A robust modular spiking neural networks training methodology for time-series datasets

With a focus on gesture control

Martijn van Wezel

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

[1]

A robust modular spiking neural networks training methodology for time-series datasets

With a focus on gesture control

by

Martijn van Wezel

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Engineering

at the Delft University of Technology,
to be defended publicly on Friday October 30, 2020 at 10:00 PM.

Student number:	4342917	
Project duration:	February 02, 2020 – October 30, 2020	
Thesis committee:	Dr.ir. T.G.R.M. van Leuken,	TU Delft, Chairman
	Dr.ir. A. Zjajo,	TU Delft, supervisor
	Dr.ir Z. Al-Ars,	TU Delft,
	Dr.ir. S. Kumar,	TU Delft,
	Ir. D. Maksimiuk,	TU Delft,

TU Delft Delft University of Technology

Abstract

Neurons in Spiking Neural Networks (SNNs) communicate through spikes, similarly that neurons in the brain communicate, thus mimicking the brain. The working of SNNs is temporally based, as the spikes are time-dependent. SNNs have the benefit to perform continual classification, and are inherently more low-power than other Artificial Neural Networks (ANNs). Both the SNNs and other ANNs need to be trained to perform specific tasks. There are several types of methodologies to train SNNs, but there is yet no silver bullet.

Backpropagation algorithms can train other ANNs, but SNNs cannot be trained using this algorithm since the spikes are not differentiable. Methods like Spike Timing Dependent Plasticity (STDP) or Liquid State Machine (LSM) have their limits. Where the complexity of SNNs depends on the dataset, the number of neurons, and other factors.

There were currently no known SNN implementations for the given gesture, achieving near state-of-the-art results. The problem with the dataset is that usually no gesture is performed in front of the. The datasets contain noise, and some data samples belong to two or more classes simultaneously.

The objective of this work is to develop an architecture and training methodology, that allows the classification of the dataset using SNNs. This work presents a novel, architectural training methodology **Suino**, which addresses the above problems.

The architecture consists of two components: the first is the spatial classifier, and the second component is the temporal classifier. The frames from the dataset are filtered in the first stage, i.e. the spatial classifier. The output of the spatial classifier is the input for the temporal classifier, which deals with the temporal properties of the data. Suino does not provide false positives, that is the neurons do not spike on the input dataset if the input dataset does not belong to any of the trained classes; hence the method is robust.

The training method is built around these components, existing of different classical training methods: backpropagation, clustering, or any other consisting of fixed threshold method for auto label correction. The second stage consists of a temporal classification method, trained using the Tempotron learning rule.

The time-series dataset of gestures validated Suino. On the test set, the baseline method had an accuracy of 97.0% with 35K parameters, while the presented method had an accuracy of 90.87% with 21K parameters. Hence, the method is more robust against false detections and continuously performs classification.

Preface

Thank you to my supervisors, fiancée, family, friends.

Innatera for supporting me with my thesis and teaching me to be a researcher. Especially Amir, Sumeet, Rene, and Davide for developing this thesis and helping me solving questions. I hope they will continue this research!

My girlfriend, fiancée, best-friend, Brenda, for supporting me during my studies and ensuring a life-work balance. Thank you Brenda for your motivation and encouragement to be my best.

My parents for supporting me mentally and financially to be their when I needed them.

My colleges at Innatera, professors, fellow students, and employers of CAS for making me feel welcome.

To all my friends how gave constructive feedback on my writing, I think it has improved; still not perfect, but to practice is the key to success!

That super unneeded boring COVID-19 for making my thesis harder, to not get distracted by my hobbies: RC-airplanes, drones, gaming, other personal projects, and my company Muino.

I had my brightest ideas and solutions in the shower!

Martijn van Wezel
Delft, October 2020

Contents

List of Figures

List of Tables

1

Introduction

Are we on the edge of achieving biologically, plausible, artificial intelligence? Can we train artificial intelligence in such a way that it can think and learn for itself? In the past decade, much research [7] was focused on Artificial Neural Networks (ANNs); inspired by biological networks, imitating the brain. Figure 1.1 shows the connections between neurons in the brain. Several types of ANNs classes exist. Conventional neural networks use numerical variables in the communication between neurons. In spiking neural networks, the neurons communicate through spikes. A spike is a small voltage peak in time where the arrival of the spike carries information.

Even though SNNs resemble the brain more than conventional neural networks, none of the existing SNNs can yet be compared to the processing mechanism of the human brain [8, 9]. The brain structure has evolved. When inspecting the brain, one sees a big spaghetti of connections between the neurons. Upon closer look, a highly sophisticated structure will be visible [2].

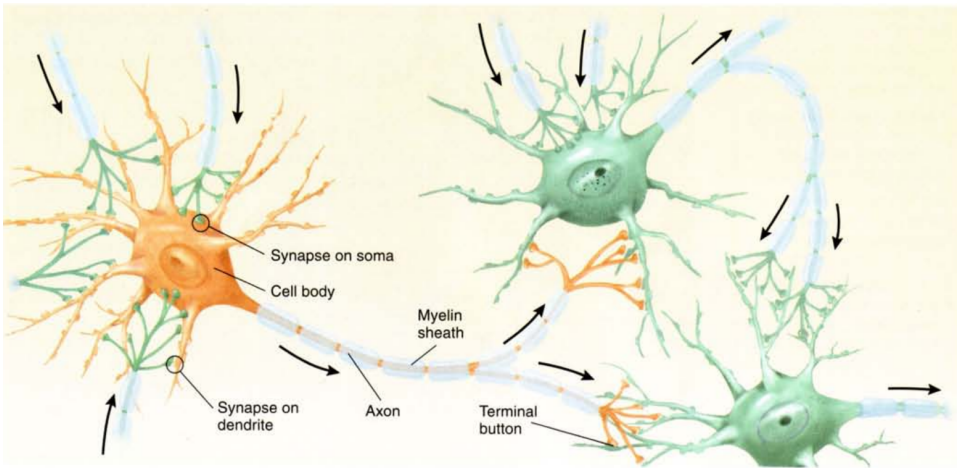

Figure 1.1: The entrance of a neuron (cell body) is a dendrite, connecting synapses of other neurons. The output of the neuron is travelling through the axon to synapses connected to other neurons [2].

