) = \begin{cases} I_{o_{i,j}}(x,y,c1) = I_{\text{moving\_edge}}^{n}(x,y) \\ I_{o_{i,j}}(x,y,c2) = I_{\text{moving\_edge}}^{n+1}(x,y) \\ I_{o_{i,j}}(x,y,c3) = I_{\text{moving\_edge}}^{n+2}(x,y) \end{cases} \tag{5.13}$$

where $n = 3.i.q + 3.j$, $i \in [0, p-1]$, and $j \in [0, q-1]$. Finally, the $p.q$ I_o frames are arranged into a grid of dimension $p \times q$, as shown in Figure 5.7 (E).

The edges represent significant local changes in intensity, and edges that contain motion can be interesting information for STDP-based CSNNs, since they require edges, as explained in Section 4.1.

Motion grid (MG) This method groups the movement information of several optical flow frames into a composite grid. Each optical flow frame is separated into four separate frames representing the displacement in four different directions, and put one after the other in the grid, as shown in Figure 5.8.The motion displacement in four different directions used in this method is represented in Equations 5.14, 5.15, 5.16, and 5.17:

$$M^l(x,y) = \frac{|\mathrm{OF}_x(x,y)| - \mathrm{OF}_x(x,y)}{2} \tag{5.14}$$

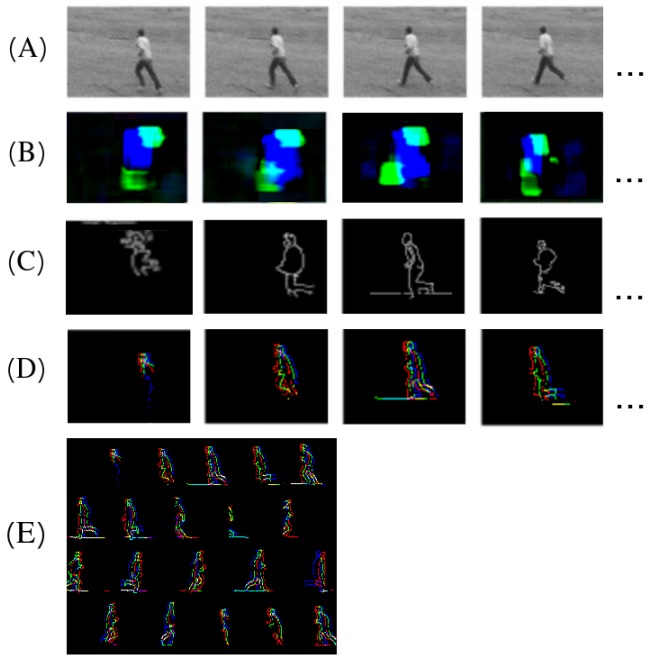


Figure 5.7: Constructing an edge grid includes: (A) taking a video sequence, (B) extracting the optical flow vectors between the consecutive frames, (C) extracting the canny edges of the original frames, (D) supplying edge locations with movement information by multiplying the response of the Canny detector with the optical flow data, and forming EG cells by regrouping three consecutive $I_{\text{moving\_edge}}$ into a three-channel frame, and (E) assembling these frames to form a $p \times q$ grid.

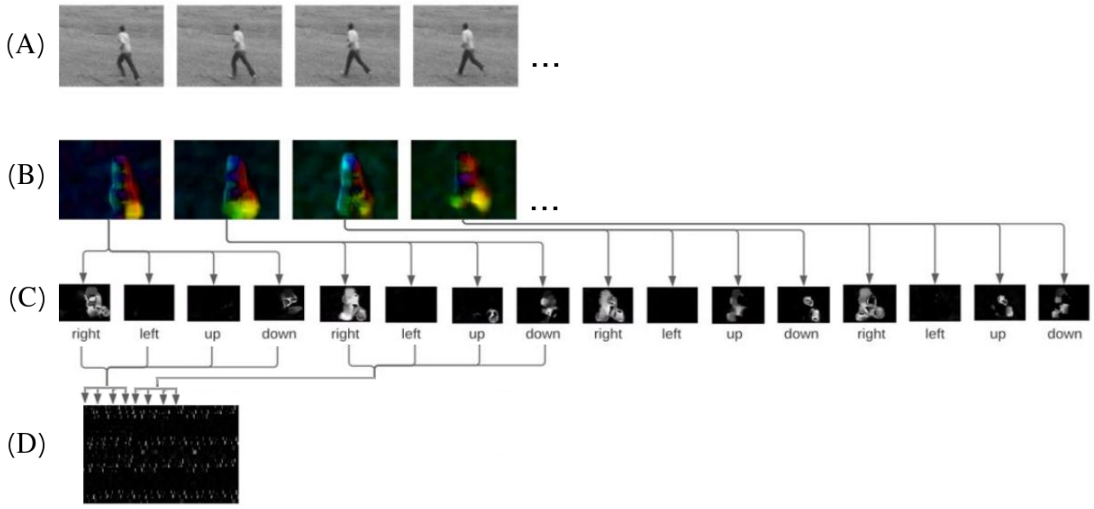


Figure 5.8: Constructing a motion grid includes: (A) taking a video sequence, (B) extracting the optical flow of the consecutive frames, (C) separating the motion into four different directions, and (D) assembling the grid.

$$M^r(x,y) = \frac{|\mathrm{OF}_x(x,y)| + \mathrm{OF}_x(x,y)}{2} \tag{5.15}$$

$$M^u(x,y) = \frac{|\mathrm{OF}_y(x,y)| - \mathrm{OF}_y(x,y)}{2} \tag{5.16}$$

$$M^d(x,y) = \frac{|\mathrm{OF}_y(x,y)| + \mathrm{OF}_y(x,y)}{2} \tag{5.17}$$

where OF_x and OF_y represent the horizontal and vertical components of the optical flow vectors, $M^u(x,y)$ is the upwards displacement at pixel (x,y), $M^d(x,y)$ the downwards displacement, $M^l(x,y)$ is the displacement to the left, and $M^r(x,y)$ is the displacement to the right. The input video, consisting of $p.q$ frames, will yield a total of $4.p.q$ frames after the separation of each frame into four different motion directions. Subsequently, these frames are combined into a motion grid with dimensions $4.p \times q$, as shown in Figure 5.8 (D), and represented in the matrix 5.18.

$$MG = \begin{bmatrix} M^r_{o_{1,1}} M^l_{o_{1,1}} M^u_{o_{1,1}} M^d_{o_{1,1}} & \dots & M^r_{o_{1,q}} M^l_{o_{1,q}} M^u_{o_{1,q}} M^d_{o_{1,q}} \\ \vdots & \ddots & \vdots \\ M^r_{o_{p,1}} M^l_{o_{p,1}} M^u_{o_{p,1}} M^d_{o_{p,1}} & \dots & M^r_{o_{p,q}} M^l_{o_{p,q}} M^u_{o_{p,q}} M^d_{o_{p,q}} \end{bmatrix} \tag{5.18}$$

STDP
$\eta_w = 0.1$, $\tau_{\text{STDP}} = 0.1$, $W \sim U(0,1)$
Threshold Adaptation
$\hat{t}_{KTH} = 0.75$, $\text{th}_{\text{min}} = 1.0$, $\eta_{\text{th}} = 1.0$, $\hat{t}_{Weizmann} = 0.75$, $V_{\text{th}}(0) \sim \mathcal{N}(8,1)$
Difference-of-Gaussian
$\sigma_1 = 1.0$, $\sigma_2 = 4.0$, s $= 7.0$

Table 5.1: The meta-parameter values used in the experiments.

5.4 Evaluation

5.4.1 Datasets and implementation details

The KTH and Weizmann datasets, previously mentioned in Section 2.1, are used to evaluate these methods, following the protocols described in Section 4.9. Each sample from both datasets is made up of 10 frames for frame-based representations, 45 frames are used to collect 15 EG cells, and construct a 5×3 grid for the EG representation. 48 optical flow frames to construct a 4×12 grid for the MG representation, where each frame is horizontally spread into 4 direction frames, resulting in a 16×12 grid.

Sum pooling is used at the output of the CSNN to limit the spatial size of the feature maps before classification. No sum pooling is done in the temporal dimension.

For the experiments where frame subtraction is applied, we choose Equation 5.4 for these experiments, where we set all negative values to their absolute value.

The meta-parameter values used in these experiments are presented in Table 5.1. Different numbers and sizes of convolutional kernels were tested, but we only report the most suitable values: 128 convolutional kernels of size 5×5, with no padding and a stride of 1.

5.4.2 Baseline performance evaluation of 2D CSNNs

A baseline is needed in order to accurately assess the performance of the CSNN with each type of motion representation. Therefore, we test the performance of the CSNN on raw videos without any pre-processing. We use early and late fusion techniques in order to conserve the temporal components of the videos. Table 5.2 (A) shows these results with a sum pooling that renders the output feature map size as 2×2, as used for image analysis in [31]. By decreasing the sum pooling severity to one that renders the final feature map size as 20×20, we get the results in Table 5.2 (B).

Dataset	KTH	Weizmann	KTH + FS	Weizmann + FS
(A) Feature map size 2×2				
Early Fusion	24.50	22.11	30.52	20.73
Late Fusion	26.38	21.03	**35.26**	**23.58**
(B) Feature map size 20×20				
Early Fusion	58.95	53.76	58.02	**63.76**
Late Fusion	65.12	59.32	**68.52**	62.28

Table 5.2: Classification rate in % using early and late fusion with the KTH and Weizmann datasets as raw frames, and with frame subtraction (FS).

The most significant conclusion drawn from these results is the substantial impact that sum pooling can have on the performance. The results presented in Table 5.2 (B) consistently outperform those in Table 5.2 (A), where the sum pooling renders the feature maps significantly smaller, resulting in a substantial loss of detail. Hence, it is evident that the excessive dimension reduction caused by sum pooling should be avoided when aiming to reduce the size of the output feature maps.

Another noteworthy finding is that using frame subtraction (FS) leads to improved classification rates when employing late fusion, although this is not always the case with early fusion. For instance, in the Weizmann dataset (Table 5.2 (A)), FS results in a decrease in the classification rate. Furthermore, it was observed that late fusion consistently outperforms early fusion with the KTH dataset with all experiments, sometimes exhibiting an advantage of up to 10 percentage points.

With regard to the Weizmann dataset, late fusion demonstrates competitive performance with early fusion, surpassing it in most scenarios. The best results on both the KTH and Weizmann datasets are achieved when using frame subtraction in combination with late fusion. The only exception is the last column in Table 5.2 (B), where early fusion with frame subtraction yields the best performance.

These initial results leave us leaning towards adopting late fusion and frame subtraction in future experiments.

5.4.3 Performance analysis of 2D CSNNs using static representations of motion

In this section, we showcase the performance of a 2D CSNN using the static representations of motion that were previously presented in Section 5.3. The frame-based methods are tested with both early and late fusion, while the shot-based methods

are considered as a type of early fusion, as the motion information is integrated into a grid prior to processing. The results in Table 5.3 (A) show the performance of the CSNN with the static representations of motion, with a sum pooling that reduces drastically the spatial size of feature maps to 2×2. Table 5.3 (B) presents the same results but with a larger sum pooling output size, which results in feature maps of size 20×20.

These results were designed using frame subtraction before all the representations except the edge grid, which is based on edge extraction, making FS less effective. The results in Table 5.3 (A) show that the shot-based methods outperform frame-based methods with the KTH dataset, as shown with EG [63.01% - 77.93%], compared to DXDY [28.70% - 68.36%], while the performance of the CSNN with these methods falls short with the Weizmann dataset, as shown with MG [28.86% - 66.21%] in comparison to frame-based methods, as shown with OA [50.42% - 71.88%], because the Weizmann videos are sparser and shorter than the KTH dataset videos, which leads to smaller grid sizes in EG and MG. This indicates that the CSNN performs well with representations that contain at least one full cycle of motion, and that more training samples than the number provided by the Weizmann dataset are needed.

To further illustrate this point, following the initial experiments, we conducted additional experiments for the EG to investigate the impact of increasing the number of training samples. We replicated the experiment presented in Table 5.3 (B), where we used the Weizmann dataset represented with EG, and achieved a classification rate of 55.08%. We increased the number of training samples from 83 to 250. This data augmentation process was accomplished by collecting supplementary samples from the same video. As mentioned earlier, we build a sample clip by skipping two frames every three frames, starting from the first frame of the video. To build more samples, we follow the same process, but starting from a random position, producing different clips from the same video. Training the network with the augmented dataset has increased the classification rate to 67.44%. It shows that more training samples can indeed ameliorate the performance of shot-based methods with the Weizmann dataset.

Dataset	KTH		Weizmann	
Fusion Method	Early	Late	Early	Late
(A) Feature map size 2×2				
DXDY	26.85	28.70	12.86	11.11
OA	41.05	45.83	**50.42**	44.44
CC	45.78	54.21	36.75	30.68
CCOA	57.87	46.60	45.70	50.91
EG	63.01	-	43.83	-
MG	**77.69**	-	28.86	-
(B) Feature map size 20×20				
DXDY	58.49	68.36	54.25	67.95
OA	54.32	61.57	61.88	**71.88**
CC	62.81	66.82	63.25	70.43
CCOA	52.31	67.59	65.21	64.99
EG	**77.93**	-	55.08	-
MG	77.78	-	66.21	-

Table 5.3: Classification rate in % using early fusion and late fusion with the KTH and Weizmann datasets pre-processed with frame subtraction (except for EG) and the static representations of motion. These representations are horizontal and vertical displacements (DXDY), orientation and amplitude (OA), composite channels (CC), composite channels with orientation and amplitude (CCOA), edge grid(EG), motion grid (MG).

5.4.4 Effects of frame subtraction on the performance of 2D CSNNs

The previous results were conducted based on the initial benchmark. This section evaluates the effects of frame subtraction on the results by performing an ablation study. Therefore, we re-run all experiments, skipping the FS step, except for EG, which was not evaluated with FS in Table 5.3.

Table 5.4 contains the performance of the CSNN with the static representations of motion without frame subtraction, and with a sum pooling that renders the feature map size as 20×20. These results reveal two observations.

- Our first observation is that there is an enhancement in the performance of

the CSNN when applied to the KTH dataset in conjunction with late fusion. This indicates that FS excessively discards spatial information like texture, particularly with the CCOA method, which shows an increase of around 5 *p.p.* in classification rate with the KTH dataset when omitting FS. This is due to the discard of texture by FS, which decreases the performance of these optical flow-based methods: optical flow needs textured patches to trace pixels, so FS makes it more difficult to produce accurate motion vectors.

- Our second observation pertains to a notable decrease in performance across the Weizmann dataset when FS is omitted. This decline is primarily attributed to FS's ability to eliminate background noise, which is prevalent in the Weizmann dataset but not as pronounced in the KTH dataset. As illustrated in Figure 5.9, the edges of this noisy background are detected by the on-center/off-center filter and are learned by the CSNN. This leads to a reduction in performance with the Weizmann dataset. The large pixel values associated with these background edges, as shown in Figure 5.9 (c), result in latency coding transforming these values into early spikes, which are more likely to prompt the neuron to spike before the integration of spikes encoding pixels of the subject in action.

These observations may appear to present a slight contradiction to the results in Table 5.2, where using FS was beneficial, but this discrepancy arises because the table featured raw videos rather than optical flow-based static representations of motion.

Dataset	KTH		Weizmann	
Fusion Method	Early	Late	Early	Late
DXDY	55.25	70.52	56.84	64.36
OA	32.56	71.76	52.69	64.96
CC	45.37	66.67	57.32	64.62
CCOA	45.83	72.38	59.32	63.76
MG	74.07	-	57.69	-

Table 5.4: Classification rate in % using early fusion and late fusion with the KTH and Weizmann datasets as pre-processed with the static representations of motion, without prior FS filtering.

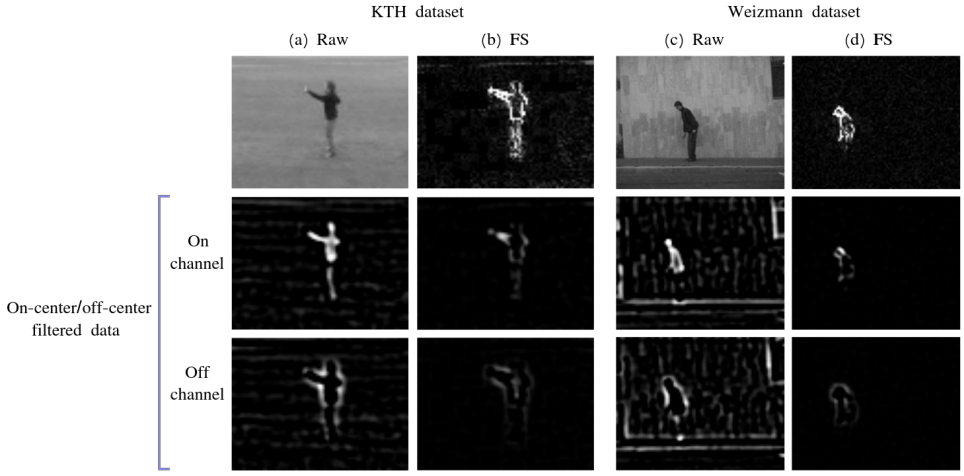


Figure 5.9: The information filtered by an on-center/off-center filter with the KTH dataset using (a) raw frames and (b) frame subtraction, and the Weizmann dataset using (c) raw frames and (d) frame subtraction.

5.5 Conclusion

This study was carried out in order to give an assessment of the effect of different data representations on spatio-temporal feature learning. Although the performance of these CSNNs is still not competitive with that of non-spiking ANNs, this assessment provides a solution for 2D CSNNs to process video data, and gives a better understanding of the type of information required for a 2D CSNN to extract spatio-temporal features. In addition, it is crucial to highlight that our spatio-temporal features were extracted without supervision, distinguishing them from their non-spiking counterparts, making it more difficult to reach the performance levels of fully-supervised solutions.

The result of testing these representations on an SNNs trained with STDP yields several conclusions. The first conclusion is that the best performance is recorded with shot-based methods when there is more than one cycle of motion, like in the case of the MG and EG representation with KTH dataset videos (77.93% for EG and 77.78% for MG in Table 5.3). The same representations gave inferior results using the Weizmann dataset, because the actions in most videos are performed only once, and the videos are not long enough to fill this grid with multiple cycles of motion. However, these methods outperform frame-based methods when they represent more than one complete cycle of the action being performed. Therefore, shot-based methods serve as a good starting point in improving human action recognition with

SNNs.

Static representations of motion, in addition to early and late data fusion methods, permit 2D CSNNs to perform spatio-temporal feature extraction, which is necessary for HAR. Our second conclusion is that velocity distribution of the moving components in the videos is a very important aspect in action classification. When employing a pre-processing method for motion modeling that emphasizes this aspect, such as CCOA, which includes information comprising optical flow vectors, the classification system can achieve higher classification rates (72.38% in Table 5.4 with the KTH dataset) than frame-based methods that include gray scale information, like CC.

Our third conclusion is that FS decreases the performance of optical flow-based methods, because optical flow needs textured patches to trace pixels. However, frame subtraction is still useful when the dataset contains a significant amount of background noise.

These static representations of motion are all based on the Farneback dense optical flow. As a result, they inherit the computational complexity of optical flow, which is $\mathcal{O}(l \cdot w \cdot h)$, where l represents the number of levels for pyramid calculations, and w and h denote the width and height of the input, respectively. Any factors that are negligible compared to h and w are disregarded. This analysis is grounded in the OpenCV implementation.

The methods presented in this chapter expand upon this foundation, each introducing additional computations. These additional computations do not increase the theoretical complexity, but do add some overhead to the computation time. The method requiring the fewest computations is the DXDY method, as it directly encodes the horizontal and vertical components of the optical flow as raw vectors. Next is the OA method, which involves additional computations for the conversion from HSV to the RGB color space. The CC method also directly encodes the horizontal and vertical components of the optical flow as raw vectors in its first two channels, but it also preserves the grayscale data of the moving part of the subject in the third channel. This entails the extra computation of frame subtraction to identify the mask that will retain the moving part by discerning the non-zero values and then recording their locations in the third channel of the frame. Subsequently, the CCOA method records the average value of optical flow in the third channel instead of grayscale data. This necessitates even more computations.

The MG method requires optical flow extraction, followed by the separation of this optical flow into four different directions, and the copy of this information at their location in the grid. However, since it is a shot-based method, the number of frames used depends on the size parameters of the motion grid. This MG method involves

fewer steps to construct than the EG method, which requires optical flow extraction multiplied by the result of Canny edge detection, making it more computationally expensive.