1.1. Motivation

The goal is to develop, a SNN hardware implementation, using radar system that detect hand-gestures. Many other use-cases, such as radar, sonar, and automotive platforms [10], would benefit from such a implementation. The automotive world uses, computationally efficient, machine learning models, which have continuous data-stream as inputs. Moreover, automotive applications have a strict power budget and require a great deal of robustness.

SNNs have inherent more low-power consumption when implemented in neuromorphic architectures [11, 12], and can continuously classification, what is ideal for edge-computing. Conventional neural networks can train with backpropagation [13]. However, this is not practical in SNNs due to the fact that the approximated Dirac spikes are not differentiable. Some studies [14, 15], have classified temporal datasets using conventional neural networks; a recurrent neural network. These networks are complex to convert to the SNN. Methods, such as Spike Timing Dependent Plasticity (STDP) [16], or architectures such as Liquid State Machine (LSM) [17], have limits. STDP suffers from the effect of the ping-pong effect [18], where high frequencies affecting the performance of STDP becomes zero were it should be potentiating (i.e. the aliasing effect). The architecture of LSM has positive feedback loops, as the complexity increases, the network starts with a few spikes, but ends in a burst of spikes that is hard to control.

1.2. Thesis Goals

There exists no silver bullet for training SNNs, which suggests a new training method be developed. This thesis explores a subset of ANNs, which a SNN architecture contain. The researcher also explores different type(s) of architecture(s) that could be mapped to an SNN and gives an evaluation of different feature representations of the radar data. The study compares methods with the state-of-the-art in terms of complexity and performance.

The dataset is a radar dataset of hand gestures. Each sample is a video which the SNN architecture needs to classify . The SNN architecture runs on an analog chip that constrains the number of neurons. The model should not exceed more than hundreds of neurons.

1.3. Thesis contributions

This thesis made the following contributions:

- an architectural concept that is suitable for energy-efficient implementation of neuromorphic processors
- a devising an adaptable feature extraction method that spans multitude of classes or labels
- an implementation of a SNN compliant network can classify continuous input data stream
- a training method consisting of two phases: Firstly, filtering the prediction of the labelled dataset and generating the unknown class, and secondly refining and retraining the filtering; subsequently decreasing the network size

1.4. Outline

This thesis briefly introduces SNNs and radar hand gesture datasets. The study will use a baseline to validate the performance of and comparison to the presented architecture. Next, the research reviews different architectures that could be converted to SNNs. The author separates the presented architecture into two different components, with in-depth explanations. Consequently, the author explores and describes the training methodology. The outline of the thesis is as follows:

- **Chapter 2**: Introduction to SNNs, datasets, and prior work.
- **Chapter 3**: Explains the presented architectural design, with a focus on the spatial and the temporal parts.

- **Chapter 4**: Describes the training methodology used to train the architecture.
- **Chapter 5**: Presents the results and analysis of the architecture.
- **Chapter 6**: Concludes the study and recommends future research.

2

Background

2.1. What is Learning?

Learning is a refers to a process where the knowledge of the neurons comes from earlier experience. It allows the brain to draw on previous experiences to learn new processes. However, the experience itself is not stored, but it changes the nervous system in the human brain. What alters the neural circuits; thinking, perceiving, and planning [2].
Why is it important to understand how humans learn? The answer gives an insight into training the learning models. The human brain does not store the information, but physically changes the nervous system: altering the network architecture by deciding which neuron is critical on this specific input.
The human brain has many different kinds of learning strategies, where perceptual-learning recognises the stimuli it perceived earlier. Relational-learning, for example, can determine the type of room, using different objects.
Active learning means adjusting the neural network parameters and not only remembering the explicit given data.

2.2. Artificial Intelligence

Artificial Neural Networks (ANN), also called 'Artificial Intelligence' is the biological inspired neural networks. This thesis distinguishes between the generations of ANNs. Perceptrons make up the first generation of ANNs. The second generation extends the first by adding a pointwise nonlinearity (i.e. activation function) after the perceptron. Finally, the third generation is the spiking neural network, a spike-based architecture instead of a numerically-valued architecture as with previous generations.
Machine learning is the study of algorithms to improve on earlier experience. Deep learning is part of machine learning, based on representation-learning. This learning is feature-based. It understands the network and tunes the features.

2.3. Spiking Neural Networks

Spiking Neural Networks (SNNs) are network-based models that mimic the brain functionalities, where the spikes trigger communication. In other ANNs, the networks use numerical values throughout the whole network.
Squids compared to the larger neuron cells of the human brain, in this experiment [3] they measured the voltage between two neurons, and they notice the spiking behaviour as shown in figure 2.1. Those voltage spikes were for a brief period, and it increased the membrane potential in the neurons. The membrane potential is the voltage in the cell; when the cell exceeds the excitement threshold the neuron will spike.

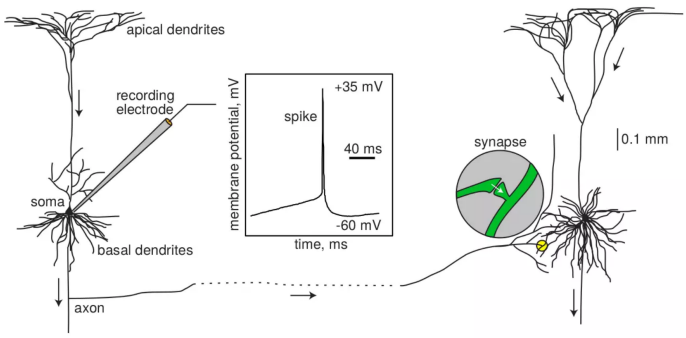

Figure 2.1: The experiment where Hodgkin measured using a recording electrode the spikes between neurons [3].