We provide the computational time measurements for each method in Python with NumPy and OpenCV on the 600 videos of the KTH dataset in Table 5.5. This table shows the methods and their associated total time to process 600 15-frame video samples, averaged over 100 repetitions:

methods	average time (seconds)	overhead (%)
Raw Images → OF	35.06	-
Raw Images → OF → DXDY	36.31	3.56
Raw Images → OF → OA	36.84	5.07
Raw Images → OF → CC	36.18	3.19
Raw Images → OF → CCOA	42.97	22.56
Raw Images → OF → MG	37.26	6.27
Raw Images → OF → EG	43.08	22.87

Table 5.5: The average processing time (in seconds) of 600 15-frame videos for various static representations of motion, averaged over 100 runs, including overhead (in %) relative to optical flow method.

Based on the data presented in Table 5.5, it is evident that there is a limited increase in processing time across the various methods compared to the baseline OF computation (around 35 seconds). This indicates that the additional computations introduced by each method slightly impact the overall computation time.

Chapter 6

3D Convolutional Spiking Neural Networks

Spatio-temporal feature extraction with STDP-based 2D CSNNs has been explored in Chapter 5. These networks are originally designed for static data analysis, and have been paired with motion modeling techniques and adapted for video analysis tasks. Throughout our investigation, we observed that to extract spatio-temporal information using these models, additional non-spiking motion modeling steps are necessary. However, these non-spiking motion modeling methods are computationally costly, and implementing them on neuromorphic hardware would be challenging. Additionally, the features learned with such methods are derived from pre-processed data, which alters the unbiased dynamic spatio-temporal patterns we aim to learn. One significant obstacle is the need to minimize or eliminate the reliance on non-spiking pre-processing techniques for modeling motion with STDP-based CSNNs, all while ensuring the capability to extract relevant spatio-temporal features from videos.

Fully spiking solutions are more promising, cost-effective methods for video analysis. Building upon the insights gained in the previous chapter concerning the performance of STDP-based 2D CSNNs for HAR, we now turn our attention to the exploration of spatio-temporal STDP-based CSNNs, specifically 3D CSNNs. In Chapter 3, we observed that unsupervised STDP involves local computations. Therefore, using unsupervised STDP could potentially facilitate the extension of a conventional 2D CSNN, which processes information in spatial dimensions, into a spatio-temporal 3D CSNN architecture for videos, which processes information in both spatial and temporal dimensions.

In this chapter, we address building a 3D CSNN model trained in an unsupervised

manner with the STDP learning rule. This model can learn spatio-temporal patterns found in videos by sliding its convolutional kernels in the temporal dimension as well as in the spatial ones. This extra dimension does increase the number of parameters of a 3D CSNN compared to a 2D CSNN, but its computational cost is still very low compared to non-spiking CNNs. To the best of our knowledge, Unsupervised STDP-based 3D CSNNs remain unexplored in the context of video analysis, even though they offer a cost-effective processing approach in theory.

The 3D CSNN introduced in this chapter can extract spatio-temporal features from videos naturally without relying on non-spiking pre-processing methods that are difficult for hardware implementation like the ones in Section 5.3. We validate this in the context of HAR, and we compare this model to its 2D equivalent in order to reach an accurate assessment of their performance. Then we list the benefits and drawbacks of each method. The main contributions of this chapter are summarized as follows:

- we present a spiking model for 3D convolution that allows learning spatio-temporal patterns with STDP in an unsupervised manner;
- we include this 3D convolution model into a state-of-the art spiking architecture for unsupervised feature learning;
- we evaluate and compare the performance of 2D and 3D CSNNs on both of the KTH and Weizmann action recognition datasets, as raw videos and after motion information extraction;
- we give an analysis of the effects of the main hyperparameters on the performance of a 3D convolutional SNN.

6.1 Spiking 3D convolutions

The kernels of a 3D CSNN can slide along the temporal dimension of the input tensor in addition to the spatial ones, and extract spatio-temporal features that correspond to space-time patterns occurring in the input video, as illustrated in Figure 6.1. This figure illustrates a 3D convolutional kernel temporally split, capturing the motion occurring in a video at different times, in principle.

The virtual movement of 3D convolutional kernels can be visualized as going through the three dimensions, width, height, and temporal depth, in steps determined by the stride in each dimension. Each neuron in the convolution output processes a local volume of the data sample in space and time. A 3D convolution

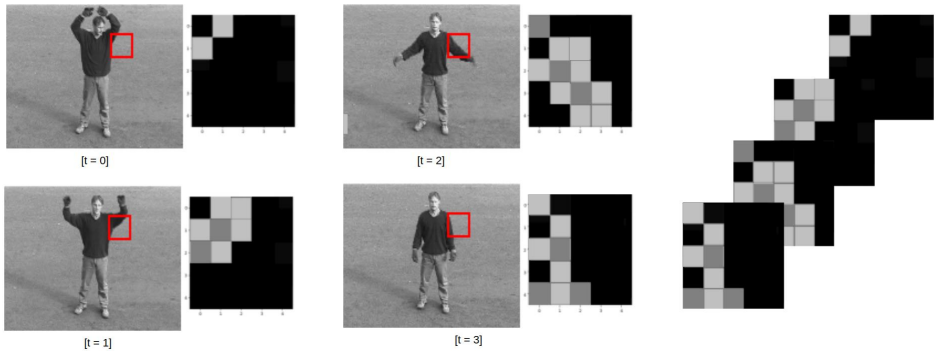


Figure 6.1: An idealization of a 3D kernel of size $5 \times 5 \times 4$ capturing a space-time pattern from a video clip.

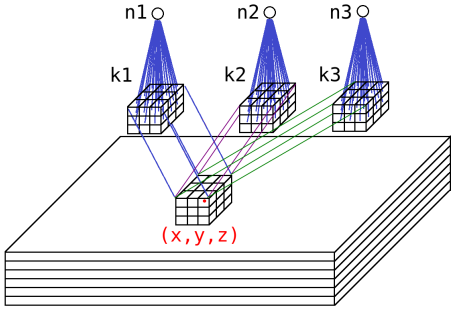


Figure 6.2: The neighborhood of a neuron at (x, y, z) is a volume, and n1, n2, and n3 are competing neurons at the same location during the training phase.

layer is therefore defined by a set of f_k trainable kernels, with sizes $f_w \times f_h \times f_{\text{td}}$, where f_w and f_h represent the width and height of a kernel respectively, and f_{td} is the temporal size of the kernel. Similarly to 2D convolution, but with the extra dimension, each neuron of a layer is connected to $f_w \times f_h \times f_{td}$ inputs from the previous layer, as illustrated in Figure 6.2. The coordinates of a neuron or a spike in this 3D model are now x, y, z, and k. 3D spiking convolution can be formalized as:

$$V_{m_{x,y,z,k}}(t) = \sum_{i \in \mathcal{E}} W_{n(x_i),j(y_i),m(z_i),k_i,k} \times f_s(t - t_i) \tag{6.1}$$

$$f_s(t - t_i) = \begin{cases} 1 & \text{if } t \geq t_i \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

where $V_m(t)$ is the neuron membrane potential at time t, x, y, z and k are the

coordinates of the neuron in the width, height, temporal and channel dimensions, respectively, W is the trainable synaptic weight tensor, $n()$, $j()$, and $m()$ are functions that map the location of the input to the corresponding location in the weight tensor in the spatial and temporal dimensions, and f_s is the kernel of spikes represented in Equation 4.6.

Similarly to Section 4.2, when the membrane potential $V_{m_{x,y,z,k}}(t)$ crosses the threshold potential $V_{\text{th}}(t)$, STDP learning occurs, so Equations 4.9 (page 65) is applied to update the synaptic weights, and the Equations 4.12 4.11 4.10 (page 66) are applied to update the thresholds of the network. It is important to note that the threshold adaptation rule and the biological STDP rule are the same in both 2D and 3D architectures, as they are independent of the input and kernel dimensions.

The training and testing processes are then the same as in the case of 2D convolution, explained in Chapter 4. With spiking 2D convolutions in [31], a fixed number of patches were randomly selected for training the 2D kernels, as mentioned in Section 4.4. However, this fixed number of patches is not suitable for the variation in input sample size for HAR. Therefore, for each input sample, we choose a specific number n_{sampling} of spatial locations in order to ensure that the number of random patches is enough to cover the input sample, the determination of n_{sampling} is based on the dimensions of the input sample and the convolutional kernel, as shown in Equation 6.3:

$$n_{\text{sampling}} = \frac{2 \times l_w \times l_h}{f_w \times f_h} \tag{6.3}$$

For 3D CSNNs, we need to update this sampling number according to Equation 6.4 to account for the temporal dimension of size l_{td}. We increase the sampling constant to 3 in order to insure a large enough sampling number to cover the temporal dimension:

$$n_{\text{sampling}} = \frac{3 \times l_w \times l_h \times l_{td}}{f_w \times f_h \times f_{td}} \tag{6.4}$$

6.2 The network pipeline

The core pipeline in this chapter consists of a multi-layer CSNN, which is made up of convolution and max pooling layers. These spiking networks are trained layer-wise, i.e., each layer is trained independently, starting from the first layer. Then the weights of the trained layer are set aside during the training of the subsequent layers. During the testing phase, all neurons are active and layers are processed sequentially, i.e., all the input spikes of one layer are processed before processing the

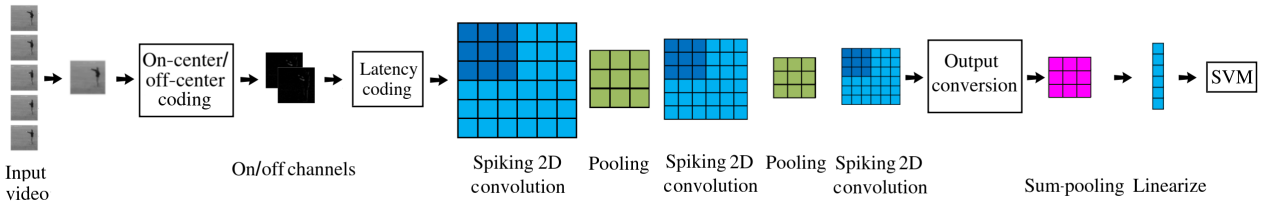


Figure 6.3: Network topology with 2D convolutions.

next layer. So, the output feature maps of one layer provide the input spike trains for the subsequent layer.

A 2D version of this pipeline is illustrated in Figure 6.3. The process begins with the analysis of the input video on a frame-by-frame basis. Each frame undergoes on-center/off-center filtering. Subsequently, latency coding is applied to convert the filtered data into spikes. These spikes are then processed through spiking convolution and max pooling layers, to give spiking output feature maps. Then, the output converter from Section 4.7, Equation 4.13 (page 69) is used to convert the spikes back into continuous values. These output feature maps are reduced in size through spatial sum pooling, for more efficient handling by the classifier.

Since 2D convolutions only process the spatial dimensions of the input, they discard any temporal information. Therefore, these output feature maps are spatial. In this work, instead of resorting to costly pre-processing, and to keep the results comparable with those of a 3D CSNN, spatio-temporal feature extraction with 2D CSNNs is achieved by replacing the spatial sum pooling in Figure 6.3 by spatio-temporal sum pooling. This joins the output feature maps over the temporal dimension into one feature map per kernel, and simultaneously reduces the spatial size of these output feature maps. This creates a sort of late fusion that integrates feature maps across the entire video sequence, as shown in Figure 6.4. Finally, the resulting data is linearized and input into a classifier.

The 3D CSNN takes a video sample as input, in the form of a 4D tensor where the temporal depth corresponds to the number of frames per video, as shown in Figure 6.5. The same general pipeline is used for the 3D CSNN, where the video sample undergoes on-center/off-center filtering, then latency coding, and then the processing through convolution and max pooling layers. The difference is that spiking 2D convolution layers are replaced by spiking 3D convolution layers that process the space-time volume directly, and the max pooling layers apply to the space-time volume, over both spatial and temporal dimensions. Our pipeline for the multi-layer 3D CSNN, is shown in Figure 6.6.

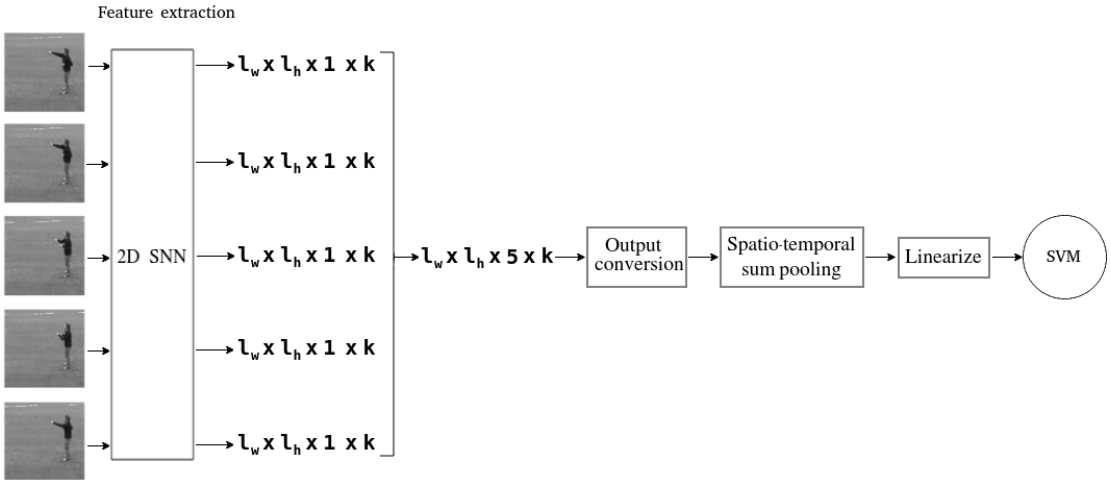


Figure 6.4: Spatio-temporal feature extraction with a 2D architecture from a five-frame video clip, and with k kernels.

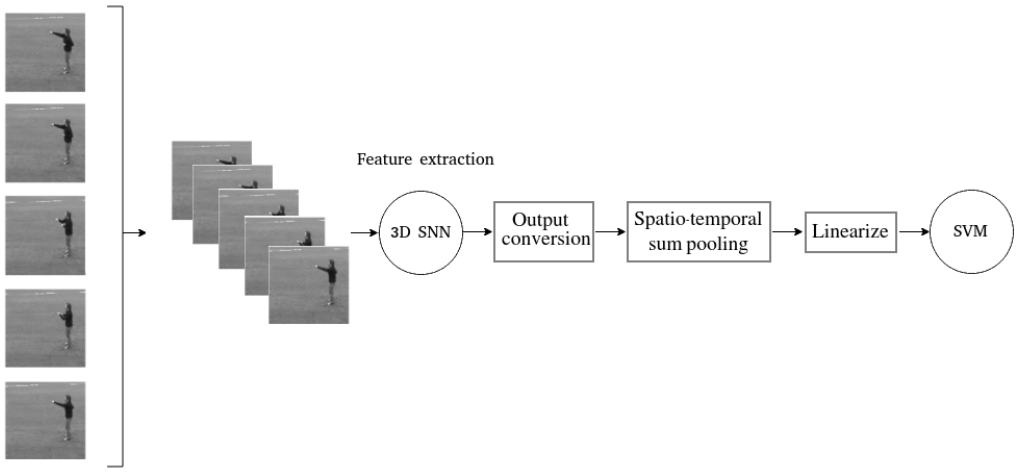


Figure 6.5: Introducing an input video for spatio-temporal feature extraction using a 3D architecture.

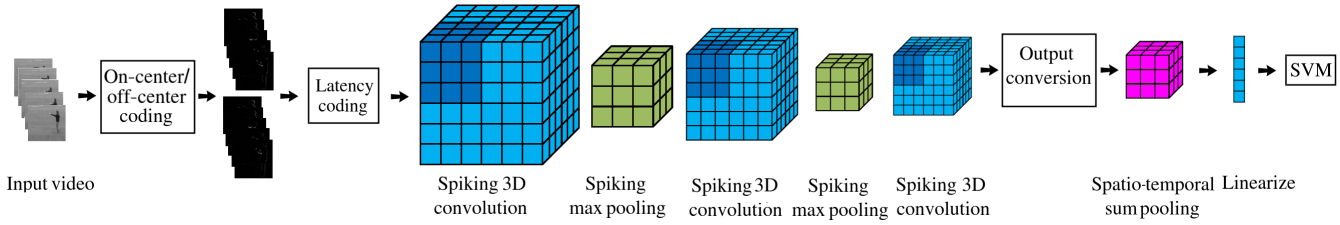


Figure 6.6: Network topology with 3D convolutions.

The dimensions of the output feature maps of this network are also decreased using sum pooling, and then linearized, as shown in Figure 6.6. We use temporal sum pooling at the output of our 3D CSNN to ensure that the outputs of both networks are of similar size. This is because, in Chapter 5, we saw that the feature map size can affect the performance of the classifier. Therefore, for a fair comparison, both networks need to have similar output feature map sizes. This output is introduced into a support vector machine (SVM) with a linear kernel, which performs the action classification.

6.3 Evaluation

6.3.1 Datasets and implementation details

Experiments are performed on the KTH [54] and Weizmann [55] datasets, previously mentioned in Section 2.1, following the protocols described in Section 4.9. We report the experiments with 8 and 20 frames per video in order to have a comprehensive comparison between the performance of spiking 2D convolutions and spiking 3D convolutions with different video clip sizes. We skip one frame between each two consecutive frames in order to make sure to capture a full cycle of the performed action. Similarly to Chapter 5, we scale down the frame sizes of both datasets to half of their original sizes for processing speed reasons. We measure the classification accuracy (in %) on the test set for all experiments. Each experiment was run three times, and we report the average accuracy over the three runs.

In this chapter, to assess the individual performance of each layer within multi-layer architectures, we incorporate an SVM after each layer solely for evaluating the features learned by that layer.

The meta-parameter values used in these experiments are presented in Table 6.1. No padding is used for the convolutions. The convolutional kernels of 2D layers use a stride of 1 in all dimensions, while those of 3D layers use a stride of 1 in spatial

STDP
$\eta_w = 0.1$, $\tau_{\text{STDP}} = 0.1$, $W \sim U(0, 1)$
Threshold Adaptation
$\hat{t}_{\text{KTH}} = (0.65, 0.3, 0.1)$, $\text{th}_{\text{min}} = 1.0$, $\eta_{\text{th}} = 1.0$, $\hat{t}_{\text{Weizmann}} = (0.75, 0.55, 0.15)$, $V_{\text{th}}(0) \sim \mathcal{N}(8, 1)$
Difference-of-Gaussian
$\sigma_1 = 1.0$, $\sigma_2 = 4.0$, $\text{s} = 7.0$

Table 6.1: The hyperparameter values used in the experiments. The $\hat{t}$ tuple correspond to the values of $\hat{t}$ used for each layer, respectively.

dimensions, and a stride of 2 in the temporal dimension. The max pooling layers use a kernel of size 2×2 and a stride of 2×2 for the 2D setup, and a kernel of size $2 \times 2 \times 2$ and a stride of $2 \times 2 \times 2$ for the 3D setup.

The videos are presented to the SNNs either as raw frames, or pre-processed with the frame subtraction method described in Equation 5.4 (page) 78. We use this method in order to see the ability of the SNNs to classify input information that consists purely of motion, and to assess the performance of 3D CSNNs without the issue of the noisy background patterns observed with the Weizmann dataset mentioned in the previous chapter. In the rest of this section, experiments are performed on raw frames unless otherwise specified.

In the following, we conduct experiments aimed at providing comprehensive comparisons between 2D and 3D CSNNs in various settings. First, we explore the impact of the target timestamp on the performance of our single-layer and multi-layer 2D and 3D CSNN architectures. Following this, we investigate the influence of kernel size and video length on multi-layer 2D and 3D CSNN architectures. Finally, we examine the effects of frame subtraction on the performance of these models.

6.3.2 Spike selectivity with target timestamp threshold adaptation

The threshold adaptation method detailed in Section 4.5 (page 66) requires choosing a target timestamp $\hat{t}$ towards which the firing time of the neuron must converge. This target timestamp impacts the patterns learned by the neurons [83]. If the neuron thresholds are low, the learned patterns would have only to integrate a few spikes that represent the most salient part of the sample, while if the neurons had large thresholds, more spikes would be integrated, and more detailed patterns can be

Dataset	KTH		Weizmann	
$\hat{t}$	0.1	0.65	0.15	0.75
2D CSNN	56.84	**56.94**	47.26	**48.63**
3D CSNN	55.09	**57.56**	**51.45**	49.49

Table 6.2: Average classification rates in % of KTH and Weizmann dataset (8-frame videos) over 3 runs, as a function of different $\hat{t}$ values with 2D and 3D single-layer architectures.

learned [83]. Therefore, this target timestamp value, which affect the convergence of the thresholds of the neurons, allows us to be more or less selective with the spikes that are used in the learning. This mechanism not only allows us to control spike selectivity, but it also has another advantage, which is that it reduces the impact of initial values for neuron thresholds, which are randomly initialized [83]. However, the optimal target timestamp value can vary with different types of input information, and is selected for a given dataset using trial and error, with regard to a validation set.

Training single-layer CSNN architectures with raw videos filtered by an on-center/off-center filter to generate edges as input yields the results shown in Table 6.2. The convolutional kernel sizes used are 5×5 for 2D CSNNs and $5 \times 5 \times 2$ for 3D CSNNs. These results show the effect of having high or low spike selectivity on the resulting classification rates, where we use small and large values of target timestamps with single-layer 2D and 3D architectures. These initial results show that 3D CSNNs outperform 2D CSNNs in most cases, but severe spike selectivity limited the benefits of 3D CSNNs with the KTH dataset, where using a $\hat{t} = 0.1$ gave a better result with 2D CSNNs in Table 6.2. The KTH dataset is characterized by large motion patterns, which makes a larger $\hat{t}$ a more suitable option, because integrating fewer spikes makes it more difficult to properly train the larger 3D convolutional kernel.

On the other hand, a similar severe spike selectivity ($\hat{t} = 0.15$), resulted in benefiting the performance of 3D CSNNs with the Weizmann dataset. This is due to the large ratio of noisy background edges to relevant edges in the Weizmann dataset. There are much less noisy background edges in the KTH dataset, as shown in Figure 6.7, so it does not benefit from severe spike selectivity. This increased spike selectivity benefited the classification with the Weizmann dataset by only taking the most salient part of the sample into account, which reduces the effect of noise.

We also explore the effect of $\hat{t}$ on multi-layer architectures, with both 2D and 3D CSNNs. We start by investigating the effect of high selectivity of spikes in the

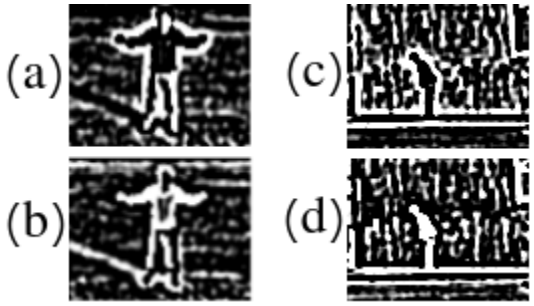


Figure 6.7: The edges extracted by the on-center/off-center filter for a KTH and a Weizmann frame. (a) The KTH frame on channel, (b) The KTH frame off channel, (c) The Weizmann frame on channel, and (d) The Weizmann frame off channel

first layer, by choosing a small $\hat{t}$. This leads to solely prompting some neurons with low thresholds to fire in response to the earliest received spikes, which represent the stronger edges. Therefore, it is not giving a chance for neurons with higher thresholds to fire, which leads to completely ignoring the details represented by the parts of the sample with weaker edges. This high spike selectivity degrades the learning in subsequent layers, because too few spikes can be integrated. $\hat{t}$ needs to be large enough to promote neural activity and learning in subsequent layers. Therefore, one might use a high $\hat{t}$ in all the layers with the KTH dataset, in order to capture large patterns throughout all layers. This results in oversaturated feature maps, as shown in Figure 6.8, which cause inconsistent performance with the KTH dataset, and decrease classification rates in subsequent layers, as shown in Table 6.3. However, increasing the selectivity over layers, by decreasing $\hat{t}$, promotes sparsity of the learned features, and improves the performance throughout the layers, as shown in Table 6.3. Therefore, with multi-layer CSNNs, the values of $\hat{t}$ must be large enough in initial layers, and then gradually decreased in order to promote the sparsity of the output feature maps.

Therefore, for the rest of the experimental procedure, the $\hat{t}$ values for the KTH dataset will be $(0.65, 0.3, 0.1)$ for the three layers of the 2D and 3D architectures. Similarly, the $\hat{t}$ values for the Weizmann dataset will be $(0.75, 0.55, 0.15)$ for the three layers of the 2D and 3D architectures. These values were chosen by an exhaustive trial-and-error process, using a validation set.

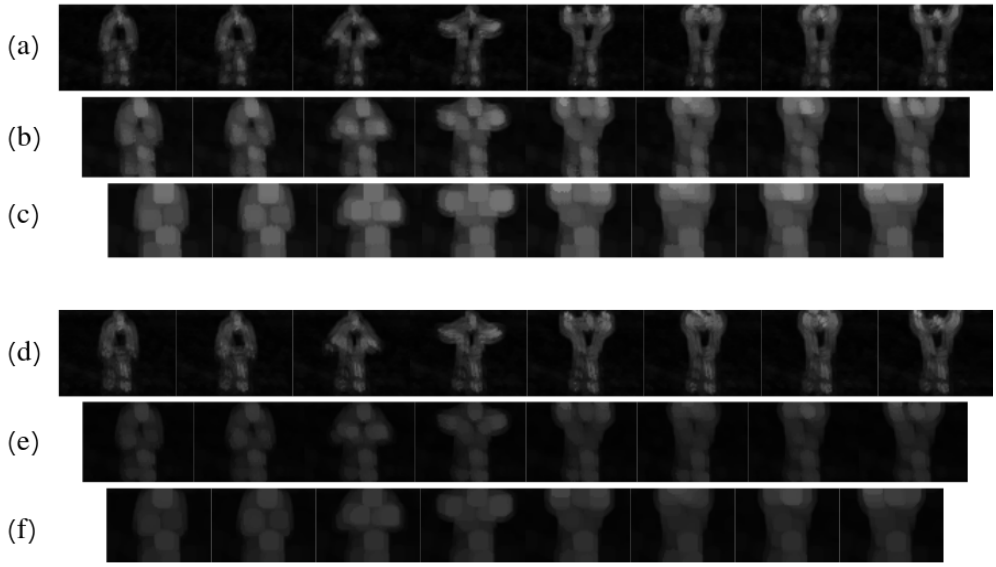


Figure 6.8: The feature maps that result from processing a KTH clip, frame by frame, with (a) $\hat{t} = 0.65$ in the first layer, (b) $\hat{t} = 0.65$ in the second layer, (c) $\hat{t} = 0.65$ in the third layer, (d) $\hat{t} = 0.65$ in the first layer, (e) $\hat{t} = 0.3$ in the second layer, and (f) $\hat{t} = 0.1$ in the third layer.

	$\hat{t} = (0.65, 0.65, 0.65)$			$\hat{t} = (0.65, 0.3, 0.1)$		
Layers	Conv1	Conv2	Conv3	Conv1	Conv2	Conv3
2D CSNN	55.89	53.80	52.67	59.72	59.10	62.35
3D CSNN	55.47	52.60	55.21	56.33	61.42	**63.43**

Table 6.3: Average classification rates in % on the KTH dataset (8-frame videos) over 3 runs, as a function of $\hat{t}$ values for three convolutional layers, with 2D and 3D multi-layer architectures.

6.3.3 Varied kernel sizes and video lengths

The convolutional kernel size has a direct effect on the learned features, so, in this section, we present the classification rates obtained using different convolutional kernel sizes with both 2D and 3D architectures. Tables 6.4 and 6.5 show the results obtained on KTH using 8 frames per video and 20 frames per video, respectively.

In Table 6.4, the classification rates obtained with a 3D convolutional SNN are slightly higher than those obtained with a 2D convolutional SNN architecture. The best classification rate is 63.43%, obtained with a 3D model that has a kernel size of $5 \times 5 \times 2$; this is only slightly higher than the classification rate of 62.35% obtained with a 2D CSNN that has a kernel size of 5×5. However, Table 6.5 shows that a larger video length decreases the performance of 2D CSNNs. This is partially due to the information saturation that results from pooling the features extracted by the 2D CSNN: it results in a sample that is harder to classify for the SVM than a sample made up of fewer frames. Therefore, we can deduce that 2D CSNNs are better suited for shorter video clips. On the other hand, 3D convolution does not have this problem, because the temporal information extraction does not depend on aggregation using spatio-temporal pooling, but on the temporal nature of the kernel itself. However, as previously mentioned, we do use spatio-temporal pooling at the output of our 3D CSNN in order to keep similar feature map sizes for fair comparison.

Another reason for why 2D CSNNs are better suited for shorter video clips is the redundancy of spatial features learned by the 2D CSNN: the spatial kernels learn similar features from similar frames, e.g. the stationary parts of the subject performing the action. The impact of redundancy is decreased with 3D CSNNs as a result of their ability to encode changes over time.

Table 6.5 shows that 3D convolutional SNNs perform significantly better than 2D convolutional SNNs with longer video sequences. However, this significant increase is only obtained in the first two layers; there is a decrease in classification rate for the third layer. This behavior suggests that the length of the video affects the learning in subsequent layers for multi-layer architectures. This is due to the challenge of finding a suitable value for $\hat{t}$, that, which is still an open research problem.

For the rest of this chapter, we use video samples made up of 8 frames for 2D CSNNs, and 20 frames for 3D CSNNs, to compare them in regard to their most convenient settings. The chosen kernel sizes are 5×5 and $5 \times 5 \times 2$ for the 2D and 3D CSNNs, respectively.

	$f_w \times f_h \times f_{td}$	#Kernels L1, L2, L3	L1	L2	L3
3D CSNN	$3 \times 3 \times 2$	$16, 32, 64$	54.63	56.33	56.33
	$3 \times 3 \times 3$	$16, 32, 64$	52.16	57.10	57.56
3D CSNN	$5 \times 5 \times 2$	$16, 32, 64$	56.33	61.42	**63.43**
	$5 \times 5 \times 3$	$16, 32, 64$	55.32	62.73	62.04
3D CSNN	$7 \times 7 \times 2$	$16, 32, 64$	55.40	60.80	56.94
	$7 \times 7 \times 3$	$16, 32, 64$	54.01	56.64	39.81
3D CSNN	$9 \times 9 \times 2$	$16, 32, 64$	56.94	59.14	49.54
	$9 \times 9 \times 3$	$16, 32, 64$	57.41	60.65	39.51
2D CSNN	3×3	$16, 32, 64$	48.38	48.15	47.22
2D CSNN	5×5	$16, 32, 64$	59.72	59.10	**62.35**
2D CSNN	7×7	$16, 32, 64$	57.99	58.80	60.76
2D CSNN	9×9	$16, 32, 64$	54.78	59.10	41.20

Table 6.4: Average classification rates in % on the KTH dataset (8-frame videos) over 3 runs, as a function of different convolutional kernel sizes for 2D and 3D CSNNs.

	$f_w \times f_h \times f_{td}$	#Kernels L1, L2, L3	L1	L2	L3
3D CSNN	$3 \times 3 \times 2$	$16, 32, 64$	65.59	67.59	63.27
	$3 \times 3 \times 3$	$16, 32, 64$	61.46	65.51	65.16
3D CSNN	$5 \times 5 \times 2$	$16, 32, 64$	62.19	**68.21**	63.12
	$5 \times 5 \times 3$	$16, 32, 64$	62.04	67.59	64.35
3D CSNN	$7 \times 7 \times 2$	$16, 32, 64$	62.19	67.59	59.26
	$7 \times 7 \times 3$	$16, 32, 64$	59.49	62.50	49.31
3D CSNN	$9 \times 9 \times 2$	$16, 32, 64$	60.19	62.96	38.66
	$9 \times 9 \times 3$	$16, 32, 64$	62.96	65.74	49.31
2D CSNN	3×3	$16, 32, 64$	45.37	51.39	50.93
2D CSNN	5×5	$16, 32, 64$	54.63	**58.80**	50.93
2D CSNN	7×7	$16, 32, 64$	52.31	56.94	44.91
2D CSNN	9×9	$16, 32, 64$	55.09	54.63	28.70

Table 6.5: Average classification rates in % on the KTH dataset (20-frame videos) over 3 runs, as a function of different convolutional kernel sizes for 2D and 3D CSNNs.

6.3.4 With/without frame subtraction

We compare the performance of 3D and 2D CSNNs to study the behavior of these networks with motion information as input, which offers a better assessment of the comparison with diverse natures of data, in addition to their performance when a part of the spatial information is eliminated. We evaluate the performance of these architectures with motion information by applying frame subtraction to the input. Table 6.7 shows the performance of each architecture using raw frames as input. The results of Table 6.6 acknowledge that 3D CSNNs yield better results than 2D CSNNs in all cases. Table 6.7 shows the performance of each architecture, using frames filtered with FS as input. We see that the motion information that results from FS has improved the classification rates with both the 2D and 3D architectures, so it is interesting to quantitatively compare the feature maps provided by these models with and without frame subtraction, which also provides some insight on the effect of the reduced spatial information on the learned patterns.

In Figure 6.9, comparing the feature maps extracted when the input video is pre-processed with frame subtraction (+FS) shows that frame subtraction has removed the fixed parts of the subject's body from the input frames, and highlights the moving parts of the subject (i.e., the hand in this boxing action). Therefore, the improvement in the classification rate is due to the network learning only from the movement which is significant to identify an action; however, frame subtraction may be less relevant when motion is subtle or when appearance is needed in classifying the action (e.g., when objects are involved).

This amelioration in performance after frame subtraction can also be attributed to the nature of the neural coding. Latency coding transforms the brightest pixels into the latest spikes, which are integrated first by the neuron. This means that bright stationary background pixels, which correspond to salient edges in the background, carry no relevant information but can also largely contribute to prompting the neuron to fire, which perturbs the learning. However, with frame subtraction, all stationary background pixels are eliminated, which improves the performance.

The feature maps obtained by the 2D CSNN and the 3D CSNN in Figure 6.9 seem mostly similar. This is because this specific sample has been classified correctly by both architectures. Figure 6.10 shows the feature maps obtained by 2D and 3D architectures for a walking sample that has been classified incorrectly by the 2D CSNN, but correctly by the 3D CSNN. We see that the 2D CSNN failed to learn features that are significant enough to classify the sample from spatial information. The 3D CSNN was able to learn and focus on motion information, which enabled it to classify the walking action correctly.

Introducing motion as an input to a 3D CSNN has allowed it to learn variations

in motion, such as acceleration, which is beneficial when the network is required to classify actions that are similar in form but different in speed, i.e., walking, running and jogging videos in the KTH dataset. The experiments conducted in this section show that 3D CSNNs trained with STDP can learn spatio-temporal features that are relevant. This also highlights the importance of convolution in the temporal dimension during spatio-temporal feature extraction with spiking models.

Dataset	KTH			Weizmann		
Layers	Conv1	Conv2	Conv3	Conv1	Conv2	Conv3
2D CSNN	57.78	58.61	58.80	48.12	55.38	55.98
3D CSNN	60.80	**67.90**	64.20	52.96	**60.55**	57.92

Table 6.6: Average classification rates in % with 2D and 3D multi-layer SNNs using the KTH and Weizmann datasets, as raw videos, over 3 runs. We use 8-frame videos for 2D architectures and 20-frame videos for 3D architectures.

Dataset	KTH			Weizmann		
Layers	Conv1	Conv2	Conv3	Conv1	Conv2	Conv3
2D	61.57	61.11	61.11	60.43	61.28	62.39
3D	69.75	**72.53**	66.05	61.28	61.54	**64.62**

Table 6.7: Classification rates in % with 2D and 3D multi-layer SNNs using the KTH and Weizmann datasets, as motion information extracted using frame subtraction, over 3 runs. We use 8-frame videos for 2D architectures and 20-frame videos for 3D architectures.

6.4 Conclusion

This chapter introduces STDP-based 3D CSNNs that learn features for action recognition. We give an assessment of 2D and 3D convolutional spiking neural network architectures trained in an unsupervised manner with STDP and challenged with action recognition datasets. Although the performance of these spatio-temporal CSNNs still does not compete with that of non-spiking CNNs, they present an initial spatio-temporal STDP-based CSNN capable of extracting spatio-temporal features without costly pre-processing. The results of this assessment yield several conclusions.

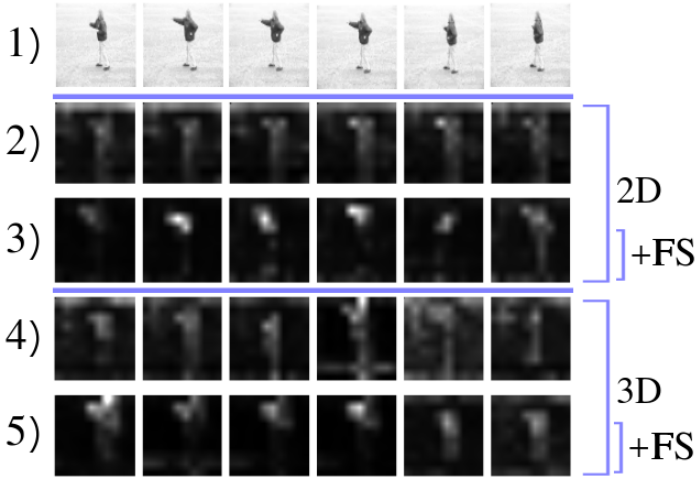


Figure 6.9: The feature maps in the first layer of the 2D and 3D CSNN models from a KTH boxing video, with (+FS) and without frame subtraction. This sample was correctly classified in all cases. 1) The raw video frames, 2) Feature maps of the 2D CSNN, 3) Feature maps of the 2D CSNNs with frame subtraction, 4) Feature maps with 3D CSNNs, and 5) Feature maps of the 3D CSNNs with frame subtraction.

The first one is that unsupervised STDP-based 3D CSNNs can learn visual features of actions, with minimal pre-processing (i.e., DoG filtering). This is thanks to their third dimension dedicated to time, and the capability of their 3D kernels to slide in the spatial and temporal dimensions simultaneously, extracting relevant space-time features for action classification.

The second conclusion is that these models are not only capable of learning space-time features, but they also outperform 2D CSNNs, especially with longer videos. 2D CSNNs require extra steps in order to model motion, like temporal sum pooling or concatenation before the classifier. This allows the final feature maps to be representative of the motion happening amongst the video frames. However, these workarounds can only function for a limited video length. If we increase the number of frames undergoing temporal sum pooling, we obtain excessively complex data. If we use concatenation instead, the final feature map size is too large to be handled effectively by the classifier. This is no longer a problem with 3D CSNNs that are able to extract spatio-temporal feature maps directly.

The third conclusion is that using a multi-layer architectures requires an exhaustive search to find the suitable hyperparameters that permit learning in subsequent layers. The appropriate $\hat{t}$ that is used to control the target timestamp threshold

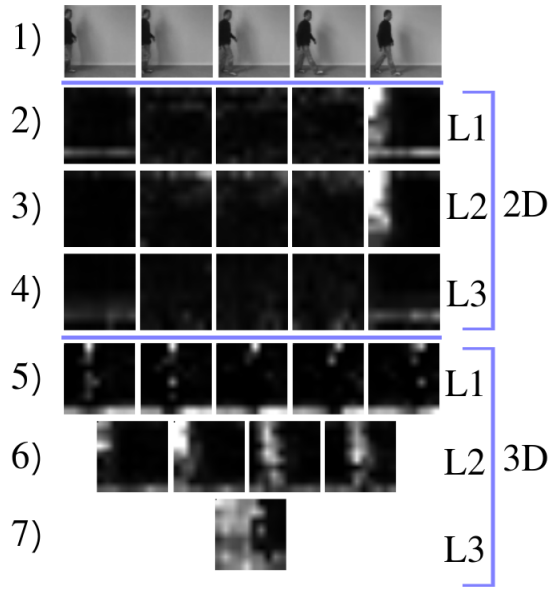


Figure 6.10: The feature maps of the three layers of the 2D and 3D multi-layer SNN models from a KTH walking video. 1) The raw video frames, 2) Layer 1 spatial feature maps, 3) Layer 2 spatial feature maps, 4) Layer 3 spatial feature maps, 5) Layer 1 spatio-temporal feature maps, 6) Layer 2 spatio-temporal feature maps, 7) Layer 3 spatio-temporal feature maps. This sample has been classified incorrectly by the 2D CSNN, while being classified correctly by the 3D CSNN.

adaptation mechanism should be selective enough to control the learned patterns, but also permit enough spike integration for the sake of the learning in subsequent layers. We believe that using permissive to restrictive selectivity by decreasing the target timestamp value throughout the layers is promising.