One of the main benefits of a SNN would be power consumption. High-end GPUs like the NVIDIA 2080Ti [19] draw hundred watts of power, making it less efficient for the radar classification. SNNs could achieve microwatts of power using neuromorphic computing [20]. The primary benefit that SNNs have compared to the same ANN is that the time dependency and continuale classification [21], which is beneficial in a time-series dataset.
The one major issue with SNNs compared with other ANNs is spike-based communication. These Dirac spikes are not differentiable, hence backpropagation [20, 22] will not work. In ANNs the backpropagation is an essential element for training these networks. Therefore, training a SNN will not be straightforward.

Leaky-Integrate-and-Fire Neuron The Leaky-Integrate-and-Fire (LIF) neuron is one of the many types of neuron models for the SNNs, which is computational efficient versus a biophysical trade-off. This model [4] is one of the most efficient computational neuron models [21], but the biological plausibility less compared with more complex models, such as Hodgkin-Huxley [21]. The LIF is not only computational efficient, but also the analog hardware model is less complex to implement [23].

The LIF neuron has the leaky component that makes the neuron important. Each biological synapse connects to the dendrites of another neuron. The signal goes first through the synapse were the importance (weight) is determined. Figure 2.2 shows that from neuron J a spike enters the 'synapse'. The spike will enable a current injection all the inputs currents together is $I(t)$ what is the input to the soma (cell body).

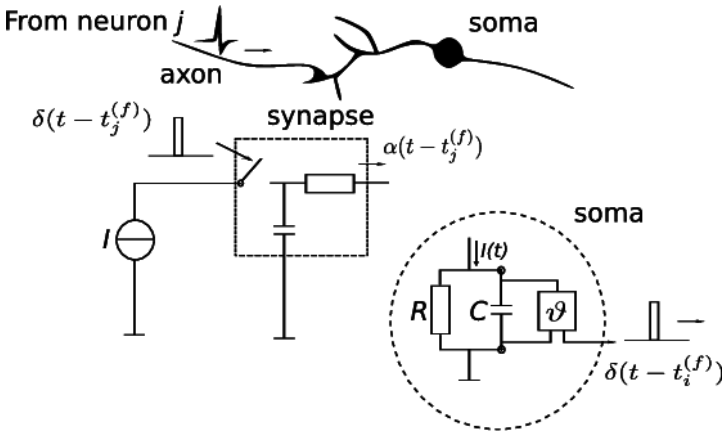

Figure 2.2: In the LIF neuron the synaptic is the receiver of the signal and the soma is the central part of the neuron [4].

2.3.1. Spike encoding

The SNN uses numerically value inputs, which need to be converted to spike-based patterns. In the next section, different approaches are discussed how to generate these spikes for the SNNs. There exist many types of spiking encoding styles. However, selecting the right one also depends on training and architecture. Rate encoding convert an ANN architecture to a spiking model [24]. For temporal encoding, spike encoding algorithms are used [16, 21, 24, 25].

Rate encoding Rate encoding is used to convert a range of values into a range of frequencies spikes [24]. When the input values are increases, the spike frequency increase with it. Figure 2.3 shows the rate encoding of a layer neurons in time. Rate encoding is usually used because of the simplicity of the conversion. However, this has its limitations. If the input values are near the maximum numerical value, the frequency will also be near the max spike frequency. In other words, many spikes are generated if the numerical value is high. The methods power-efficiency decrease, slowing down the network [12].
Rate encoding produces spike trains, but maintaining a single variable is expensive [11]. It can also be likely that the timing of the individual spike trains will convey information penalising the performance of the SNN [24].

Spike encoding algorithms The spike encoding algorithms are encoding methods that convert signals into spike sequences, based on an algorithm. There are many different encoding strategies. Threshold-based encoding is an algorithm that generates spikes when the thresholds of neurons have exceeded. Rank-Order coding and Population-Rank Order Coding assume that the first spike generated hold [24] the most important information. Opposing neurons will spike in time based on their rank of information. Many others exist, but spike encoding algorithms are more complex to implement [12]. However, these algorithms can be 5.68 times faster in making decisions consuming 15.12 times less power. Figure 2.4 shows the temporal encoding. Compared to figure 2.3, it is clear that temporal encoding reduced the number of spikes, so each spike contains more information.

2.3.2. Learning in SNN model

Learning SNN is challenging part and remains an open research question [16, 23]. There is still no general rule for learning a SNN. The transfer function is usually non-differentiable [23]. Backpropagation is the primary method for many ANN models that update the weights using the gradient descent method. The backwards-pass is done using a differential equation that calculates the gradient. However, doing this backwards-pass the transfer function in a SNN is typically not differentiable because of the spike [23].
This section describes the state-of-the-art training methods in SNN architectures. As explained in the previous section, there are different ways to encode the spike. The selection depends on the type of learning strategies used.

Figure 2.3: Rate based encoding in time where each horizontal axis presence a neuron.

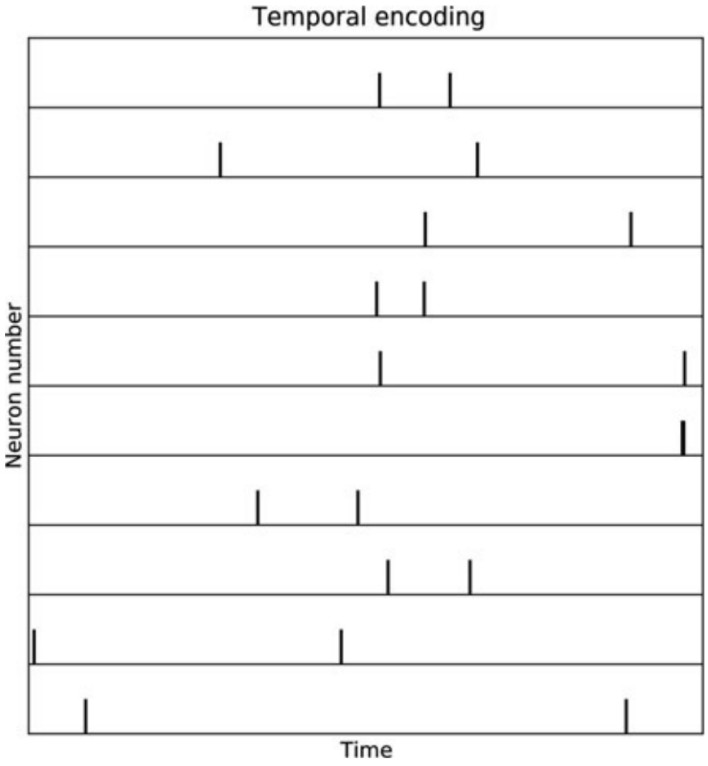

Figure 2.4: Each row represents the spike rate of a neuron using temporal encoding. The presented information is less dense than figure 2.3, but contains the same information.