Finally, our last conclusion concerns the use of frame subtraction to assess the performance of both 2D and 3D CSNNs architectures. We conclude that frame subtraction improves the performance of both architectures because it allows the elimination of stationary background pixels that raise irrelevant features. Without FS, these pixels are transformed into very early spikes using latency coding, and they cause the neuron to fire erroneously. It is fair to say that even in this setting, 3D CSNNs still outperform 2D CSNNs. This shows once again their superiority for HAR video analysis.

The 3D convolutional spiking neural networks tested in this work serve as a good building block in improving human action recognition with unsupervised STDP-based spiking models. The performance of unsupervised STDP-based SNNs is still far behind that of state-of-the-art CNNs. However, it should be noted that our features are learned without supervision, which presents inherent challenges. Unsupervised learning, unlike supervised learning, lacks access to ground truth labels, making it more difficult to train and evaluate models effectively. This complexity in the learning process, in part, may explain the performance gap observed between our SNNs and CNNs trained with supervised methods. Yet, further research is still needed to improve the performance of CSNNs.

Chapter 7

Spiking Separable Convolutions

A 3D convolutional spiking neural network (CSNN) has the advantage of being a fully spiking solution for learning motion patterns. Therefore, as explained in Chapter 6, it can be used for video analysis in the spiking domain, without needing additional non-spiking processes to extract the temporal components of videos. However, similarly to traditional methods, spiking 3D convolutions increase the number of parameters with respect to spiking 2D convolutions. This can make it harder to construct this model with neuromorphic hardware, since the number of required connections are increased. Some neuromorphic hardware allows the implementation of arbitrary multigraph networks, but the resource constraints of the core should be taken into consideration [15]. Moreover, this increased number of parameters also increases the computational costs with regard to 2D CSNNs. Therefore, it is important to find methods that can reduce the number of parameters while conserving the same performance as spiking 3D CSNNs. Separated convolutions, as explained in Chapter 2 (page 33), are a solution to the increased number of parameters that has been used in the CNN-related literature [90] [91], but has not yet been explored in the spiking domain.

In this chapter, we present Spiking Separated Spatial and Temporal Convolutions (S3TCs), where we reduce the number of parameters in a spiking spatio-temporal 3D convolution by factorizing it into two separate smaller spatial and temporal convolutions. We train these CSNNs in an unsupervised manner using the STDP learning rule. S3TCs are expected to be more efficient and hardware friendlier spiking solutions than 3D CSNNs for video analysis. To the best of our knowledge, our work is the first to address the subject of separated convolutions with spiking neural networks. Our hypothesis is that simpler 2D and 1D kernels in S3TCs can enhance STDP-based learning by capturing more patterns, thereby prompting the neurons

to fire more spikes. Since STDP updates are triggered by neuron firing events, this increased activity can lead to an increased frequency of updates.

This chapter is a building block towards improving the performance of spatio-temporal spiking models, while promoting hardware friendliness and energy efficiency. The main contributions of this chapter are summarized as follows:

- we introduce Spiking Separated Spatial and Temporal Convolutions (S3TCs);
- we evaluate the performance of S3TC models with different kernel sizes on the KTH [54], Weizmann [55], and IXMAS [56] datasets;
- we compare the performance of S3TCs to that of spiking 3D convolution from Chapter 6, and we conclude that S3TCs can achieve better performance;
- we show that using smaller kernel sizes along with STDP, to a certain extent, prompts the neurons to fire more spikes, thus increasing the network activity and improving the learning.

In the following sections, we begin by providing the number of trainable parameters in a 3D CSNN, which offers insights into the computational costs associated with these networks. Then, we introduce S3TCs, highlighting their smaller number of training parameters compared to 3D CSNNs. In subsequent sections, we discuss the impact of kernel size on both network architectures. We then proceed to examine and compare the spiking activity between 3D CSNNs and S3TCs, along with its influence on network performance.

7.1 Parameter study of 3D CSNN

The 3D CSNN introduced in Chapter 6 has f_k complex 3D trainable kernels of sizes $f_w \times f_h \times f_{\text{td}}$, where f_w and f_h represent the width and height of the kernel respectively, and f_{td} is the temporal size of the kernel. These 3D kernels permit spatio-temporal feature extraction from videos, but have a larger number of parameters than their 2D counterpart. These numbers of trainable parameters P_{2D} for 2D and P_{3D} for 3D kernels are introduced in Equation 7.1 and 7.2, respectively:

$$|P_{2D}| = f_k \times n_{\text{c}} \times f_w \times f_h \tag{7.1}$$

$$|P_{3D}| = f_k \times n_{\text{c}} \times f_w \times f_h \times f_{\text{td}} \tag{7.2}$$

where f_k is the number of kernels, n_{c} is the number of input channels, f_w, f_h, and f_{td} represent the width and height and temporal dimension of the kernel, respectively.

7.2 Spiking separated spatial & temporal convolutions

With separable convolutions, the kernel connectivity of the spiking 3D convolution layer introduced in Chapter 6 can be broken down into two parts: space-wise and time-wise convolutions, as shown in Figure 7.1.

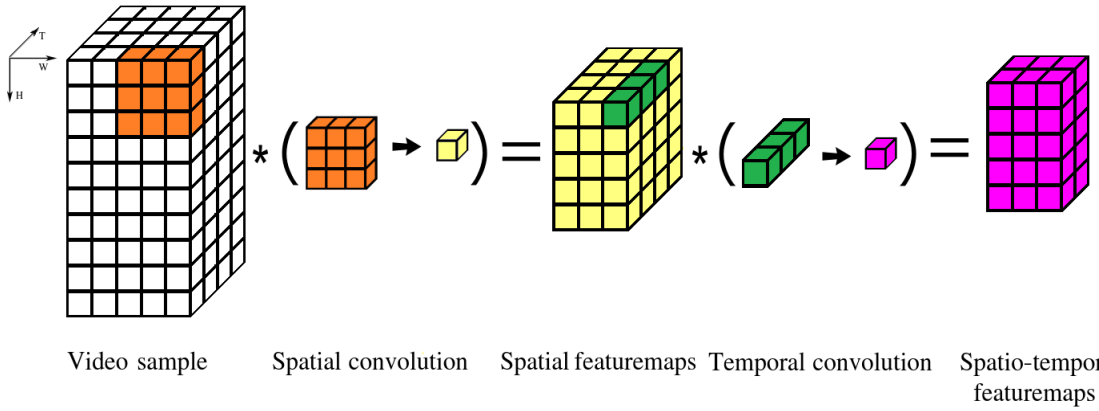


Figure 7.1: Separable spatial and temporal convolutions.

In the first phase, a 2D kernel crosses over the spatial dimension of the input, one frame at a time, so this kernel has a dimension of $f_w \times f_h \times 1$. This results in spatial feature maps as an output of the first phase. In the second phase, with the time-wise convolution, we compute a linear combination of the previous outputs by undergoing a $1 \times 1 \times f_{\text{td}}$ convolution in the temporal dimension to extract meaningful temporal information from the spatial feature maps. The same training algorithm introduced in Chapter 4 (page 67) still applies for training the 2D layer of the S3TC block. The network is trained layer-wise: after the 2D layer is trained, its weights remain constant, and the output of this layer serves as the input spikes that are used to train the 1D block, following the same algorithm. The threshold adaptation and STDP update mechanisms of the 1D block are the same as their 2D counterparts, since they are independent of input dimension.

Spiking separated convolution can be formalized as Equation 7.3 for the space-wise convolution, and as Equation 7.4 for the time-wise convolution:

$$V_{m_{x,y,k}}(t) = \sum_{i \in \mathcal{E}_s} W_{n(x_i),j(y_i),k_i,k} \times f_s(t - t_i) \tag{7.3}$$

$$V_{m_{z,k}}(t) = \sum_{i \in \mathcal{E}_t} W_{m(z_i),k_i,k} \times f_s(t - t_i) \tag{7.4}$$

$$f_s(t - t_i) = \begin{cases} 1 & \text{if } t \geq t_i \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

where f_s is the kernel of spikes, V_m is the potential of the neuron membrane at time t, and x, y, z, and k are the coordinates of the spike in the width, height, time, and channel dimensions, respectively. W is the trainable synaptic weight matrix, $n()$, $j()$, and $m()$ are functions that are used to map the location of the input neuron to the corresponding location in the weight matrix, and $\mathcal{E}_s$ and $\mathcal{E}_t$ are the sets of input connections in the spatial and temporal neighborhoods, respectively.

The number of parameters P_{S3TC} required for training S3TCs is:

$$|P_{S3TC}| = f_k \times n_\text{c} \times (f_w \times f_h + f_\text{td}) \tag{7.6}$$

This number of parameters is lower than that of a spiking 3D convolution in Equation 7.2, therefore this reduced parameter count would also translate to lower energy consumption, which aligns well with the need for energy-efficient solutions for implementation on devices with limited energy.

7.3 Evaluation

7.3.1 Datasets and implementation details

Experiments are performed on the KTH [54], Weizmann [55] and IXMAS [56] datasets, previously introduced in Section 2.1 (page 17). The protocols used are mentioned in Section 4.9 (page 70).

We use the same on-center/off-center filter [196] detailed in Chapter 4 (page 63).

The value of the target timestamp $\hat{t}$ discussed in Section 6.3.2 (page 102) is specific for each dataset. Similarly to the preceding chapters, we use a value of $\hat{t} = 0.65$ for the KTH dataset and a value of $\hat{t} = 0.75$ for the Weizmann dataset, for both convolutional layers. For the IXMAS dataset, we use $\hat{t} = 0.65$, which was determined using an exhaustive search. A spatio-temporal pooling is applied at the end of each network, in order to limit the size of the output feature maps to $20 \times 20 \times 2$. The neuron thresholds are randomly initialized with a normal distribution $V_{th} \sim \mathcal{N}(\mu, \sigma^2)$ with a mean of $\mu = 8$ and variance of $\sigma^2 = 0.1$ for all experiments, except those with a kernel size of 3, where we decrease the mean to 7, as shown in Table 7.2. This is needed because very small kernel sizes mean fewer synapses, which results in no initial spiking activity when threshold values are too high.

STDP
$\eta_w = 0.1$, $\tau_{\text{STDP}} = 0.1$, $W \sim U(0, 1)$
Threshold Adaptation
$\hat{t}_{\text{KTH}} = \hat{t}_{\text{IXMAS}} = (0.65, 0.65)$, $\hat{t}_{\text{Weizmann}} = (0.75, 0.75)$ $\text{th}_{\text{min}} = 1.0$, $\eta_{\text{th}} = 1.0$
Difference-of-Gaussian
$\sigma_1 = 1.0$, $\sigma_2 = 4.0$, s = 7.0

Table 7.1: The hyperparameter values used in the experiments.

kernel size	μ	σ^2
3	7	0.1
5	8	0.1
7	8	0.1
9	8	0.1
10	8	0.1

Table 7.2: The values of the mean μ and variance σ^2 selected for the Gaussian distribution $V_{th} \sim \mathcal{N}(\mu, \sigma^2)$ which randomly initializes the threshold for each kernel size.

We use 10 frames per video, and we also skip three frames between each two selected frames in order to make sure to capture a full cycle of the performed action. We scale down the frame sizes to half of their original sizes for processing speed reasons. We measure the classification accuracy (in %) on the test set for all experiments. Each experiment was run three times, and we report the average accuracy over the three runs. Each SNN is trained independently, and an SVM with a linear kernel is placed at the end of each neural network.

7.3.2 3D CSNNs vs. S3TCs

We implement separated convolutions for five different kernel sizes. For the sake of limiting the possible kernel size combinations, we use the same size f for both the spatial and temporal kernels. Each 3D convolution has a kernel size of $f \times f \times f$, while the kernel sizes of its corresponding separated convolutions are $f \times f \times 1$ for the spatial convolution and $1 \times 1 \times f$ for the temporal one. The most commonly used kernel size is $f = 3$ in the literature [90, 91, 202]. However, larger kernel sizes like $f = 5$ and $f = 7$ have shown to give better results in [203]. We extend this idea by also using kernel sizes of $f = 9$ and $f = 10$, with the aim of exploring whether further increases in kernel size can provide additional performance gains, in the context of spiking networks. Table 7.3 shows the results of experiments with different kernel sizes.

These results show that separated convolutions can achieve competitive performance with 3D convolution, while having less parameters. S3TC shows a consistent amelioration in performance against its 3D CSNN counterpart with the KTH dataset, with the best performance being 70.52% recorded in Table 7.3 (C). This result is an amelioration of around 6 *p.p.* from the best performance of the 3D CSNN. The best performance with the Weizmann dataset is also recorded in Table 7.3 (C), where an amelioration of around 7.6 *p.p.* is recorded. However, with a larger kernel size of 10, separated convolutions start to fall slightly behind the performance of 3D CSNNs (around 1 *p.p.*). On the other hand, with the IXMAS dataset, smaller kernel sizes make the performance of our S3TC fall slightly behind the performance of 3D CSNNs (around 1 *p.p.*). With larger kernel sizes, S3TCs outperform 3D CSNNs, with a difference that can go up till around 12*p.p.* This shows the importance of choosing an appropriate kernel size for these datasets.

(A) Kernel size = 3		
Dataset	3D Convolutions	Separated Convolutions
KTH	59.41	**60.65**
IXMAS	**53.81**	52.26
Weizmann	56.58	**59.29**
(B) Kernel size = 5		
Dataset	3D Convolutions	Separated Convolutions
KTH	61.88	**69.29**
IXMAS	**51.56**	51.44
Weizmann	57.61	**66.24**
(C) Kernel size = 7		
Dataset	3D Convolutions	Separated Convolutions
KTH	64.20	**70.52**
IXMAS	40.74	**48.87**
Weizmann	57.09	**65.78**
(D) Kernel size = 9		
Dataset	3D Convolutions	Separated Convolutions
KTH	62.81	**65.43**
IXMAS	34.01	**46.46**
Weizmann	56.50	**64.62**
(E) Kernel size = 10		
Dataset	3D Convolutions	Separated Convolutions
KTH	60.65	**61.11**
IXMAS	27.94	**38.50**
Weizmann	**58.09**	57.12

Table 7.3: Classification rates in % (average ± standard deviation) for the KTH, IXMAS, and Weizmann datasets (10 frames per video) over 3 runs with 3D convolution and separated convolutions.

7.3.3 Kernel size impact

As explained in the previous section, different datasets have different ideal kernel sizes for the separated convolutions, as shown in Table 7.3 (i.e., $f = 7$ for the KTH dataset, $f = 3$ for IXMAS, and $f = 5$ for Weizmann). This is because these datasets have different scales and resolutions, which would impact the optimal kernel size. The results in Table 7.3 also show that kernel size has more impact on the results with separated convolution. For instance, with the KTH dataset, changing the kernel size from $f = 3$ to $f = 7$ using 3D convolutions has an impact of around 5 $p.p.$, while with separated convolutions, the impact is around $10p.p.$ In summary, tailoring kernel sizes to the unique characteristics of each dataset showcases the nuanced nature of separated spatial and temporal convolutional operations.

In the context of action recognition, the spatial kernels are responsible for extracting a series of positions taken by a subject in successive frames. However, these kernels cannot learn the temporal patterns that model the relationship between these extracted features. The larger the spatial kernel, up to some extent, the easier it is to capture and define the patterns of the subject in the video frames. The 1D temporal kernels model the time dimension, thus combining the output feature maps of the 2D kernels into more discriminant combinations that include motion. Larger kernels provide a better extraction of these moving patterns with datasets that exhibit significant variations or large movements relative to the frame size, like the KTH dataset. Smaller kernels are needed for datasets that exhibit smaller variations, like the Weizmann dataset. Hence, the performance of spiking separated convolutions, much like 3D convolutions, relies significantly on selecting suitable hyperparameter values. However, S3TC is even more responsive to these values than 3D convolutions.

7.3.4 Spiking activity

Separated convolutions can outperform regular 3D convolutions for most datasets. This behavior is similar to the one observed with non-spiking networks in [92], where they specify that 2D and 1D kernels are easier to optimize with supervised learning than 3D kernels. This kernel optimization with backpropagation is done by calculating the gradients of the loss, as previously explained in Chapter 3 (page 51), with respect to each kernel in a CNN, and an optimization algorithm like stochastic gradient descent is used to update the kernel weights, where the weights are gradually updated over multiple epochs to improve the ability of the network to detect relevant features in the data. However, this same kernel optimization process does not happen with STDP, where we use unsupervised learning, and not gradient descent. Therefore, simpler kernel optimization cannot be the direct reason for the improved

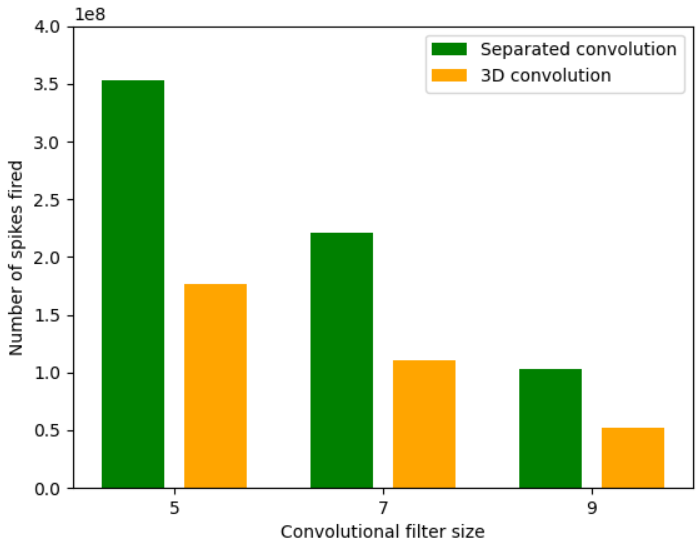


Figure 7.2: The number of spikes fired by separable spatial and temporal convolutions compared to 3D convolutions with kernel sizes of 5, 7 and 9 using the KTH dataset.

performance we witness with spiking models. STDP is a local learning rule that is independent of dimension, because it acts on the synapse connecting each two neurons independently of the others.

In the case of our unsupervised spiking model, we observe that more spikes are fired when using separated convolutions than its 3D counterpart, as shown in Figure 7.2. This is due to the simplicity of the 2D spatial and 1D temporal kernels, which enables them to respond more often to the input than the more complex 3D kernels. For kernels with fewer parameters, the number of possible combinations of input spike patterns that could trigger an output spike is significantly lower than that of a larger kernel. This results in firing a larger number of output spikes with separated convolutions and increases the activity of the network, which contributes to higher accuracy.

Firing a larger number of output spikes can help counteract the spike vanishing problem discussed in Chapter 1 (page 7). This problem is a significant challenge encountered with SNNs, and has kept deep STDP-based SNNs an open research topic till this day [167]. It results from spiking neurons needing to integrate many spikes to reach their threshold and fire one output spike, which reduces the spiking activity in subsequent layers, and hinders learning. Therefore, the fact that separated convo-

lutions produce a greater number of output spikes makes them more advantageous than 3D CSNNs for training multi-layer spiking models.

7.4 Conclusion

Spiking neural networks can offer an energy-efficient solution on neuromorphic hardware. However, the usage of 3D convolutions, which are suitable for video analysis, increases the number of parameters, making training more challenging and potentially leading to more complex hardware requirements. To mitigate this issue, we opt to reduce the number of parameters in the network by replacing 3D convolutions with separated convolutions. In this chapter, we factorize a single 3D spiking convolution into two separate spatial and temporal spiking convolutions. This separation decreases the number of needed parameters, and can improve the performance when using sufficiently large kernels. The difference in performance between 3D convolutions and separable convolutions is highly dependent on choosing the appropriate kernel size.

Our first conclusion is that the optimum kernel sizes vary from one dataset to another depending on their motion variations. If the ratio of significant motion to background is small, larger kernels may not be ideal, like with the Weizmann dataset. Otherwise, smaller kernels could provide more benefit.

A second conclusion is that spiking separated convolutions can outperform 3D convolutions due to their smaller number of parameters, which leads to capturing more patterns and thus firing more spikes. There is a proportional relationship between the number of input spikes in a layer and the quality of the learned kernels. This is because the number of spikes needs to be sufficient to allow the kernels to converge and learn meaningful patterns effectively. The increased spiking activity at the output of S3TC layers compared to their 3D counterparts makes these networks more attractive candidates than 3D CSNNs for multi-layer architectures, since it diminishes the severity of the spike vanishing problem by promoting more activity at the outputs of its layers.

Although the performance of these S3TCs is not competitive with that of non-spiking methods, they present an initial spatio-temporal model based on STDP that can extract features without supervision. In contrast, the non-spiking methods discussed in the literature necessitate significant amounts of annotated data for effective training, unlike S3TCs, reducing the reliance on labeled data.