Unsupervised Learning in SNNs

USing micro-electrodes in rabbits, the study detected a mechanism for learning [26]. This method called spike-timing-dependent plasticity (STDP) is bio-inspired learning (i.e. inspired by this mechanism). There are many variants of STDP implementations [27]. The Tripple-STDP [28] uses three spikes to conclude if the weight of that neuron should be increased or decreased. The implementations are unsupervised and, work in one layer, but to the training is a challenge [18]. The reason for the performance problem is the limiting ping-pong effect. Training using STDP is compared with backpropagation time-consuming.

Supervised learning in SNNs

Supervised learning is a training methodology were the input is mapped to the output. Backpropagation opened many doors for different applications. It had a performance boost, and training became easier to gain higher accuracy. Some researchers combined SNNs with the backpropagation [17, 20, 23, 29–31]. The list is continuously growing, where others combined STDP with backpropagation are designed [30]. There exist other methods such as SpikeTemp [32], E-prop [33], Chronotron [16, 34], to name a few. This thesis discuss the SpikeProp, and Tempotron more in-depth.

SpikeProp Similar to the backpropagation algorithm, the SpikeProp determines to a given input pattern the desired firing times [30]. It is using the least mean squares error for determining the difference between training outputs times and desired output times. The research shows that it overcomes the discontinuities introduced by the thresholding. It would also solve complex non-linearity classifications using the faster temporal encoding and comparable to rate encoded architects. In the paper [24], they explain possible assumptions: each neuron in the network can only fire once in each processing step, ignoring the time-course of the neuron's membrane potential after the firing.

Tempotron The Tempotron learning rule is a supervised synaptic learning rule to tune the synaptic efficacies with the following amount equation 2.1.

$$\triangle\omega_i = \lambda \sum_{t_i<t_{max}} K(t_{max} - t_i) \tag{2.1}$$

Where λ is the maximal synaptic update for the spikes, and the $K(..)$ is the spike itself. It is important to note that the learning rule (λ) is essential to tune right. The t_{max} is the postsynaptic potential voltage, where it the max is reach. This implements a gradient-descent, which minimizes the cost function. The cost function is $V(t_{max}) - V_{thr}$, where V_{thr} is the threshold voltage [35]. The limitation of Tempotron is that it only allows having one layer, because of the update rule.

2.4. ANN to SNN

Spiking neural networks are biologically plausible models but struggle with achieving accuracy. In contrast, deep-learning has been the state-of-art research methodology attaining the highest performance. It is due to state-of-the-art results the representation-learning model quickly achieves. There is extensive research converting deep-learning methods to the SNN field [36–38]. However, most are conversions of a model architecture to a digital implementation of a SNN.
A limitation of this method is ignoring the benefit of the SNNs of temporal classification. Most recent SNN studies focused on backpropagation [17]. Porting it to an ANN but neglecting the timing is essential for low-latency handling of temporal problems. However, these methods solve the continuous classification and use less power. Many of the experiments for the SNNs backpropagation implementations use digital SNN hardware accelerations [39].
The convolutional neural networks (CNNs) [40] are mostly used for image processing, for example,if it is a cat or dog, or which rowing boat team is in an image [41]. The CNNs are resemble the opposite of recurrent neural network (RNN) [17, 42]]. Speech, language, or signal processing applications mostly use the recurrent neural networks (RNNs). The RNNs are used for a temporal sequence, such as the SNNs. The LSTM looks most closely at the SNNs because of the leaky parameter. The research focussed on solving the gap between LSTM and SNNs [14, 17, 39, 43–47]. The sSNU method of [17] uses state-of-the-art LSTM implementation.

2.5. Prior work

In this section discusses, the related work from scientific publications and explores the basics of the radar, hand gesture datasets .

2.5.1. The Range-Doppler radar

The generation of the Range-Doppler are developed from a signal, shown in figure 2.5. Understanding how the Doppler frames are generated explains the number of processing steps taken and the issues these may cause. Using multiple antennas could also improve the accuracy of the classification, as shown in the paper [48]. Here the study use the radar connected to a short-time discrete Fourier transform (STDFT) to obtain the angle information from multiple antennas.

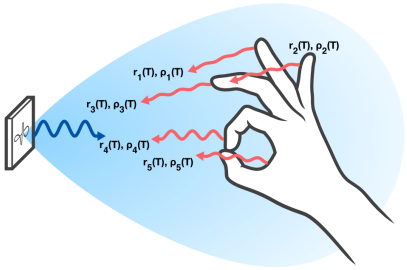

Figure 2.5: The radar transmits a signal to the hand; the movement of the hand creates a doppler effect. In this effect of reflections from multiple dynamic scattering centres change the frequencies of the waves [5].

The radar sensor sends a RF carrying frequency [5] to the environment. The radial distance $r_i(T)$ variates in time T and the complex reflective parameter $p_i(T)$ reflects with a frequency variation, because of the Doppler effect.

Two Fast-Fourier Transforms (FFTs), showed in figure 2.6 generates the Range-Doppler images [5, 6]. As figure 2.5 shows, the radars send a Linear Frequency Modulated Continuous Waveform (LFMCW), to the object and receive it back. Multiple sweeps are collected and placed into a matrix form, where each row represents one sweep, as shown in figure 2.7. After the first FFT, each row contains the range profiles. Finally, the second FFT creates the Doppler in each column. If there is insufficient information in that column from the first FFT, the column would have an evenly distributed valued range. The moving object will have in the scatterings changing of frequencies in the same range, and can therefore be seen as (a) blob(s). Moving a hand will have a scatter of velocities, and in the Range-Doppler shows the hand as a blob.

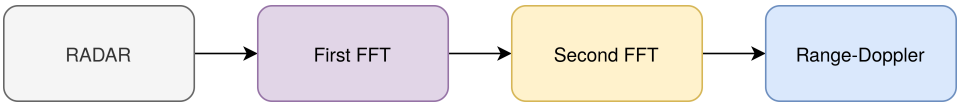

Figure 2.6: The raw data that is entered, based on the two FFTs generates each Range-Doppler image.

Range-Doppler images are beneficial when there is a need for having velocities and range data. An example of such an application is the exact speed or direction of the object. For finding the angle of an object, researchers need multiple antennas. Tuning the parameters of the radar will change the focus of the radar.

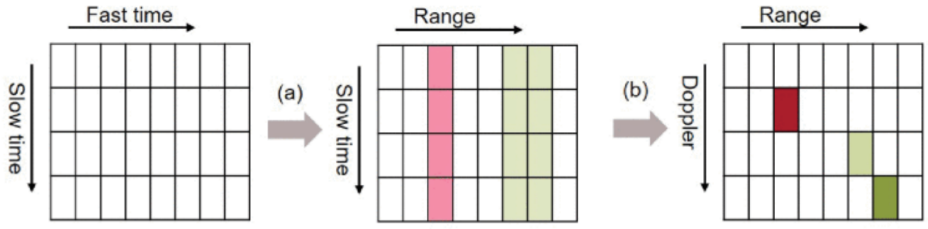

Figure 2.7: Range-Doppler generation by a) The first 1D FFT along the fast time, b) The second FFT along the slow time [6].

2.5.2. Google Soli

Google Pixel 4 smartphone is using the radar for gesture control. They are using a $\sim 60GHz$ sub-millimetre radar placed next to the front camera. The earlier gesture-radar research [5, 49] found an improved algorithm. The published papers discussed the use of RNN architectures or 3D CNN [6] models.
These models use many parameters. However, compared with the hardware model, these will not fit the design requirement of this thesis.
The paper [50] describes the implementation of the exact sensor. The radar used in Pixel 4 uses multiple antennas inside of the chip. This radar can discriminate between multiple targets. They need this information to determine the angle of multiple movements of a hand gesture.

2.5.3. Gesture recognition using LSMs

Liquid State Machine The Liquid State Machine (LSM) would be a biologically plausible [51] architecture. The LSM is essentially an one-layer trained network. Zooming into this structure, it shows many recurrent connections, a reservoir of neurons stimulating other neurons with a recurrent, connected topology [8]. In the network, there are different neurons: input neurons, feedback neurons, and output neurons. This paper [52] highlights the issue with LSMs; the number of spikes explodes throughout the network. However, paper [8], contradicts this arguing that this will not happen with more inputs and a bigger model using their method.

2.5.4. RGB video datasets

There exist a few datasets, such as the Jester [53], and IsoGD [54]. Those are RGB video clips of human hand gestures. This study could use this dataset if the sensing were the same. However, this research uses the dataset of radar hand gestures. The researcher selected this fundamentally different dataset because of RGB values instead of Doppler velocities.
Use this dataset as a baseline will no fair comparison. The features that the network will train cannot be compared with the radar dataset. The radar dataset is a different type of data representation. The radar dataset exists mostly out of blobs [6], were the RGB dataset exist of a picture with a subject's hand.

Transfer learning Transfer learning is a method [55] to use a well-trained model and convert it to a different architecture or learning purpose. When converted, this architecture will be used to train it further. The benefit of this method is to decrease training time, so there is only one fine-tune phase needed. The disadvantage of transfer is the overfitting to particular classes [56] that were pre-trained by the other.

3

Architectural design

This chapter represents the SNN architecture. Section 3.1, describes the baseline and the different design possibilities while Section 3.3.3 discusses the presented architecture.
The architecture is separated into two distinct stages. The first stage is the spatial classifier that finds spatial importance in the design, acting as a filter. The second stage of the network is the temporal classifier that determines the classification using the temporal aspect of the time-series dataset. These stages describe different methods and arguments.
The SNN architecture is an analogue-spiking, neural network with hardware limits: the architecture is developed for hand gesture recognition. The next chapter explains the training method for the presented architecture.

3.1. The baseline for the radar dataset

The researcher chose the state-of-the-art convolution, Long Short-Term Memory Conv_LSTM [14], as the baseline for the gesture time-series dataset. The architecture combines the CNN with the LSTM. Each frame has a CNN detecting features and an LSTM determining the class. The research needs a baseline for two reasons. Firstly, there was insufficient information about the dataset and what the values represent. Secondly, a baseline compares a state-of-the-art design with the presented SNN architecture to understand the performances.

3.2. Architectural requirements

The chosen architecture meets design requirements. The design needs to be small, not exceeding thousands of neurons. Adding an extra digital circuit in the network will not be a significant problem, but this cannot be implemented in the final design. The architecture should consider that the neuron model is pre-defined. Another neuron model, the Izhikevich [21] is not viable. Instead, the SNN neuron model selected should be the leaky-integrate and fire (LIF).

Figure 3.1 explains the basic structure of the overall architecture. In a practical environment, the time a person takes to perform a gesture cannot be determined accurately. Therefore, the architecture should be a flexible design, which could be altered. The Range-Doppler dataset contains hand gestures video recordings, not consistently starting or ending at the same video frame. The architecture needs to allow for empty frames.

3.3. Architectural overview

The author discusses architectures in the following section.