Chapter 8

Spiking Two-stream Networks

Two-stream networks are spatio-temporal models specifically designed for effective video analysis [86], as discussed in Chapter 2 (page 36). When compared to other deep methods, they exhibit state-of-the-art accuracy on certain action recognition datasets [2]. However, the strong performance of these models, like most deep ANNs, comes with a considerable computational cost [8], and requires large amounts of labeled data for back-propagation. These issues of traditional two-stream networks can be mitigated by using STDP-based CSNNs as spiking streams, which can learn visual features in an unsupervised manner, mitigating the reliance of the recognition pipeline on labeled data.

Spiking two-stream networks harness the advantages of both SNNs and two-stream architectures. These methods have been introduced previously in [185], where the authors introduce a deep two-stream SNN based on a spiking ResNet50 architecture and a recurrent spiking neural network (RSNN) fusion module. They employ ANN-to-SNN conversion with their unique hybrid method. However, ANN-to-SNN conversion still necessitates training with a non-spiking ANN, diminishing the energy efficiency gains.

The use of two-stream methods in conjunction with unsupervised STDP-based learning in the spiking domain has not yet been explored.

In this chapter, we explore the effectiveness of unsupervised STDP-based two-stream CSNNs in learning spatio-temporal features for HAR with reduced computational and labeling costs, compared to non-spiking two-stream methods. To do so, we adopt a mixed classification pipeline, in which unsupervised 2D or 3D CSNNs are used to learn visual features to be fed to a supervised classifier. The main contributions of this chapter are summarized as follows:

- we present a two-stream spiking convolutional model, and we investigate mo-

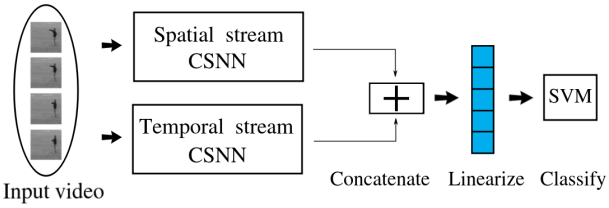


Figure 8.1: The general two-stream architecture.

tion modeling with five different types of temporal streams in Section 8.1;

- we evaluate and compare the performance of two-stream spiking convolutional models with our different temporal streams on the KTH [54], Weizmann [55], IXMAS [56], and UCF Sports [57] action recognition datasets in Section 7.3.

8.1 Two-stream architectures

Two-stream networks have two different streams used for extracting different types of features from the same video sample, as shown in Figure 8.1. These networks are effective in scenarios where both the spatial appearance and temporal dynamics of the visual data are beneficial, like HAR.

The network takes a video clip as input. The appearance information is extracted by the spatial stream, and the motion information is extracted by the temporal stream. Then the information extracted from both streams is concatenated and sent into the classifier.

By leveraging the complementary strengths of spatial and temporal processing, two-stream networks enhance the ability to analyze complex visual patterns in HAR videos.

8.1.1 The spatial stream

The spatial stream used in this chapter is an unsupervised STDP-based 2D CSNN, which is responsible for extracting appearance information from the video. It extends the recognition pipeline from Figure 4.1 in Chapter 4 (page 63)(classification step excluded) to process videos. Our approach is displayed in Figure 8.2. It consists in processing multiple spatial frames separately, producing a set of spiking feature maps for each frame in the video. The feature maps are subsequently combined through sum pooling. This process generates a single set of feature maps that effectively

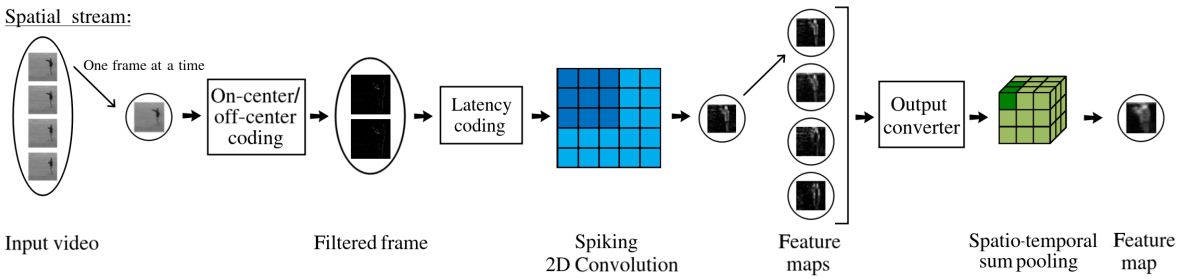


Figure 8.2: Architecture of the spatial stream. This architecture replaces the box labeled "Spatial stream CSNN" in Figure 8.1
.

represents the prominent spatial features within the whole video clip. The resulting feature maps are then ready for fusion with the output of the temporal stream.

This approach differs from the method adopted in conventional two-stream methods. Typically, these methods only use a single frame from the video, selected either randomly from a subset of frames, or through an algorithm that identifies the most relevant frame for accurately depicting the silhouette of the subject for a specific class, which requires some supervision [204]. With some action recognition datasets, selecting a frame randomly can lead to similar frames between actions. For instance, in a video of people clapping, the selected frame should capture the moment when the subject's hands touch, as choosing any other frame might make it seem like they are waving. Supervised frame selection methods, on the other side, are not suited to our goals, since we aim at non-supervised feature learning. As a result, we opt to use multiple frames and extract multiple spatial feature maps using our spatial stream.

8.1.2 The temporal stream

The temporal stream tackles the temporal dynamics of videos, focusing on identifying and extracting motion-related features. We explore five distinct streams to capture temporal data from videos: four temporal stream configurations that use a 2D pipeline, as shown in Figure 8.3, and one configuration that uses a 3D pipeline, as shown in Figure 8.4. All of these configurations are inspired by previous chapters.

- Early Fusion: Taking advantage of the early fusion method described in Section 5.1.1 (page 76), which implicitly provides elementary motion information, we evaluate the capability of the SNN to acquire spatio-temporal patterns from data processed using this technique. This early fusion method allows the CSNN

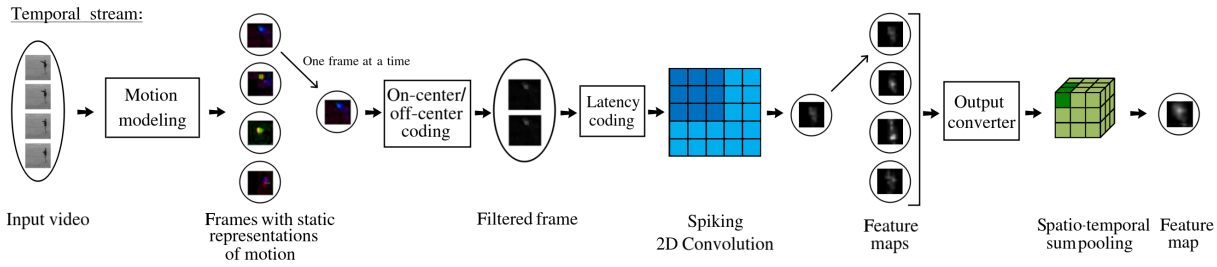


Figure 8.3: Architecture of the temporal streams based on 2D convolutions. This architecture replaces the box labeled "Temporal stream CSNN" in Figure 8.1

to process the entire video clip as one sample, which maintains the temporal information between video frames without requiring the use of complex handcrafted motion modeling methods such as optical flow. We aim to determine whether the feature maps extracted by the temporal stream with this elementary motion modeling method complement those of the spatial stream.

- Optical flow: Optical flow is the most commonly used form of temporal information in non-spiking two-stream methods [4, 205, 206]. We apply a motion modeling method, based on Farneback's dense optical flow [63], to the temporal stream in our spiking models and assess the relevance of motion patterns obtained through this conventional technique in the context of spiking two-stream networks. Similarly to what is explained in Section 5.3.1 (page 81), we compute the magnitude and orientation of optical flow vectors using Equations 5.9 and 5.10 (page 81). As explained in Section 5.3.1, since orientation data is periodic and difficult to apply latency coding, the optical flow is displayed in the HSV color space, then converted into the RGB color space. Finally, latency coding is applied to the resulting frame.

- Frame subtraction: There are simpler ways for motion modeling for the temporal stream than optical flow, which require fewer computations. Therefore, we investigate the performance of the spiking two-stream CSNN when using frame subtraction for motion extraction in the temporal stream. This method is described in Section 5.2, Equation 5.4 (page 78). We find this method suitable for the temporal stream as it offers a straightforward motion extraction technique, excluding static shape-related details like texture. The question arises whether the spatial stream can effectively compensate for this omission.

- Motion Grid: We also assess the performance of our spiking two-stream network

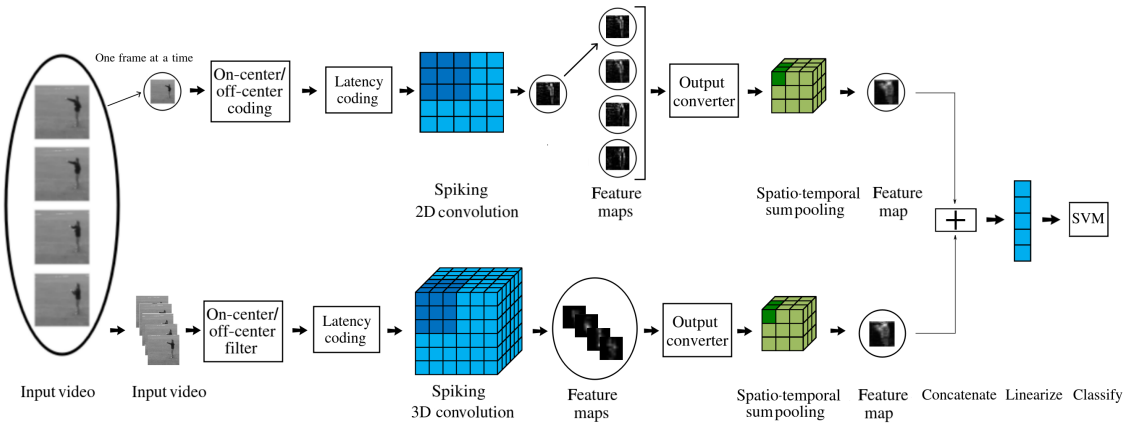


Figure 8.4: Architecture of the spiking two-stream network with temporal stream based on 3D convolutions.

using shot-based methods explained in Section 5.3.2 (page 83) for the temporal stream. We use the motion grid because of its demonstrated effectiveness for HAR with STDP-based SNNs, as demonstrated in Chapter 5.

Motion modeling with the motion grid method requires collecting the optical flow of each two consecutive frames, and separate the displacement information into four different directions. Each direction is collected in a separate sub-frame. Then these sub-frames are joined together to construct the grid, as described in Section 5.3.2 (page 83).

- Spatio-temporal data: Lastly, we address whether including redundant spatial and temporal data is beneficial. With two-stream methods, the temporal stream was originally intended to only extract temporal features, while the spatial stream extracts the appearance information. Yet, authors in [111] claim that it is more effective to fuse a spatial stream with a spatio-temporal stream that uses 3D CNNs. In this work, we replace the temporal stream by the 3D convolutional spiking neural network introduced in Chapter 6 to extract spatio-temporal information. This aims at checking if the same amelioration in performance is attainable in the spiking domain as that attained in the non-spiking domain with traditional 3D CNNs. Figure 8.4 shows the architecture of the streams used in this case.

Learning
$f_k = 64$
Input settings
$l_{td} = 10$ for all experiments, except MG: $l_{td} = 48$
STDP
$\eta_w = 0.1$, $\tau_{\text{STDP}} = 0.1$, $W \sim U(0, 1)$
Threshold Adaptation
$\text{th}_{\text{min}} = 1.0$, $\eta_{\text{th}} = 1.0$, $V_{\text{th}}(0) \sim \mathcal{N}(8, 1)$
Difference-of-Gaussian
$s = 7.0$, $\sigma_1 = 1.0$, $\sigma_2 = 4.0$

Table 8.1: The hyperparameter values used in the experiments.

8.2 Evaluation

This section provides the details of our experiments. First, we present the datasets, along with the implementation details and the hyperparameters of our network. Then, we show the results of implementing and testing our spiking two-stream neural network, where we evaluate the individual performance of the spatial and temporal streams, in addition to the performance of the complete architecture.

8.2.1 Datasets and implementation details

We use the KTH, Weizmann, IXMAS and UCF-Sports action recognition datasets which were previously introduced in 2.1 (page 17). Their evaluation protocols were described in Section 4.9 (page 70).

The hyperparameters used in this chapter are presented in Table 8.1. We use the same 2D on-center/off-center as in previous chapters, with the same values of σ_1 and σ_2.

The convolutional layer of each stream has $f_k = 64$ kernels for all settings. The two-stream CSNN uses the spatial stream of Figure 8.2 for all experiments with a convolutional kernel size of 5×5. The 2D temporal streams also have a convolutional kernel size of 5×5. The 3D temporal stream has a convolutional kernel size of $5 \times 5 \times 2$, where the kernel size in the temporal dimension is small in order to detect small motion variations, which proved beneficial compared to a temporal size of 3 (see Table 6.4, page 107). The value of the target timestamp $\hat{t}$ that is discussed in

Section 4.5 (page 66) is specific to each dataset. We use a value of $\hat{t} = 0.65$ for the KTH, IXMAS and UCF Sports datasets, and a value of $\hat{t} = 0.75$ for the Weizmann dataset, which has smaller motion variations because the camera is further away from the subjects than in other datasets. The size of the output feature maps is set to $20 \times 20 \times 2$, except for the streams that use a single-frame representation of motion as a pre-processing, i.e. early fusion and motion grid, in which case the feature map size is set to $20 \times 20 \times 1$.

We use 10 frames per video, skipping three frames between each two selected frames. This number of frames per sample is fixed in all experiments except the motion grid, which uses 48 frames to construct one grid. As mentioned in Section 4.9 (page 70), we also scale down the frame sizes to half of their original sizes for processing speed reasons.

After the spiking feature maps are obtained at the output, they are converted back into non-spiking feature maps using Equation 4.7 (page 69), as in previous chapters. Spatio-temporal sum-pooling is used at the output of each CSNN.

We measure the classification accuracy (in %) on the test set for all experiments. Each experiment is run three times, and we report the average accuracy over the three runs. Each CSNN stream is trained independently and an SVM is placed at the end of each stream, in order to have an accurate measure of performance for the feature maps before and after fusion.

8.2.2 Performance with varied temporal stream configurations

In this section, we study the effects of using different temporal configurations within the two-stream architectures. Tables 8.2, 8.3, 8.4, 8.5, and 8.6 show the results of experiments where the spatial stream processes raw video frames, and the temporal streams use early fusion, optical flow, frame subtraction, motion grid, or 3D convolution, respectively.

The results in Table 8.2 show that the performance of the spatial stream with raw videos is better than that of the temporal stream, which uses early fusion for motion modeling. This is because we implement temporal pooling at the end of the spatial stream, in order to join the output feature maps together. This is equivalent to a sort of late fusion, and, as shown in [86] and in our results in Chapter 5, late fusion has a better performance than early fusion with CNNs and with STDP-based CSNNs. Nevertheless, fusing both streams gives superior results to the individual performance of those streams with all datasets, except for the Weizmann dataset. With this dataset, fusing the features together did not give any added value, and thus

Dataset	Raw video (2D conv)	EF (2D conv)	Fused
KTH	54.01	52.16	**55.71**
UCF Sports	35.33	26.67	**38.00**
IXMAS	43.46	38.44	**48.54**
Weizmann	**52.31**	51.91	51.91

Table 8.2: Classification rates in % on the KTH, UCF Sports, IXMAS, and Weizmann datasets (10 frames per video) over 3 runs. The spatial stream uses raw frames, and the temporal stream uses early fusion (EF).

Dataset	Raw video (2D conv)	OF (2D conv)	Fused
KTH	54.01	61.73	**64.51**
UCF Sports	35.33	38.00	**40.00**
IXMAS	43.46	61.41	**62.52**
Weizmann	52.31	65.84	**66.21**

Table 8.3: Classification rates in % on the KTH, UCF Sports, IXMAS, and Weizmann datasets (10 frames per video) over 3 runs. The spatial stream uses raw frames, and the temporal stream uses optical flow (OF).

did not result in a better classification rate. This is a consequence of noise patterns that are learned by the spatial stream, as we will discuss later in Section 8.3.

The results in Table 8.3 show that the difference in performance between the temporal stream that extracts features from optical flow data and the raw spatial stream can be large (more than 10 p.p.). Yet, this does not affect the fact that the temporal and spatial recognition streams are complementary, because their fusion systematically improves the results over having only one stream or the other. Therefore, with spiking models, the streams do not have to have similar performance in order for the fusion to be beneficial.

The same applies to the results displayed in Table 8.4, where some experiments show a significant difference in performance between the streams. The features extracted by these two streams are diverse and complementary: one of them extracts appearance information, while the other one extracts features from edges of motion, because a DoG filter is applied after frame subtraction. These results further show that these networks leverage the strengths of each stream, resulting in improved

Dataset	Raw video (2D conv)	FS (2D conv)	Fused
KTH	54.01	64.97	**67.75**
UCF Sports	35.33	38.33	**38.67**
IXMAS	43.46	60.93	**61.69**
Weizmann	52.31	**71.23**	63.45

Table 8.4: Classification rates in % on the KTH, UCF Sports, IXMAS, and Weizmann datasets (10 frames per video) over 3 runs. The spatial stream uses raw frames, and the temporal stream uses frame subtraction (FS).

Dataset	Raw video (2D conv)	MG (2D conv)	Fused
KTH	54.01	66.82	**67.28**
UCF Sports	35.33	46.00	**50.67**
IXMAS	43.46	60.50	**63.83**
Weizmann	52.31	**70.77**	68.52

Table 8.5: Classification rates in % on the KTH, UCF Sports, IXMAS, and Weizmann datasets (10 frames per video) over 3 runs. The spatial stream uses raw frames, and the temporal stream uses the motion grid (MG) motion modeling.

performance when fusing the spatial and temporal feature maps.

Table 8.5 shows the result of fusing the spatial stream feature maps with those extracted by using a single frame representation of motion. The same behavior as above is observed: the temporal stream that uses the motion grid representation significantly outperforms the spatial one, and yet the fusion performance is still ameliorated.

Table 8.6 shows the results of combining the features extracted by a 2D CSNN with those of a 3D CSNN. These results show that combining spatial features with spatio-temporal features does not systematically show notable improvement with all datasets. This is because the 3D CSNN is already able to extract the appearance information extracted by the spatial stream, in addition to temporal features. Therefore, adding the spatial features coming from the spatial stream to the spatio-temporal one results in information redundancy that does not always help ameliorate the results. However, the results do ameliorate after fusion with the IXMAS dataset. This is because convolutional neural networks are less effective when extracting spa-

Dataset	Raw video (2D conv)	Raw video (3D conv)	Fused
KTH	54.01	**61.27**	54.48
UCF Sports	35.33	**52.00**	48.67
IXMAS	43.46	55.33	**55.54**
Weizmann	52.31	**55.41**	55.07

Table 8.6: Classification rates in % on the KTH, UCF Sports, IXMAS, and Weizmann datasets (10 frames per video) over 3 runs. The spatial stream uses raw frames, and the temporal stream uses 3D Convolution.

Dataset	FS (2D conv)	Raw video (3D conv)	Fused
KTH	**64.97**	61.27	64.35
UCF Sports	38.33	**52.00**	43.33
IXMAS	60.93	55.33	**62.39**
Weizmann	**71.23**	55.41	60.31

Table 8.7: Classification rates in % on the KTH, UCF Sports, IXMAS and Weizmann datasets (10 frames per video) over 3 runs. The spatial stream uses frame subtraction with 2D convolution, and the temporal stream uses raw frames with 3D convolution.

tial information from datasets with different viewpoints, like the IXMAS dataset, because very different image observations can be obtained from observing the same action from different viewpoints [39]. What was observed in [39] still applies for STDP-based learning, where having several viewpoints increases the number of patterns to learn. This decreases the feature redundancy, and thus makes the fusion beneficial. In the same spirit, we check if the same effect applies to adding additional temporal information for some temporal redundancy. Table 8.7 shows the results of the fusion between a stream that uses frame subtraction and another that uses 3D convolution. These results show that redundant temporal information does not systematically ameliorate the fusion performance. Again, the only recorded amelioration is with the IXMAS dataset due to an increased number of patterns to learn from several viewpoints. These results support the initial claim that adding redundant information regardless of its nature (spatial or temporal) to spatio-temporal information does not always result in better performance.