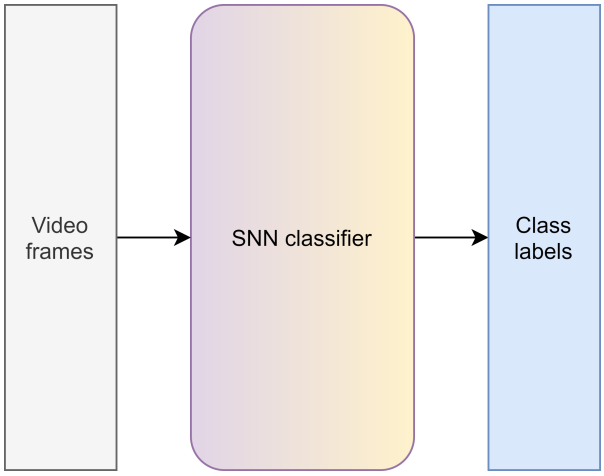

Figure 3.1: The radar is sensing the gestures and will be processed by the SNN that determines the existence and types of hand gestures. If no hand gesture is present, there will be no response. The SNN will spike if a gesture is present. This design record radar data continuously.

3.3.1. Liquid State Machine

An earlier study explored the Liquid State Machine (LSM) design [34]. It found that the biggest obstacle was tuning the parameters, in the order of 20-dimensional space. There could be multiple, workable solutions but finding the best solution required a large amount of computation power. However, some parameters could already be determined to decrease the parameter space. The other obstacle for using an LSM is the positive feedback loops. These positive feedback loops will explode the spike on the output. The LSM has recurrent connections between neurons, which are the positive feedback loops. The neurons spikes on earlier signals; due to design, there are no negative spikes. Studies showed that LSMs are easy to tune for small problems using temporal encoding [8, 51, 52]. Another study has shown the stability of inputting nine neurons with temporal encoding, which will not explode with spikes throughout the network [8].

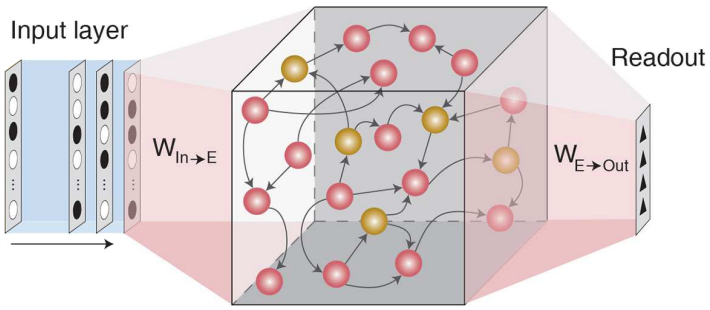

Figure 3.2: The LSM is shown as the grey box where multiple neurons are recurrently connected. The readout of the network relates to the specific set of output neurons.

The LSM has the possibility of tuning the time scale due to the recurrent connections. The LSM was proposed as a brain-like architecture. However, this network is only a small part of the brain seen as a hyper-parameter itself [51]. The sparser the network, the weaker the interaction between neurons; a more dense network would become more chaotic.

3.3.2. Long short-term memory in SNN

The long short-term memory in SNN (LSTM) is a recurrent neural network (RNN), that is specialised in sequential data. Speech, translation of a text, or the gesture dataset are examples of such data. The

LSTM uses the same principle as the leaky unit methods that in time, delayed information or saved it for extended periods.
LSTM architectural use recurrent neurons behind each other. There are studies done in the LSTM to SNN implementation. The paper [17] proposed an LSTM model with SNNs and showed similar accuracy with practical Spiking Neural Unit (SNU). However, exist an implementation, as the results that were shown, were based on a simulation. There is no implementation found, and no substantial evidence that learning such architecture and implementing would work. This can still be something to consider, as [39] showed that such a design had many obstacles to overcome. The main advantage of an LSTM in SNN is the time dependency.

3.3.3. The layered network

The solution would be that the frames come in it continuously. To have a smaller network, as suggested by [57] will provide a filter at the start where the second part is classified. Figure 3.3 gives an overview for the layered network of a radar dataset. The video frames enter the architecture using an encoder and the spatial classifier filters the encoded radar frames one-by-one. The output of the spatial classifier directly connects with the temporal classification. In the temporal classifier, the temporal aspect is analysed to determine the type of gesture. When there is a gesture found, the output will fire from the temporal classifier enabling the decoder can translate the spike to a digital signal.

In a practical application, many of the frames hold no gesture information. The temporal classifier inputs directly connect to the output of the spatial classifier. The temporal classifier will work as a temporal decision buffer making decisions based on the output of the spatial classifier.

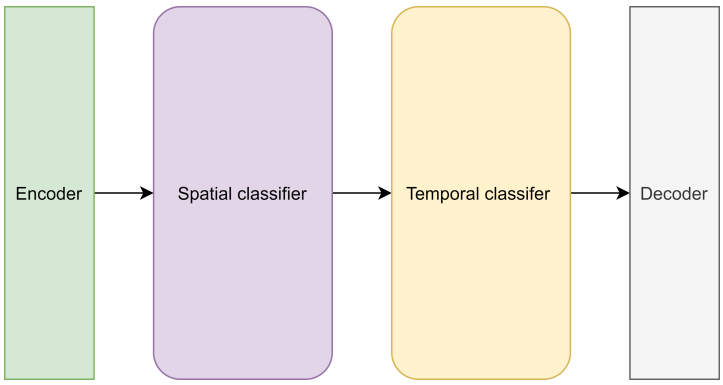

Figure 3.3: Network architecture-layered video frames come in one-by-one to the spatial classifier. Where it classifies each frame to which class it belongs. The temporal classifier decides whether a class is enough stimulated and fires on that gesture.

3.4. Spatial classification

The spatial classifier is the input filter of the network and tries to categorise the input frames that come in. The frames will enter individually, and the output will distinguish between the class and the frame. In the hand gesture dataset, many frames are empty or belong to multiple classes, which require further research to gain insight into its ability to perform. The spatial classifier should be robust against empty frames or those that are similar between classes. This makes the spatial classifier perform well in continues data streams, but it does not mean that this is the most critical function. The primary aim of the spatial classifier will be pre-filtering the data so that the temporal classifier can act as the relational time-depend decision-maker, similar to the relational-learning principle. The following section discusses the different designs of possibilities. In future datasets, the learned information will still be valuable. No one design fits all.

3.4.1. Self-Organising map

The Self-Organising Map (SOM) is a brain-like structure were the neighbour neurons are connected. Figure 3.4 shows the SOM structure. The SOM structures benefit is that it has the perfect topology-preserving architecture. The method is unsupervised, but there exist supervised versions that help to accelerate the learning [58]. The architecture has a tree-type structure for each class, with a likely strong response between a specific neuron and class. The further away a neuron is from a sample, the less correlation that neuron had with that sample. The paper on the SOM-layer [59] suggests that it works well with data that contains blobs. In the given time-series dataset, the information is given in blobs, but some classes have minor differences. The SOM layer SNN implementation will be the same type of architecture as the ANN.

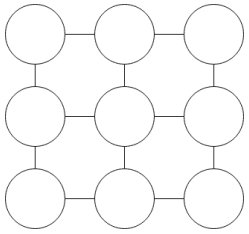

Figure 3.4: The self-organizing map holds neurons that are connected with their neighbours. The further away from a neuron, the less correlation the sample has with that neuron.

The architecture of the SOM layer can exist in many forms: circular, perfectly preserved, or 3D cube. The architecture could be tuned to a specific dataset. Before the training starts, the weights are set randomly with a standard feed, making it reproducible on any computer for that specific library version.
Each sample will be compared to the network for finding the closest neurons. Finding the closest neuron to the sample is done using the Euclidean distance between the weights and the sample. The neuron weights that is the closest to the sample will be updated. The weight update formula uses a learning-rate ($\eta_\eta(t)$) for stability.

$$Wj^T \leftarrow Wj^T + \eta_\eta(t) * h(\eta_b, t) * (X - Wj^T) \tag{3.1}$$

In this approach, the neuron that is more closely related to that specific data-point will respond. There are no biases; thus no updates are done. Equation 3.1 presents the weight-update formula. The $h(\eta_b, t)$ is the neighbourhood update function. Essential this means that the neurons around the best match neuron $h(\eta_b, t)$ will be 1. When that neuron in the network is not the direct neighbour, the function returns to 0.

3.4.2. Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is a proven architecture in the field of artificial intelligence. The MLP dates from 1986 [13] when there were limited computing power and resources for learning deep networks consisting of millions of neurons [7]. The benefit of the MLP compared with CNNs is the more natural conversion from an MLP to SNN. The disadvantage of this conversion is that the encoding variant should be rate encoded and the loss converted to SNN. The two main advantages of using an MLP for the conversion to SNN has many hidden layers. Firstly, it makes the architecture more noise-robust [58] and solves the XOR problem. Secondly, the network has bias implemented with a RELU activation function. There exist MLPs without biases in the neuron, but these networks have shown problems classifying the radar dataset.

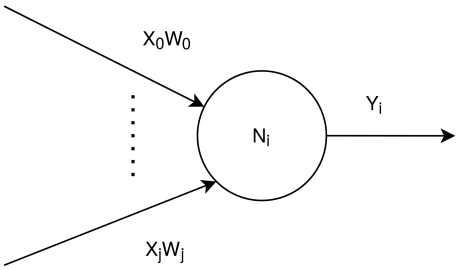

Figure 3.5: An neuron (N_i) in a MLP where X is the input, W the weights, and Y is the output of the neuron what is for another neuron behind its X.

Figure 3.5 shows the neuron in the MLP network. The biases (β) is in the neuron itself (equation 3.2). Where the activation function ($act(.)$) is the RELU activation function. The reason is that is more closer to the activation function in the SNNs. In the analog SNN neurons negative voltages are not possible; the activation function should be a positive valued [46].

$$Y = act(\beta_i + \sum_{k=0}^{j} \omega_{i,k} x_k) \tag{3.2}$$

The training is done using the Python Scikit-learn library [60]. The reason for being that many learning functions or extra functions for testing the performance were already implemented. These weights and biases are saved and imported to the simulator using the function $ANN2SNN.m$.

3.4.3. Determining the value n-outputs

For the dataset, the only labels that exist are the labels for the full video. In the future, many more gestures can be added to the dataset. Another approach of a dataset could exist only of gestures that can be seen as a vector. Instead of a full gesture, the gesture is divided into parts like waving. Waving exist of two elements left and right, separating those into two different spatial outputs, where the first output is the one going left movement and the other to the right.
There are six gestures, with each having multiple vector movements. Labelling the dataset by hand is not an option, so an evaluation of different feature representation has been done to determine n-outputs of the spatial classifier.

Histogram A histogram can detect the blobs. There are two histograms for the row and column. The histogram is simply the sum of each row and column. The highest peaks of both histograms mean that the blob is being placed. Calculating the maximum value of the two histograms gives the location of the blob. However, there can be multiple blobs. A threshold can determine all the peaks above this value if the blob is in the middle points. One disadvantage is that if it is a digital approach, the blob itself contains essential information about the type of gesture. The background of such a gesture (e.g. moving the body) could contain critical information in the classification. Another issue with a

histogram is that most of the blobs are not perfect circles. Only using the histograms as an output for the network would mean an n-by-n output with a value range for each output.

Scaling and orientation The field of computer vision, many studies were done to discover features, filters, and stereo images that generate 3D point-clouds. The results identified several ways to find feature points in an image needed to stitch multiple images together. There are many approaches that all work differently, even better, such as SIFT [61].