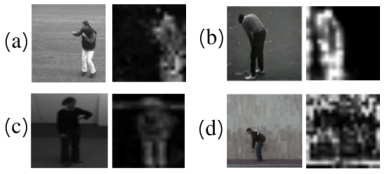


Figure 8.5: Raw videos and spatial stream feature maps as samples of the (a) KTH, (b) UCF Sports, (c) IXMAS, and (d) Weizmann datasets.

8.3 Discussion

In some experiments, the variance of the UCF Sports dataset remains zero regardless of the different seeds used, owing to the limited number of samples. Among various seeds, not always the same samples were recognized, but overall, the same number of correctly classified samples was observed.

Results show that the individual stream performances of the spiking networks do not have to be similar so that their fusion achieves better performance. Moreover, fusing spatial and temporal information is usually beneficial, even when the temporal stream records less performance than the spatial one. These results also show that spiking models can effectively learn spatial as well as temporal features with STDP, and that the fusion performance is related to the nature of the fused features. This is visible when fusing spatial information obtained from a raw video with motion information obtained from the same video pre-processed with optical flow extraction. This delivers a richness in the resulting fused spatio-temporal features, and thus ameliorates the results. On the other hand, fusing spatial features with spatio-temporal features generally gives no improvement in performance, as shown in Tables 8.6 and 8.7, except when there are notable variations in the patterns of the same action, such as in the case of videos shot from different viewpoints. This makes it more difficult to learn representative features, and therefore the spatio-temporal CSNN can benefit from additional support from another stream.

The results of applying the two-stream method to the Weizmann dataset, in Tables 8.2, 8.4 and 8.5, show a decline in performance after fusion. As previously mentioned, the extracted feature maps, as shown in Figure 8.5, contain a significant amount of background noise learned by the CSNN with the Weizmann dataset, as opposed to the other datasets. This sensitivity to spatial noise is due to the fact that STDP relies on the precise timing of spikes to update the synaptic weights. In the presence of spatial noise, the larger, noisy pixel values will be translated into earlier spikes by latency coding. This will facilitate their integration by the neuron, causing it to fire erroneously. This leads to unreliable updates in synaptic strengths

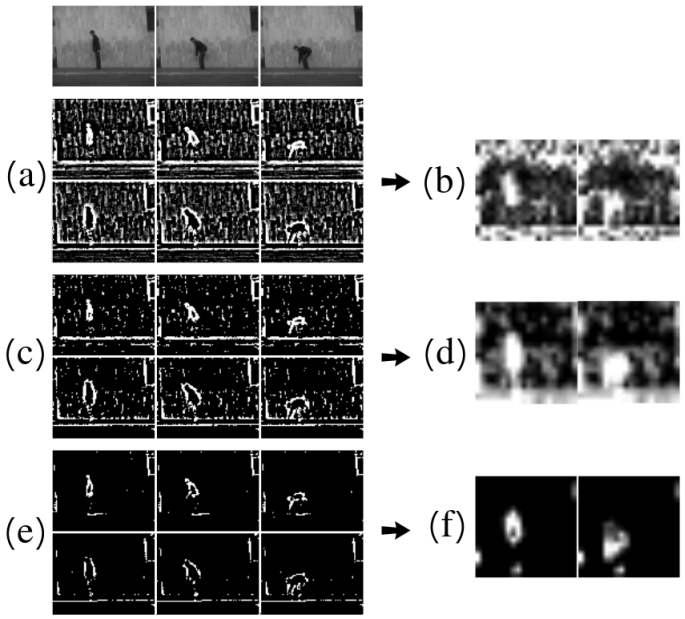


Figure 8.6: A video sample filtered using a DoG filter. We display both the on and off channels with (a) no minimum pixel intensity (b) its corresponding extracted feature map (c) a minimum pixel intensity of 10 (d) its corresponding feature map (e) a minimum pixel intensity of 20 (f) its corresponding feature maps.

and degrades the network performance.

We attempted reducing this noise by applying a Gaussian filter to both the spatial and temporal streams. The noise has slightly decreased, but it was not fully removed, and it still affected the fusion result. Therefore, in order to confirm our hypothesis concerning the impact of noise on the Weizmann dataset, we resort to a different method for noise reduction. We use a minimum pixel intensity as a cutoff value for the DoG filter, thus eliminating all the pixels of the DoG response whose intensity is lower than this value.

When using a minimum pixel intensity of 10, the classification rates of the individual streams increase as a result of noise reduction, as shown in Table 8.8 (a). This occurs with most streams, except the one using frame subtraction. The spatial noise with this stream is not significant; therefore, the loss of information from the cutoff has a greater degrading impact on the performance than the gain of performance attributed to noise reduction. With a cutoff value of 10, the noise is not fully removed, as shown in Figure 8.6 (d), and therefore the fusion is still ineffective. However, when increasing the minimum DoG response to 20, the classification rates

(a) Cutoff = 10		
Raw video (2D conv)	EF (2D conv)	Fused
59.32	65.61	**67.35**
Raw video (2D conv)	OF (2D conv)	Fused
59.32	60.43	**62.91**
Raw video (2D conv)	FS (2D conv)	Fused
59.32	**59.91**	58.80
Raw video (2D conv)	MG (2D conv)	Fused
59.32	**72.17**	59.57
Raw video (2D conv)	Raw video (3D conv)	Fused
59.32	55.61	57.58
(b) Cutoff = 20		
Raw video (2D conv)	EF (2D conv)	Fused
47.83	48.75	**50.43**
Raw video (2D conv)	OF (2D conv)	Fused
47.83	57.21	**59.69**
Raw video (2D conv)	FS (2D conv)	Fused
47.83	48.21	**51.05**
Raw video (2D conv)	MG (2D conv)	Fused
47.83	30.83	**48.72**
Raw video (2D conv)	Raw video (3D conv)	Fused
47.83	**50.71**	48.66

Table 8.8: Classification rates in % of the Weizmann dataset (10 frames per video) over 3 runs: (a) with a minimum pixel intensity of 10, (b) with a minimum pixel intensity of 20.

of the individual streams are significantly decreased, as shown in Table 8.8. This is a result of discarding a large amount of the information while trying to discard the noise. However, with this configuration, the fusion operation is no longer affected by the spatial noise. This successful fusion performance highlights the effects of noise on spiking two-stream methods. This indicates that there is a need for alternative noise reduction methods that improve the feature fusion without compromising the information. Moreover, the feature fusion with 3D convolution is still unsuccessful because the extracted features are still redundant in both streams, which is consistent with our previous conclusions.

8.4 Conclusion

In this chapter, we focused on transposing a popular deep learning method to the spiking domain. We present spiking two-stream methods, which allow 2D CSNNs to extract spatio-temporal information from videos: we implement two-stream methods and a number of spatio-temporal or temporal streams using STDP-based CSNNs, and we gather the following conclusions:

- spiking two-stream methods trained with unsupervised STDP can successfully extract spatio-temporal features from action recognition videos;
- spiking spatial and temporal streams are complementary with all the chosen temporal stream configurations;
- spatio-temporal streams (like 3D CSNNs), used in addition to a spatial stream, lead to information redundancy and does not give effective results with STDP-based two-stream spiking models;
- spiking neural networks trained with STDP and employing latency coding are sensitive to significant spatial noise.

STDP-based unsupervised two-stream SNNs can effectively integrate cost-effective computations with robust spatio-temporal information processing. These models represent a significant stride for video analysis with spiking models. While the performance of these spiking two-stream networks has yet to match that of non-spiking methods, they introduce an initial STDP-based spatio-temporal model capable of extracting features without supervision. In contrast, the non-spiking methods introduced in the literature require large amounts of annotated data for effective training.

Part IV

Conclusion and Future Work

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this manuscript, we addressed video analysis using feature learning with SNNs trained in an unsupervised manner with the STDP learning rule, and we have presented four key contributions.

The first contribution presented in Chapter 5 of this manuscript demonstrates the capability of 2D CSNNs to perform video classification by processing motion information that is modeled in the spatial domain as static representations of motion. We discuss two categories for these static representations, shot-based and frame-based representations. In the case of frame-based representations, they solely model the motion between each two consecutive frames. Therefore, data fusion methods are required to incorporate more than just two frames, because actions cannot be differentiated accurately using only two frames. On the other hand, shot-based methods effectively encapsulate at least one complete action cycle because they inherently contain early fusion. These innovative representations offer a more descriptive modeling of the data, which makes it easier to process by the 2D CSNN.

Our findings highlight that motion modeling is essential for HAR with STDP-based 2D CSNNs. We establish that shot-based methods offer a more complete data representation than frame-based methods, as reflected in the superior performance of the CSNN with these methods. We observed that the sum pooling of output feature maps should not be too severe, otherwise it can hinder the performance of the SVM. We also conclude that frame subtraction can be beneficial in the presence of spatial background noise, but detrimental when dealing with optical-flow-based data or scenarios where spatial information is of significant importance. Finally, We conclude that STDP-based CSNNs need more than simple fusion techniques, like

early and late fusion, to process sequences of frames effectively for HAR.

Our second contribution introduces a spatio-temporal spiking solution for action recognition, which is unsupervised STDP-based 3D CSNNs. This model has a temporal dimension dedicated to time, which enables them to learn spatio-temporal features from videos naturally and without needing extra processes like the ones mentioned in Chapter 5. Through a comparative analysis of 2D CSNNs and 3D CSNNs, both operating on raw frames, we establish that 3D CSNNs have a superior performance to 2D CSNNs for the task of human action recognition. It is important to note that our spatio-temporal features are learned without supervision, which is more difficult than supervised alternatives. Additionally, we conclude that the application of multi-layer architectures necessitates an exhaustive search to identify optimal hyperparameters to facilitate subsequent layer learning. This contribution addresses the challenge of SNN network architecture designs for motion modeling described in Chapter 1, and particularly the limited availability of spatio-temporal STDP-based models tailored for video analysis.

Our third contribution addresses the environmental and hardware limitation concerns that arise from using 3D CSNN architectures. Although these fully spiking solutions have a superior performance to their 2D counterparts, they have a greater number of parameters. We present spiking separated spatial and temporal convolutions, which are a fully spiking solution with fewer parameters than 3D CSNNs. These networks have lower computational complexity and promote potential easier hardware implementation. We conclude that spiking separated convolutions can outperform 3D convolutions due to the simplicity of their kernels, which leads to responding to more patterns and thus firing more spikes. Moreover, we conclude that optimal kernel sizes vary across datasets, depending on the diversity of motion patterns to be captured.

Our fourth and final contribution addresses the lack of network architecture designs for unsupervised STDP-based SNNs. Specifically, a spiking version of two-stream methods, which are state-of-the-art networks for HAR. These methods harness spiking spatial and temporal streams, which are complementary with most chosen temporal stream configurations. However, we observe that spatio-temporal streams, like 3D CSNNs, can cause information redundancy and negatively affect the results with these spiking models. Finally, we conclude that these models are sensitive to significant spatial noise, which degrades their performance.

The findings and contributions presented within this manuscript serve as building blocks towards advancing the task of human action recognition through spiking models trained with unsupervised STDP. By closely examining the limitations of current SNNs, we propose new spiking methods which are able to perform video

feature learning without supervision. Our end goal is to use spiking models to extract important patterns in videos without resorting to costly workarounds. We also explore diverse spiking architectures, like the spiking separated convolutions and two-stream methods, which can improve performance while also being compatible with neuromorphic hardware. These findings are a step forward in using SNNs for video analysis, and open up compelling possibilities for future progress in this area.

9.2 Future work

The preceding chapters have explored various aspects of STDP-based CSNNs in the context of motion modeling and video analysis. As we consider the potential advancements and extensions of this research, it becomes imperative to provide an overview of the perspectives and driving forces that underpin the proposed future work.

In the subsequent sections, we outline the key areas of research and development that hold promise for advancing the current understanding and implementation of SNNs in video analysis. Each subsection highlights specific avenues for exploration, emphasizing the need to introduce spiking solutions for various stages of the video analysis pipeline. By addressing these aspects, we aim to expand the capabilities and effectiveness of spiking neural networks, paving the way for cost-effective video understanding and motion modeling.

9.2.1 Spiking shot-based motion modeling

2D CSNN cannot conserve the temporal information present between frames in videos, because these methods are solely capable of spatial feature extraction, as discussed in Chapter 5. 2D CSNNs need motion modeling methods as a pre-processing step in order to extract the temporal dynamics inherent in videos. Non-spiking motion modeling methods are not natively computationally efficient, and are difficult to implement with hardware. Therefore, a spiking pre-processing method for motion modeling can be useful in conserving the temporal information, without being too costly, and can still be implementable on neuromorphic hardware.

In Chapter 5, we demonstrate the superiority of shot-based methods for video analysis. Therefore, spiking versions of these methods can be interesting avenues for future work. An example of this would be a spiking motion grid, which can be created by replacing the non-spiking optical flow method with another motion modeling method that is implementable on hardware, like the Reichardt detector [207]. Then

the motion displacements can be collected, and the grid can be built as a fully spiking solution.

This would be useful for creating a fully spiking solution for STDP-based spiking two-stream methods. The two-stream methods presented in Chapter 8 use classical pre-processing methods that are non-spiking. However, fully spiking SNNs represent a pivotal step towards harnessing the true power and efficiency of SNNs for complex tasks like video analysis. Therefore, future work shall focus on creating a fully spiking two-stream solution.

9.2.2 Multi-layer SNN

Chapter 6 introduces a multi-layer fully-spiking model which necessitates a careful selection of hyperparameters that govern various aspects of the network's operation, such as learning rates, kernel sizes, and threshold adaptation parameters. Decreasing the complexity of choosing these parameters has the potential to promote the usage of spiking models. This opens the door for other research questions, like what methods can be used to set hyperparameters automatically [208], and how to improve the extraction of relevant space-time features with an end-to-end SNN. The work in [208] serves as a relevant reference in automated hyperparameter tuning that could be extended to suit the specifics of the multi-layer fully-spiking model.

The S3TC architecture discussed in Chapter 7 could prove advantageous in this context as well. This is because S3TCs exhibit higher spiking activity at its output than a 3D convolutional layer, which is beneficial for subsequent layer learning within SNNs.

9.2.3 Multi-stream S3TC architecture

In Chapter 7, we discuss the need to tailor kernel sizes to capture relevant spatio-temporal information from datasets with different characteristics. Therefore, a promising avenue for future work would involve using a multi-stream architecture with separated convolutional networks. Some streams would have a large kernel size, and others would have a small kernel size. This approach would enable the extraction of relevant information about both small and large motion patterns, which leads to better generalization across datasets.

9.2.4 2D CSNN with memory

In previous chapters, we have seen the advantages of spatio-temporal architectures with CSNNs, and we have tackled the problem of reducing the computational costs of CSNNs by using separated convolutions. However, these networks are deeper than 3D CSNNs, which means that they require training two layers instead of one.

As a future work, we would like to create a CSNN that is spatio-temporal, has the number of parameters associated with a 2D CSNN, and does not require two layers to get the spatio-temporal data. Therefore, we opt for a solution that is a spatio-temporal 2D CSNN with memory.

This can be done by adapting the nature of the IF neuron, instead of adding feedback loops for memory. The nature of the neuron could be adapted in a way that instead of re-setting the internal voltage to zero when introducing a new sample, we can instead apply a leak to this voltage, which brings it down to 35% of its value. This allows the neuron to keep some memory from one frame to the next in the video. It would allow the network to process sequential data with a 2D architecture, and capture the motion using the history of the voltage in the neuron.

Appendix A

Acronyms

ADD-STDP: Additive Spatio-Temporal Back-Propagation.
ANN: Artificial Neural Network.
ATIS: Asynchronous Time-Based Image Sensor.
BCM: Bienenstock-Cooper-Munro.
BP: Back Propagation.
BPTT: Back Propagation Through Time.
CC: Composite Channels representation.
CCOA: Composite Channels with Orientation and Amplitude representation.
CMA: Covariance Matrix Adaptation.
CNN: Convolutional Neural Network.
CSNN: Convolutional Spiking Neural Network.
DNN: Deep Neural Network.
DoG: Difference of Gaussians.
DVS: Dynamic Vision Sensor.
DXDY: Displacement in X and Y Directions representation.
EG: Edge Grid.
ESURF: Enhanced Speeded-Up Robust Features.
FC: Fully Connected Layer.
FF: Feed Forward Layer.
FS: Frame Subtraction.
GRN: Gene Regulatory Network.
GRUs: Gated Recurrent Units.
HAR: Human Action Recognition.
HOG: Histogram of Oriented Gradients.
HOF: Histogram of Optical Flow.

HRSNN: Heterogeneous Recurrent Spiking Neural Network.
IF: Integrate-and-Fire.
IoT: Internet of Things.
ISI: Inter-Spike Interval.
KLT: Kanade–Lucas–Tomasi Tracker.
LAT: Leaky Adaptive Threshold.
LIF: Leaky Integrate-and-Fire.
LSTM: Long Short-Term Memory.
LSM: Liquid State Machine.
LTD: Long-Term Depression.
LTP: Long-Term Potentiation.
M-STDP: Mirror Spike Timing-Dependent Plasticity.
MBH: Motion Boundary Histograms.
MG: Motion Grid.
Mult-STDP: Multiplicative Spatio-Temporal Back-Propagation.
OF : Optical Flow.
OA: Orientation and Amplitude Representation.
P-STDP: Probabilistic Spike Timing-Dependent Plasticity.
ReLU: Rectified Linear Unit.
ResNet: Residual Network.
Rev-STDP: Reversed Spike Timing-Dependent Plasticity.
RNN: Recurrent Neural Network.
R-STDP: Reward-Modulated Spike Timing-Dependent Plasticity.
SD: Spatial Domain.
SDSP: Spike-Driven Synaptic Plasticity.
SLAYER: Spike Layer Error Reassignment.
SFA: Spike Frequency Adaptation.
SIFT: Scale-Invariant Feature Transform.
SNN: Spiking Neural Network.
S-STDP: Stable Spike Timing-Dependent Plasticity.
SVM: Support Vector Machine.
S3TC: Spiking Separated Spatial and Temporal Convolution.
STS-ResNet: Spatio-Temporal Spiking Residual Network.
STBP: Spatio-Temporal Back-Propagation.
STDP: Spike Timing-Dependent Plasticity.
SURF: Speeded-Up Robust Features.
TSRNN: Temporal Spiking Recurrent Neural Network.
T-STDP: Triplet Spike Timing-Dependent Plasticity.

TD: Temporal Domain.
TTFS: Time-to-First Spike encoding.
WTA: Winner-Takes-All.

Appendix B

List of Symbols

Generic Parameters:

x	Pixel coordinates in the horizontal dimension.
y	Pixel coordinates in the vertical dimension.
$\mathcal{U}$	Uniform distribution.
$\mathcal{N}$	Normal distribution.
μ	Mean of distribution.
σ^2	Variance of distribution.
W_{xh}	Weight for the input-to-hidden connection.
W_{hh}	Hidden-to-hidden recurrent connection.
W_{hy}	Hidden-to-output connection.
b_h	Bias for the hidden state.
b_y	Bias for the output state.

Network parameters:

l_w	Width of the input frames.
l_h	Height of the input frames.
l_c	Number of input channels.
l_{td}	Temporal depth of the input.
n_{sampling}	Number of input patches.
X	Input vector.
$x \in [0, 1]$	Input value.
f_w	Width of the filter.
f_h	Height of the filter.
f_{td}	Temporal depth of the filter.

f_k	Number of filters.
l_{pad}	Padding pixels.
l_{stride}	Stride of the kernel.
P_{2D}	Number of parameters of a 2D spiking convolution.
P_{3D}	Number of parameters of a 3D spiking convolution.
P_{S3TC}	Number of parameters of an S3TC.
n_c	Number of channels.

Pre-Processing Parameters:

θ_m	Threshold of the sum of pixel frames channels.
DoG	Difference-of-Gaussians filter.
G	Gaussian kernel.
σ_1, σ_2	Standard deviations of Gaussians.
x_{on}	On channel.
x_{off}	Off channel.
s	Size of Gaussian kernel.

Video and frame processing:

n	Index in a sequence.
I	Input frame.
$c1, c2, c3$	Image channels.
I_o	Output frames.
r	Row index.
c	Column index.
P_d	Pixel difference.
p	Number of rows of a shot-based grid.
q	Number of columns of a shot-based grid.
M^u	Upwards displacement of the optical flow vector.
M^d	Downwards displacement of the optical flow vector.
M^l	Displacement to the left of the optical flow vector.
M^r	Displacement to the right of the optical flow vector.
$*$	Convolution operator.
OF_x	Horizontal component of the optical flow vector.
OF_y	Vertical component of the optical flow vector.

Neuron and synaptic plasticity:

$\mathrm{f_a}$	Activation function.
W	Synaptic weight matrix.
b	Bias.
V_m	Neuron membrane potential.
V_{th}	Neuronal threshold.
$\hat{t}$	Target timestamp.
Δ_{th}	Change in neural threshold.
x	Coordinates of the spike in the width dimension.
y	Coordinates of the spike in the height dimension.
z	Coordinates of the spike in the time dimension.
k	Coordinates of the spike in the channel dimension.
$n()$, $j()$, $m()$	Functions that map the location of the input neuron to the corresponding location in the weight matrix.
Δ_w	Change in synaptic weight.
η_w	Weight learning rate.
η_{w_+}	A constant learning rate for LTP.
η_{w_-}	A constant learning rate for LTD.
a_+	Amplification parameter of LTP.
a_-	Amplification parameter of LTD.
τ_{STDP}	Time constant for the STDP learning window.
t	Neuron firing timestamp.
η_{th}	Threshold learning rate.
l_d	Number of neurons that are in competition in the layer.
t_i	Firing timestamp of neuron i.
t_{pre}	Timestamp for input spikes.
t_{post}	Timestamp for output spikes.
$\min\{t_0, ..., t_N\}$	Minimum timestamp.
$\mathrm{th_{min}}$	Minimum possible threshold value.
$\mathcal{E}$	Set of incoming spikes.
$\mathcal{E}_s$	Set of incoming spikes of neurons in the spatial neighborhood.
$\mathcal{E}_t$	Set of incoming spikes of neurons in the temporal neighborhood.
i	Spike index.
δt	Relative timing of a pre-synaptic and post-synaptic spike pair.
f_s	Kernel of spikes.

Neural coding properties:

$t_{\text{exposition}}$	Presentation duration for one sample.
$r(x_i)$	The rank of the input.
ISI	Inter-spike interval.
N_s	Number of spikes in a burst.
T_{max}	Maximum inter-spike interval.
T_{min}	Minimum inter-spike interval.
w_s	Variable spike weight for phase coding.
ϕ	Phase.

Bibliography

[1] Ericsson. Ericsson Mobility Report, 2023.

[2] Itsaso Rodríguez-Moreno, José María Martínez-Otzeta, Basilio Sierra, Igor Rodriguez, and Ekaitz Jauregi. Video Activity Recognition: State-of-the-Art. Sensors, 19(14):3160, 2019.

[3] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3D Convolutional Neural Networks for Human Action Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 35(1):221–231, 2013.

[4] Karen Simonyan and Andrew Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. In International Conference on Neural Information Processing Systems (NIPS), volume 1, page 568–576, 2014.

[5] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the Carbon Emissions of Machine Learning. ArXiv, arXiv:1910.09700 [cs.CY], 2019.

[6] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Deep Learning in NLP. AAAI Conference on Artificial Intelligence, abs/1906.02243, 2019.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-scale Hierarchical Image Database. In Conference on Computer Vision and Pattern Recognition (CVPR), pages 248–255, 2009.

[8] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep Learning in Spiking Neural Networks. Neural Networks (NN), 111:47 – 63, 2019.

[9] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. A Survey of Neuromorphic Computing and Neural Networks in Hardware. ArXiv, arXiv:1705.06963 [cs.NE], 2017.

[10] Wolfgang Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. Neural Networks (NN), 10(9):1659–1671, 1997.

[11] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. Science, 345(6197):668–673, 2014.

[12] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The SpiNNaker Project. Institute of Electrical and Electronics Engineers, 102(5):652–665, 2014.

[13] Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel. The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity. Frontiers in Neuroscience, 16, 2022.

[14] Lei Deng, Guanrui Wang, Guoqi Li, Shuangchen Li, Ling Liang, Maohua Zhu, Yujie Wu, Zheyu Yang, Zhe Zou, Jing Pei, Zhenzhi Wu, Xing Hu, Yufei Ding, Wei He, Yuan Xie, and Luping Shi. Tianjic: A Unified and Scalable Chip Bridging Spike-Based and Continuous Neural Computation. IEEE Journal of Solid-State Circuits, 55(8):2228–2246, 2020.

[15] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. IEEE Micro, 38(1):82–99, 2018.

[16] A. Joubert, B. Belhadj, O. Temam, and R. Héliot. Hardware spiking neurons design: Analog or digital? In The 2012 International Joint Conference on Neural Networks (IJCNN), pages 1–5, 2012.

[17] Amar Shrestha, Haowen Fang, Zaidao Mei, Daniel Patrick Rider, Qing Wu, and Qinru Qiu. A Survey on Neuromorphic Computing: Models and Hardware. IEEE Circuits and Systems Magazine, 22(2):6–35, 2022.

[18] Qinyu Cheni, Bodo Rueckauer, Li Li, Tobi Delbruck, and Shih-Chii Liu. Reducing Latency in a Converted Spiking Video Segmentation Network. In IEEE International Symposium on Circuits and Systems (ISCAS), pages 1–5, 2021.

[19] Manon Dampfhoffer, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel. Backpropagation-Based Learning Techniques for Deep Spiking Neural Networks: A Survey. IEEE Transactions on Neural Networks and Learning Systems (NNLS), pages 1–16, 2023.

[20] Ali Shahin Shamsabadi, Brij Mohan Lal Srivastava, Aurélien Bellet, Nathalie Vauquier, Emmanuel Vincent, Mohamed Maouche, Marc Tommasi, and Nicolas Papernot. Differentially Private Speaker Anonymization. Proceedings on Privacy Enhancing Technologies, 2023(1):98–114, January 2023.

[21] Neha Dahiya and Chander Kant. Biometrics security concerns. In 2012 Second International Conference on Advanced Computing and Communication Technologies, pages 297–302, 2012.

[22] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128× 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor. IEEE Journal of Solid-State Circuits (IJSSC), 43(2):566–576, 2008.

[23] Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data. Frontiers in Neuroscience (FN), 11, 2017.

[24] Youngeun Kim and Priyadarshini Panda. Optimizing Deeper Spiking Neural Networks for Dynamic Vision Sensing. Neural Networks (NN), 144:686–698, 2021.

[25] Federico Paredes-Vallés, Kirk Y. W. Scheper, and Guido C. H. E. de Croon. Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow

Estimation: From Events to Global Motion Perception. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 42(8):2051–2064, 2020.

[26] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A Low Power, Fully Event-Based Gesture Recognition System. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 7388–7397, 2017.

[27] Dighanchal Banerjee, Smriti Rani, Arun M. George, Arijit Chowdhury, Sounak Dey, Arijit Mukherjee, Tapas Chakravarty, and Arpan Pal. Application of Spiking Neural Networks for Action Recognition from Radar Data. In International Joint Conference on Neural Networks (IJCNN), pages 1–10, 2020.

[28] Yuhang Li, Ruokai Yin, Hyoungseob Park, Youngeun Kim, and Priyadarshini Panda. Wearable-based Human Activity Recognition with Spatio-Temporal Spiking Neural Networks. ArXiv, arXiv:2212.02233 [cs.NE], 2022.

[29] Daniela Micucci, Marco Mobilio, and Paolo Napoletano. UniMiB SHAR: A Dataset for Human Activity Recognition Using Acceleration Data from Smartphones. Applied Sciences (AS), 7(10):1101, 2017.

[30] D. Anguita, Alessandro Ghio, L. Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition using Smartphones. In The European Symposium on Artificial Neural Networks (ESANN), 2013.

[31] Pierre Falez. Improving Spiking Neural Networks Trained with Spike Timing Dependent Plasticity for Image Recognition. In Ph.D. Thesis, 2019.

[32] Timothée Masquelier and Simon J Thorpe. Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity. PLOS Computational Biology, 3(2):1–11, 2007.

[33] Mingliang Zhai, Xuezhi Xiang, Ning Lv, and Xiangdong Kong. Optical Flow and Scene Flow Estimation: A Survey. Pattern Recognition, 114:107861, 2021.

[34] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going Deeper in Spiking Neural Networks: VGG and Residual Architectures. Frontiers in Neuroscience (FN), 13, 2019.

[35] Taras Iakymchuk, Alfredo Rosado-Muñoz, Juan F Guerrero-Martínez, Manuel Bataller-Mompeán, and Jose V Francés-Víllora. Simplified Spiking Neural Network Architecture and STDP Learning Algorithm Applied to Image Classification. EURASIP Journal on Image and Video Processing (JIVP), 2015(1):1–11, 2015.

[36] Xiang Wang, Jie Yang, and Nikola K. Kasabov. Integrating Spatial and Temporal Information for Violent Activity Detection from Video Using Deep Spiking Neural Networks. Sensors, 23(9), 2023.

[37] Michael Pfeiffer and Thomas Pfeil. Deep Learning With Spiking Neurons: Opportunities and Challenges. Frontiers in Neuroscience (FN), 12, 2018.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems (NIPS), volume 25, page 84–90, 2012.

[39] Ronald Poppe. A Survey on Vision-based Human Action Recognition. Image and Vision Computing (IVC), 28(6):976–990, 2010.

[40] Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A Review of Human Activity Recognition Methods. Frontiers in Robotics and AI (FRAI), 2, 2015.

[41] Shang-Chai Yang, Ming-Lun Wu, and Yuan-Kai Wang. Human Activity Recognition with Spiking Neural Network. In IEEE 11th Global Conference on Consumer Electronics (GCCE), pages 482–483, 2022.

[42] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. Smart Devices Are Different: Assessing and MitigatingMobile Sensing Heterogeneities for Activity Recognition. In Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (CENSS), page 127–140, 2015.

[43] Dina Chahyati, Mohamad Ivan Fanany, and Aniati Murni Arymurthy. Tracking People by Detection Using CNN Features. Procedia Computer Science (PCS), 124:167–172, 2017.

[44] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple Online and Realtime Tracking. In 2016 IEEE International Conference on Image Processing (ICIP), 2016.

[45] Fangao Zeng, Bin Dong, Yuang Zhang, Tiancai Wang, Xiangyu Zhang, and Yichen Wei. MOTR: End-to-End Multiple-Object Tracking with Transformer. European Conference Computer Vision (ECCV), page 659–675, 2022.

[46] Wafa Lejmi, Mohamed Ali Mahjoub, and Anouar Ben Khalifa. Event Detection in Video Sequences: Challenges and Perspectives. In International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pages 682–690, 2017.

[47] Junghoon Chae, Dennis Thom, Harald Bosch, Yun Jang, Ross Maciejewski, David S. Ebert, and Thomas Ertl. Spatiotemporal Social Media Analytics for Abnormal Event Detection and Examination Using Seasonal-trend Decomposition. In IEEE Conference on Visual Analytics Science and Technology (VAST), pages 143–152, 2012.

[48] Mahmoud Taha, Abdelwahab Alsammak, and Ahmed Zaky. Inspector-Net: Transformer Network for Violence Detection in Animated Cartoon. Engineering Research Journal - Faculty of Engineering (Shoubra), 52(2):114–119, 2023.

[49] Ge-Peng Ji, Guobao Xiao, Yu-Cheng Chou, Deng-Ping Fan, Kai Zhao, Geng Chen, and Luc Van Gool. Video Polyp Segmentation: A Deep Learning Perspective. Machine Intelligence Research (MIR), 19(6):531–549, 2022.

[50] Mahesh Ramachandran, Ashok Veeraraghavan, and Rama Chellappa. CHAPTER 5 - Video Stabilization and Mosaicing. In Al Bovik, editor, The Essential Guide to Video Processing (EGVP), pages 109–140, 2009.

[51] Thierry Bouwmans. Traditional and Recent Approaches in Background Modeling for Foreground Detection: An Overview. Computer Science Review (CSR), 11-12:31–66, 2014.

[52] Damien Bouchabou, Christophe Lohr, Ioannis Kanellos, and Sao Mai Nguyen. Human Activity Recognition (HAR) in Smart Homes. Encyclopedia, page https://encyclopedia.pub/17108, November 2021.

[53] Rizwan Chaudhry, Avinash Ravichandran, Gregory Hager, and Rene Vidal. Histograms of Oriented Optical Flow and Binet-Cauchy Kernels on Nonlinear Dynamical Systems for the Recognition of Human Actions. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1932–1939, 2009.

[54] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing Human Actions: A Local SVM Approach. In International Conference on Pattern Recognition (ICPR), volume 3, page 32–36, 2004.

[55] Lena Gorelick, Moshe Blank, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as Space-Time Shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 29(12):2247–2253, 2007.

[56] Daniel Weinland, Rémi Ronfard, and Edmond Boyer. Free Viewpoint Action Recognition Using Motion History Volumes. Computer Vision and Image Understanding (CVIU), 104(2-3):249–257, 2006.

[57] Mikel D. Rodriguez, Javed Ahmed, and Mubarak Shah. Action MACH: A Spatio-temporal Maximum Average Correlation Height Filter for Action Recognition. In IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2008.

[58] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. ArXiv, arXiv:1212.0402 [cs.CV], 2012.

[59] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: A Large Video Database for Human Motion Recognition. In International Conference on Computer Vision (ICCV), pages 2556–2563, 2011.

[60] Marcin Marszalek, Ivan Laptev, and Cordelia Schmid. Actions in Context. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2929–2936, 2009.

[61] Lucas Smaira, João Carreira, Eric Noland, Ellen Clancy, Amy Wu, and Andrew Zisserman. A Short Note on the Kinetics-700-2020 Human Action Dataset. ArXiv, arXiv:2010.10864 [cs.CV], 2010.

[62] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/Kanade Meets Horn/Schunck: Combining Local and Global Optic Flow Methods. International Journal of Computer Vision (IJCV), 61(3):1–21, 2005.

[63] Gunnar Farnebäck. Two-Frame Motion Estimation Based on Polynomial Expansion. In Josef Bigun and Tomas Gustavsson, editors, Image Analysis, pages 363–370, 2003.

[64] Laptev and Lindeberg. On Space-Time Interest Points. In Proceedings Ninth IEEE International Conference on Computer Vision (CV), pages 432–439 vol.1, 2003.

[65] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Dense Trajectories and Motion Boundary Descriptors for Action Recognition. International Journal of Computer Vision (IJCV), 103, 2013.

[66] Amir H. Shabani, David A. Clausi, and John S. Zelek. Evaluation of Local Spatio-temporal Salient Feature Detectors for Human Action Recognition. In Ninth Conference on Computer and Robot Vision (CRV), pages 468–475, 2012.

[67] C. G. Harris and M. Stephens. A Combined Corner and Edge Detector. In Alvey Vision Conference (AVC), 1988.

[68] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie. Behavior Recognition via Sparse Spatio-temporal Features. In IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance, pages 65–72, 2005.

[69] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision (IJCV), 60(2):91–110, 2004.

[70] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-Dimensional Sift Descriptor and Its Application to Action Recognition. In Proceedings of the 15th ACM International Conference on Multimedia (ICM), page 357–360, 2007.

[71] Geert Willems, Tinne Tuytelaars, and Luc Gool. An Efficient Dense and Scale-Invariant Spatio-Temporal Interest Point Detector. In Proceedings of the 10th European Conference on Computer Vision: Part II (CV), page 650–663, 2008.

[72] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). Computer Vision and Image Understanding (CVIU), 110(3):346–359, 2008. Similarity Matching in Computer Vision and Multimedia.

[73] Bruce D. Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In International Joint Conference on Artificial Intelligence (IJCAI), volume 2, pages 674–679, 1981.

[74] Jianbo Shi and Tomasi. Good Features to Track. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 593–600, 1994.

[75] Ross Messing, Chris Pal, and Henry Kautz. Activity Recognition using the Velocity Histories of Tracked Keypoints. In IEEE 12th International Conference on Computer Vision (CV), pages 104–111, 2009.

[76] Ju Sun, Yadong Mu, Shuicheng Yan, and Loong-Fah Cheong. Activity Recognition using Dense Long-duration Trajectories. In IEEE International Conference on Multimedia and Expo (ICME), pages 322–327, 2010.

[77] Alexander Klaser, Marcin Marszalek, and Cordelia Schmid. A Spatio-Temporal Descriptor Based on 3D-Gradients. In British Machine Vision Conference (BMVC), pages 275:1–10, 2008.

[78] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), volume 1, pages 886–893 vol. 1, 2005.

[79] Janez Perš, Vildana Sulić, Matej Kristan, Matej Perše, Klemen Polanec, and Stanislav Kovačič. Histograms of Optical Flow for Efficient Representation of Body Motion. Pattern Recognition Letters (PRL), 31(11):1369–1376, 2010.

[80] Ivan Laptev, Marcin Marszalek, Cordelia Schmid, and Benjamin Rozenfeld. Learning Realistic Human Actions from Movies. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–8, 2008.

[81] Xiaojiang Peng, Limin Wang, Xingxing Wang, and Yu Qiao. Bag of Visual Words and Fusion Methods for Action Recognition: Comprehensive Study and Good Practice. Computer Vision and Image Understanding, 150:109–125, 2016.

[82] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. Nature, 323:533–536, 1986.

[83] Pierre Falez, Pierre Tirilly, Ioan Marius Bilasco, Philippe Devienne, and Pierre Boulet. Multi-layered Spiking Neural Network with Target Timestamp Threshold Adaptation and STDP. In International Joint Conference on Neural Networks (IJCNN), 2019.

[84] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a Convolutional Neural Network. In International Conference on Engineering and Technology (ICET), pages 1–6, 2017.

[85] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann Lecun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. International Conference on Learning Representations (ICLR), 2013.

[86] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-Scale Video Classification with Convolutional Neural Networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1725–1732, 2014.

[87] Rafael C. González and Richard E. Woods. Digital Image Processing, 3rd Edition. Pearson Education, 2008.

[88] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning Spatiotemporal Features with 3D Convolutional Networks. In IEEE International Conference on Computer Vision (ICCV), pages 4489–4497, 2015.

[89] Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning Spatiotemporal Features with 3D Convolutional Networks. ArXiv, arXiv:1412.0767 [cs.CV], 2014.

[90] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. ArXiv, arXiv:1704.04861 [cs.CV], 2017.

[91] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pages 1251–1258, 2017.

[92] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A Closer Look at Spatiotemporal Convolutions for Action Recognition. In 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pages 6450–6459, 2018.

[93] Abdulmajid Murad and Jae-Young Pyun. Deep Recurrent Neural Networks for Human Activity Recognition. Sensors, 17(11), 2017.

[94] Munir Husein and Il-Yop Chung. Day-Ahead Solar Irradiance Forecasting for Microgrids Using a Long Short-Term Memory Recurrent Neural Network: A Deep Learning Approach. Energies, 12:1856, 2019.

[95] P.J. Werbos. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10):1550–1560, 1990.

[96] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. Neural Computation (NC), 9:1735–1780, 1997.

[97] Jiaxin Cai, Junlin Hu, Xin Tang, Tzu-Yi Hung, and Yap-Peng Tan. Deep Historical Long Short-term Memory Network for Action Recognition. Neurocomputing (NC), 407:428–438, 2020.

[98] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, 2014.

[99] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. International Conference on Neural Information Processing Systems (NIPS) Workshop, abs/1412.3555, 2014. cite arxiv:1412.3555Comment: Presented in NIPS 2014 Deep Learning and Representation Learning Workshop.

[100] Piotr Antonik, Nicolas Marsal, Daniel Brunner, and Damien Rontani. Human Action Recognition with a Large-scale Brain-inspired Photonic Computer. Nature Machine Intelligence (NMI), 1(11):530–537, 2019.

[101] Benjamin Schrauwen, David Verstraeten, and Jan Campenhout. An Overview of Reservoir Computing: Theory, Applications and Implementations. In Proceedings of the 15th European Sympsosium on Artificial Neural Networks (ESANN), pages 471–482, 2007.

[102] Rohit Nale, Mahesh Sawarbandhe, Naveen Chegogoju, and Vishal Satpute. Suspicious human activity detection using pose estimation and lstm. In 2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation (IRIA), pages 197–202, 2021.

[103] Robin M. Schmidt. Recurrent Neural Networks (RNNs): A gentle Introduction and Overview. ArXiv, arXiv:1912.05911 [cs.LG], 2019.

[104] Martin N. Hebart and Guido Hesselmann. What Visual Information Is Processed in the Human Dorsal Stream? Journal of Neuroscience (JN), 32(24):8107–8109, 2012.

[105] Qianqian Xiong, Jianjing Zhang, Peng Wang, Dongdong Liu, and Robert X. Gao. Transferable Two-stream Convolutional Neural Network for Human Action Recognition. Journal of Manufacturing Systems (JMS), 56:605–614, 2020.

[106] Shahbaz Khan, Ali Hassan, Farhan Hussain, Aqib Perwaiz, Farhan Riaz, Maazen Alsabaan, and Wadood Abdul. Enhanced spatial stream of two-stream network using optical flow for human action recognition. Applied Sciences, 13:8003, 07 2023.

[107] Biao Guo, Mingrui Liu, Qian He, and Ming Jiang. Two-stream spatial-temporal auto-encoder with adversarial training for video anomaly detection. IEEE Access, PP:1–1, 01 2023.

[108] Amany Abdelbaky and Saleh Aly. Two-stream spatiotemporal feature fusion for human action recognition. The Visual Computer, 37, 07 2021.

[109] Ju-Chin Chen, Chien-Yi Lee, Peng-Yu Huang, and Cheng-Rong Lin. Driver behavior analysis via two-stream deep convolutional neural network. Applied Sciences, 10:1908, 03 2020.

[110] Hongsong Wang and Liang Wang. Modeling Temporal Dynamics and Spatial Configurations of Actions Using Two-Stream Recurrent Neural Networks. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3633–3642, 2017.

[111] Hong Liu, Juanhui Tu, and Mengyuan Liu. Two-Stream 3D Convolutional Neural Network for Skeleton-Based Action Recognition. ArXiv, arXiv:1705.08106 [cs.CV], 2017.

[112] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional Two-Stream Network Fusion for Video Action Recognition. In International Conference on Computer Vision and Pattern Recognition (CVPR), pages 1933–1941, 2016.

[113] Romain Belmonte, Nacim Ihaddadene, Pierre Tirilly, Ioan Marius Bilasco, and Chaabane Djeraba. Video-Based Face Alignment With Local Motion Modeling. In Winter Conference on Applications of Computer Vision (WACV), pages 2106–2115, 2019.

[114] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep Learning for Sensor-Based Activity Recognition: A Survey. Pattern Recognition Letters (PRL), 119:3–11, 2019.

[115] J. Arunnehru, G. Chamundeeswari, and S. Bharathi. Human Action Recognition using 3D Convolutional Neural Networks with 3D Motion Cuboids in Surveillance Videos. Procedia Computer Science (CS), 133:471–477, 2018.

[116] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential Deep Learning for Human Action Recognition. In B. Lepri A.A. Salah, editor, International Workshop on Human Behavior Understanding (HBU), pages 29–39. Springer, November 2011.

[117] V. Veeriah, N. Zhuang, and G. Qi. Differential Recurrent Neural Networks for Action Recognition. In 2015 IEEE International Conference on Computer Vision (ICCV), pages 4041–4049, 2015.

[118] Congcong Liu, Jie Ying, Haima Yang, Xing Hu, and Jin Liu. Improved Human Action Recognition Approach Based on Two-Stream Convolutional Neural Network Model. The Visual Computer (VC), 37, 2021.

[119] Amin Ullah, Jamil Ahmad, Khan Muhammad, Muhammad Sajjad, and Sung Wook Baik. Action Recognition in Video Sequences using Deep Bi-Directional LSTM With CNN Features. IEEE Access, 6:1155–1166, 2018.

[120] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional Two-Stream Network Fusion for Video Action Recognition. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1933–1941, 2016.

[121] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the Difficulty of Training Recurrent Neural Networks. In Proceedings of the 30th International Conference on Machine Learning (ICML), volume Volume 28, page III–1310–III–1318, 2013.

[122] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-term Dependencies with Gradient Descent is Difficult. IEEE Transactions on Neural Networks (NN), 5(2):157–166, 1994.

[123] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. In International Conference on Neural Information Processing Systems (NIPS), volume 1, page 1135–1143, 2015.

[124] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep Learning with Limited Numerical Precision. In International Conference on International Conference on Machine Learning, volume 37, page 1737–1746, 2015.

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

[126] Zhenshan Bing, Ivan Baumann, Zhuangyi Jiang, Kai Huang, Caixia Cai, and Alois Knoll. Supervised Learning in SNN via Reward-Modulated Spike-Timing-Dependent Plasticity for a Target Reaching Vehicle. Frontiers in Neurorobotics (FN), 13, 2019.

[127] E.M. Izhikevich. Which Model to Use for Cortical Spiking Neurons? IEEE Transactions on Neural Networks (NN), 15(5):1063–1070, 2004.

[128] A. L. Hodgkin and A. F. Huxley. A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve. The Journal of Physiology (JP), 117(4):500–544, 1952.

[129] Casey Henley. Action Potential Speed, 2022.

[130] William Erik Sherwood. FitzHugh–Nagumo Model. In Dieter Jaeger and Ranu Jung, editors, Encyclopedia of Computational Neuroscience (ECN), pages 1–11, 2013.

[131] James Hindmarsh and R Rose. A Model of Neuronal Bursting Using Three Coupled First Order Differential Equations. Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain), 221:87–102, 1984.

[132] E.M. Izhikevich. Simple Model of Spiking Neurons. IEEE Transactions on Neural Networks (NN), 14(6):1569–1572, 2003.

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

[134] Vladimir Kornijcuk, Hyungkwang Lim, Jun Yeong Seok, Guhyun Kim, Seong Keun Kim, Inho Kim, Byung Joon Choi, and Doo Seok Jeong.

Leaky Integrate-and-Fire Neuron Circuit Based on Floating-Gate Integrator. Frontiers in Neuroscience (FN), 10:212, 2016.

[135] Siying Liu, Vincent C. H. Leung, and Pier Luigi Dragotti. First-spike coding promotes accurate and efficient spiking neural networks for discrete events with rich temporal structures. Frontiers in Neuroscience, 17, 2023.

[136] Wenzhe Guo, Mohammed E. Fouda, Ahmed M. Eltawil, and Khaled Nabil Salama. Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems. Frontiers in Neuroscience (FN), 15, 2021.

[137] Simon Thorpe and Jacques Gautrais. "Rank Order Coding", pages 113–118. Springer US, 1998.

[138] Arnaud Delorme, Laurent Perrinet, and Simon J. Thorpe. Networks of Integrate-and-fire Neurons Using Rank Order Coding B: Spike Timing Dependent Plasticity and Emergence of Orientation Selectivity. Neurocomputing (NC), 38-40:539–545, 2001. Computational Neuroscience: Trends in Research 2001.

[139] Tim Gollisch and Markus Meister. Rapid Neural Coding in the Retina with Relative Spike Latencies. Science, 319(5866):1108–1111, 2008.

[140] Roland S Johansson and Ingvars Birznieks. First Spikes in Ensembles of Human Tactile Afferents Code Complex Spatial Fingertip Events. Nature Neuroscience (NN), 7(2):170–177, 2004.

[141] John O'Keefe and Michael L. Recce. Phase Relationship between Hippocampal Place Units and the EEG Theta Rhythm. Hippocampus, 3(3):317–330, 1993.

[142] Jaehyun Kim, Heesu Kim, Subin Huh, Jinho Lee, and Kiyoung Choi. Deep Neural Networks with Weighted Spikes. Neurocomputing (NC), 311:373–386, 2018.

[143] Eugene M. Izhikevich, Niraj S. Desai, Elisabeth C. Walcott, and Frank C. Hoppensteadt. Bursts as a Unit of Neural Information: Selective Communication via Resonance. Trends in Neurosciences (TN), 26(3):161–167, 2003.

[144] Alexander Kugele, Thomas Pfeil, Michael Pfeiffer, and Elisabetta Chicca. Efficient Processing of Spatio-Temporal Data Streams With Spiking Neural Networks. Frontiers in Neuroscience (FN), 14, 2020.

[145] A. V. Gavrilov and K. O. Panchenko. Methods of Learning for Spiking Neural Networks. A survey. In International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE), volume 02, pages 455–460, 2016.

[146] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures. Frontiers in Neuroscience (FN), 14, 2020.

[147] Alex Vigneron and Jean Martinet. A Critical Survey of STDP in Spiking Neural Networks for Pattern Recognition. In International Joint Conference on Neural Networks (IJCNN), pages 1–9, 2020.

[148] Bodo Rueckauer, I. Lungu, Yuhuang Hu, M. Pfeiffer, and Shih-Chii Liu. Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification. Frontiers in Neuroscience (FN), 11, 2017.

[149] Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy Spiking Deep Networks through Weight and Threshold Balancing. In 2015 International Joint Conference on Neural Networks (IJCNN), pages 1–8, 2015.

[150] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training Deep Spiking Neural Networks Using Backpropagation. Frontiers in Neuroscience (FN), 10, 2016.

[151] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. Frontiers in Neuroscience (FN), 12, 2018.

[152] Ali Lotfi Rezaabad and Sriram Vishwanath. Long Short-Term Memory Spiking Networks and Their Applications. In International Conference on Neuromorphic Systems 2020, 2020.

[153] Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning. Frontiers in Neuroscience (FNS), 12, 2018.

[154] D. O. Hebb. The Organization of Behavior: a Neuropsychological Theory. L. Erlbaum Associates, 2002.

[155] Mengting Lan, Xiaogang Xiong, Zixuan Jiang, and Yunjiang Lou. PC-SNN: Supervised Learning with Local Hebbian Synaptic Plasticity based on Predictive Coding in Spiking Neural Networks. CoRR, abs/2211.15386, 2022.

[156] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J. Thorpe, and Timothée Masquelier. STDP-based Spiking Deep Convolutional Neural Networks for Object Recognition. Neural Networks (NN), 99:56–67, 2018.

[157] Manu Srinath Halvagal and Friedemann Zenke. The Combination of Hebbian and Predictive Plasticity Learns Invariant Object Representations in Deep Sensory Networks. bioRxiv, 2023.

[158] Stefano Fusi. Spike-driven Synaptic Plasticity for Learning Correlated Patterns of Mean Firing Rates. Reviews in the Neurosciences (RN), 14:73–84, 2003.

[159] Adithya Gurunathan and Laxmi R Iyer. Spurious Learning in Networks with Spike Driven Synaptic Plasticity. In International Conference on Neuromorphic Systems (ICNS), 2020.

[160] Sen Song, Kenneth D. Miller, and L. F. Abbott. Competitive Hebbian Learning Through Spike-timing-dependent Synaptic Plasticity. Nature Neuroscience (NNS), 3(9):919–926, 2000.

[161] S. Jeba Berlin and Mala John. R-STDP Based Spiking Neural Network for Human Action Recognition. Applied Artificial Intelligence (AAI), 34(9):656–673, 2020.

[162] Flavio Fröhlich. Chapter 4 - Synaptic Plasticity. In Flavio Fröhlich, editor, Network Neuroscience (NN), pages 47–58, 2016.

[163] Pierre Falez, Pierre Tirilly, Ioan Marius Bilasco, Philippe Devienne, and Pierre Boulet. Unsupervised Visual Feature Learning with Spike-timing-dependent Plasticity: How Far are We From Traditional Feature Learning Approaches? Pattern Recognition, 93:418–429, 2019.

[164] Guo-qiang Bi and Mu-ming Poo. Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type. Journal of Neuroscience (JN), 18(24):10464–10472, 1998.

[165] Matthieu Gilson and Tomoki Fukai. Stability versus Neuronal Specialization for STDP: Long-Tail Weight Distributions Solve the Dilemma. PLOS ONE, 6(10):1–18, 2011.

[166] Kendra S. Burbank. Mirrored STDP Implements Autoencoder Learning in a Network of Spiking Neurons. PLOS Computational Biology (CB), 11(12):1–25, 2015.

[167] Pierre Falez, Pierre Tirilly, Ioan Marius Bilasco, Philippe Devienne, and Pierre Boulet. Mastering the output frequency in spiking neural networks. In International Joint Conference on Neural Networks (IJCNN), 2018.

[168] Amirhossein Tavanaei, Timothée Masquelier, and Anthony S. Maida. Acquisition of visual features through probabilistic spike-timing-dependent plasticity. In 2016 International Joint Conference on Neural Networks (IJCNN), pages 307–314, 2016.

[169] Kendra S. Burbank and Gabriel Kreiman. Depression-Biased Reverse Plasticity Rule Is Required for Stable Learning at Top-Down Connections. PLOS Computational Biology (CB), 8(3):1–16, 2012.

[170] Dalius Krunglevicius. Modified STDP Triplet Rule Significantly Increases Neuron Training Stability in the Learning of Spatial Patterns. Advances in Artificial Neural Systems, 2016:1687–7594, 2016.

[171] Julijana Gjorgjieva, Claudia Clopath, Juliette Audet, and Jean-Pascal Pfister. A Triplet Spike-timing–dependent Plasticity Model Generalizes the Bienenstock–Cooper–Munro Rule to Higher-order Spatiotemporal Correlations. Proceedings of the National Academy of Sciences (PNAS), 108(48):19383–19388, 2011.

[172] Gilad Silberberg and Henry Markram. Disynaptic Inhibition between Neocortical Pyramidal Cells Mediated by Martinotti Cells. Neuron, 53(5):735–746, 2007.

[173] Dongcheng Zhao, Yi Zeng, and Yang Li. BackEISNN: A Deep Spiking Neural Network with Adaptive Self-feedback and Balanced Excitatory–inhibitory Neurons. Neural Networks (NN), 154:68–77, 2022.

[174] Damien Querlioz, Olivier Bichler, and Christian Gamrat. Simulation of a Memristor-based Spiking Neural Network Immune to Device Variations. In

International Joint Conference on Neural Networks (IJCNN), pages 1775–1781, 2011.

[175] Yanqing Chen. Mechanisms of Winner-Take-All and Group Selection in Neuronal Spiking Networks. Frontiers in Computational Neuroscience (FCN), 11, 2017.

[176] Paul Ferré, Franck Mamalet, and Simon J. Thorpe. Unsupervised Feature Learning With Winner-Takes-All Based STDP. Frontiers in Computational Neuroscience (FCN), 12:24, 2018.

[177] Peter Udo Diehl and Matthew Cook. Unsupervised Learning of Digit Recognition using Spike-timing-dependent Plasticity. Frontiers in Computational Neuroscience (CN), 9, 2015.

[178] Jixuan Wang, Bin Deng, Tianshi Gao, Jiang Wang, and Hong Tan. Spike-frequency Adaptation Inhibits the Pairwise Spike Correlation. Frontiers in Neuroscience (FN), 17, 2023.

[179] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling Deep Spiking Neural Networks with Hybrid Conversion and Spike Timing Dependent Backpropagation. CoRR, abs/2005.01807, 2020.

[180] Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian Schönbach, editors. Encyclopedia of Bioinformatics and Computational Biology, volume 1. Elsevier, 2019.

[181] Yan Meng, Yaochu Jin, Jun Yin, and Matthew Conforth. Human Activity Detection using Spiking Neural Networks Regulated by a Gene Regulatory Network. In International Joint Conference on Neural Networks (IJCNN), pages 1–6, 2010.

[182] Eugene M. Izhikevich and Niraj S. Desai. Relating stdp to bcm. Neural Computation, 15(7):1511–1523, 2003.

[183] Yan Meng, Yaochu Jin, and Jun Yin. Modeling Activity-Dependent Plasticity in BCM Spiking Neural Networks With Application to Human Behavior Recognition. IEEE Transactions on Neural Networks (NN), 22(12):1952–1966, 2011.

[184] Biswadeep Chakraborty and Saibal Mukhopadhyay. Heterogeneous Recurrent Spiking Neural Network for Spatio-temporal Classification. Frontiers in Neuroscience (FN), 17, 2023.

[185] Jingren Zhang, Jingjing Wang, Xie Di, and Shiliang Pu. High-Accuracy and Energy-Efficient Action Recognition with Deep Spiking Neural Network. In Neural Information Processing (ICONIP), pages 279–292, 2022.

[186] Yangfan Hu, Huajin Tang, and Gang Pan. Spiking Deep Residual Networks. IEEE Transactions on Neural Networks and Learning Systems (NNLS), 34(8):5200–5205, 2023.

[187] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep Residual Learning in Spiking Neural Networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, volume 34, pages 21056–21069, 2021.

[188] Ali Samadzadeh, Fatemeh Sadat Tabatabaei Far, Ali Javadi, Ahmad Nickabadi, and Morteza Haghir Chehreghani. Convolutional Spiking Neural Networks for Spatio-Temporal Feature Extraction. Neural Processing Letters (NPL), 2023.

[189] S. Berlin and Mala John. Spiking Neural Network based on Joint Entropy of Optical Flow Features for Human Action Recognition. The Visual Computer (VC), 38:1–15, 2022.

[190] Yan Zhou, Yaochu Jin, and Jinliang Ding. Surrogate-Assisted Evolutionary Search of Spiking Neural Architectures in Liquid State Machines. Neurocomputing (NC), 406:12–23, 2020.

[191] Wei Wang, Siyuan Hao, Yunchao Wei, Shengtao Xiao, Jiashi Feng, and Nicu Sebe. Temporal Spiking Recurrent Neural Network for Action Recognition. IEEE Access, 7:117165–117175, 2019.

[192] Yannan Xing, Gaetano Di Caterina, and John Soraghan. A New Spiking Convolutional Recurrent Neural Network (SCRNN) With Applications to Event-Based Hand Gesture Recognition. Frontiers in Neuroscience (FN), 14, 2020.

[193] Esteban Anides, Luis Garcia, Giovanny Sanchez, Juan-Gerardo Avalos, Marco Abarca, Thania Frias, Eduardo Vazquez, Emmanuel Juarez, Carlos Trejo, and

Derlis Hernandez. A Biologically Inspired Spiking Neural P System in Selective Visual Attention for Efficient Feature Extraction from Human Motion. Frontiers in Robotics and AI (FRAI), 9, 2022.

[194] Ungsoo Samuel Kim, Omar A. Mahroo, John D. Mollon, and Patrick Yu-Wai-Man. Retinal Ganglion Cells—Diversity of Cell Types and Clinical Relevance. Frontiers in Neurology (FN), 12, 2021.

[195] T.A. Münch. Information Processing: Ganglion Cells. In Darlene A. Dartt, editor, Encyclopedia of the Eye (EE), pages 355–362, 2010.

[196] Zahra Babaiee, Ramin M. Hasani, Mathias Lechner, Daniela Rus, and Radu Grosu. On-Off Center-Surround Receptive Fields for Accurate and Robust Image Classification. In International Conference on Machine Learning (ICML), pages 1–21, 2021.

[197] Davide L Manna, Alex Vicente-Sola, Paul Kirkland, Trevor J Bihl, and Gaetano Di Caterina. Simple and Complex Spiking Neurons: Perspectives and Analysis in a Simple STDP Scenario. Neuromorphic Computing and Engineering, 2(4), 2022.

[198] Bodo Rueckauer and Shih-Chii Liu. Conversion of Analog to Spiking Neural Networks using Sparse Temporal Coding. In 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pages 1–5, 2018.

[199] Simon J. Thorpe, Rudy Guyonneau, Nicolas Guilbaud, Jong-Mo Allegraud, and Rufin VanRullen. SpikeNet: Real-time Visual Processing With One Spike per Neuron. Neurocomputing (NC), 58-60:857–864, 2004. Computational Neuroscience: Trends in Research 2004.

[200] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based Strategies for Rapid Processing. Neural Networks (NN), 14(6-7):715–725, 2001.

[201] John Canny. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 8:679 – 698, 1986.

[202] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4510–4520, 2018.

[203] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. Searching for MobileNetV3. In IEEE International Conference on Computer Vision (ICCV), pages 1314–1324, 2019.

[204] Jian Ren, Xiaohui Shen, Zhe Lin, and Radomír Měch. Best Frame Selection in a Short Video. In IEEE Winter Conference on Applications of Computer Vision (WACV), pages 3201–3210, 2020.

[205] Ali Diba, Ali Mohammad Pazandeh, and Luc Van Gool. Efficient Two-Stream Motion and Appearance 3D CNNs for Video Classification. ArXiv, arXiv:1608.08851 [cs.CV], 2016.

[206] Yuxuan Zhao, Ka Man, Jeremy Smith, Kamran Siddique, and Sheng-Uei Guan. Improved Two-stream Model for Human Action Recognition. EURASIP Journal on Image and Video Processing (JIVP), 2020, 2020.

[207] Joshua A. Solomon, Charles Chubb, Adrian John, and Michael Morgan. Stimulus Contrast and the Reichardt Detector. Vision Research (VR), 45(16):2109–2117, 2005.

[208] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating Learnable Membrane Time Constant To Enhance Learning of Spiking Neural Networks. In International Conference on Computer Vision (ICCV), 2021.