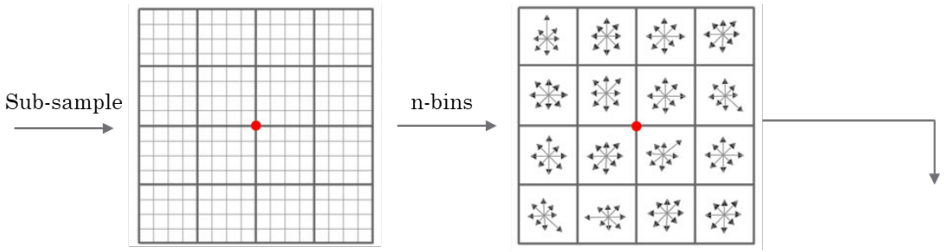

Figure 3.6: The frames are sub-sampled where each sub-samples is seperated into n-bins.

Finding feature points in a frame and generating a label should be as unique, yet universal. When two images shift in orientation or magnitude, the feature point should still be the same.
The following section explains how the SIFT feature detector is used on the radar dataset. The approach of the difference of Gaussian is not used. It was used to find the feature-area point, but this dataset had already contained interest points. Figure In figure 3.6 gives two steps:

- The frames are filtered using a first-order Gaussian filter with a kernel size of three in both x and y direction and divided by two.
- The absolute is taken from the frames. Then the frames are normalised between the minimum value and the maximum value.
- Next, the frames are sub-sampled. In the paper of SIFT, this will be around the interest-point, where it will be ignored.
- For each interest-point, the magnitude and orientation are calculated.
- Then the orientations are divided into k-bins, where k is a hyper-parameter. If the value eight has been chosen, bin zero would be from 0-45 degrees, bin one from 45-90, till 315-360 degrees.
- There are eight subspaces in the rows and eight in the columns, so 8-by-8 sub-spaces each containing 8 bins, therefore $8 * 8 * 8 = 512$ output vector element. The researcher divided it under k-bins again.

Meansift The Meansift [62] is a clustering algorithm focusing on finding the maximum of the density function. It looks for the maximum in the frame where the blobs are positioned. Meanshift is used for finding the blobs in the frames, where the number of clusters is not being defined. In other clustering methods, the number of required clusters to be found are given in advance, such as K-means [63]. The Meansift algorithm defines the number of several types of blobs.

However, to have a better understanding if the amount of information is sufficient to tell if the clustering value is big enough. The number of frames used is compared with the clustering coefficient. This comparison is obtained from the input frames versus output clusters. Figure 3.7 gives an example of the Meansift implementation.

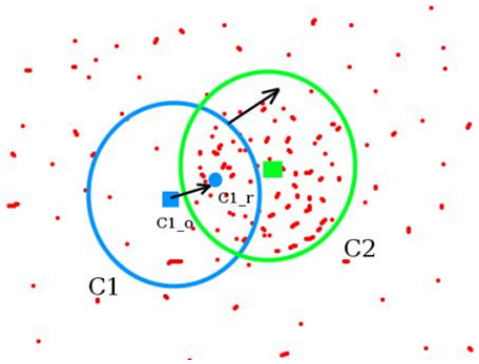

Figure 3.7: The clustering coefficient is the diameter from which a cluster should exist.

3.5. Temporal classification

In the human brain, there are many learning strategies. The spatial classifier is the perceptual-learning strategy for recognising perceived stimulants. The perceptual-learning strategy will learn to identify the objects so that the temporal classifier can classify the relational-learning strategies. The radar frames will be labelled in time so that the multiple gesture frames can be recognised as a gesture. The temporal classifier would be confident if it perceived more of those gestures before.
This section of the chapter discusses the temporal classifier. Theoretically, there are several approaches possible, but this paper explores the following practical approaches. Firstly, the dataset generated from the special classifier is discussed . Secondly, the author explores different methods, such as the counting method, array buffer, and training using Tempotron.

3.5.1. Dataset setup

The output of the spatial classifier will connect to the temporal classifier. Each video is classified in time, and the output of the spatial classifier will spike on each frame. The temporal-dataset consists of an array of gestures labelled by the spatial classifier. Each video label is the original label from the radar dataset. Equation 3.3 shows the dataset. Each row represents the frame classification time-step, and the last row represents the original label for the video. The radar dataset consists of six gestures. However, the output of the spatial classifier has an extra unknown gesture output. In the given equation, it is gesture seven (g7). Each frame label in this temporal-dataset is entering with the same time-stamp 20msec (50Hz).

$$\begin{bmatrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 & \tau_5 & \tau_6 & \dots & \tau_{49} & \tau_{50} & Label \\ g1 & g2 & g1 & g2 & g7 & g1 & \dots & g7 & g7 & g1 \\ g5 & g7 & g3 & g7 & g5 & g5 & \dots & g3 & g5 & g5 \\ & & & & & & \dots & & & \\ g4 & g6 & g7 & g4 & g4 & g5 & \dots & g6 & g6 & g6 \end{bmatrix} \tag{3.3}$$

3.5.2. Counting the frames

To determine the performance of the spatial classifier, the counting method was used for its practical implementation. It will allow insight into the performances and its failures. When starting with a complex implementation, it is more difficult to establish where it will fail. It is not only essential to distinguish cases where it failed, but it can also be used to determine how well the spatial part of the network performed. The counting method and heat-maps are used to tune the spatial classifier.

There are n-frames where it is not known when the actual gesture starts. The temporal-dataset contains m (seven) different gestures, which calles for $m+1$ counters. The extra counter is for keeping track off the number of frames that came in. Each time a new gesture comes in a counter of that gestures are increased and the frame counter. In the radar dataset there are always 50 frames for each gesture; in figure 3.8, the frame counter will count to 50 frames.

The counting method can have more advanced features, for example, for each gesture, a different threshold value. When there are not enough gestures found in the class, the video can be ignored . With the condition that there are enough gestures found, the method will determine the maximum counted gesture, and return the most founded gesture.
The benefits of this implementation are that it gives an uncomplicated overview of each video gesture, and the digital implementation can be implemented in hardware without complex operations. The disadvantage of the counting approach that is it is not possible to determine connections between certain gesture classifications that overlap. If more frames of gesture one is in gesture video two, the counting method cannot find correlations for the video classification. There is also a loss of information with the temporal gesture classification. There are videos where the last frame(s) shows many false detections. The counting method removes the time information knowledge what will result in a false

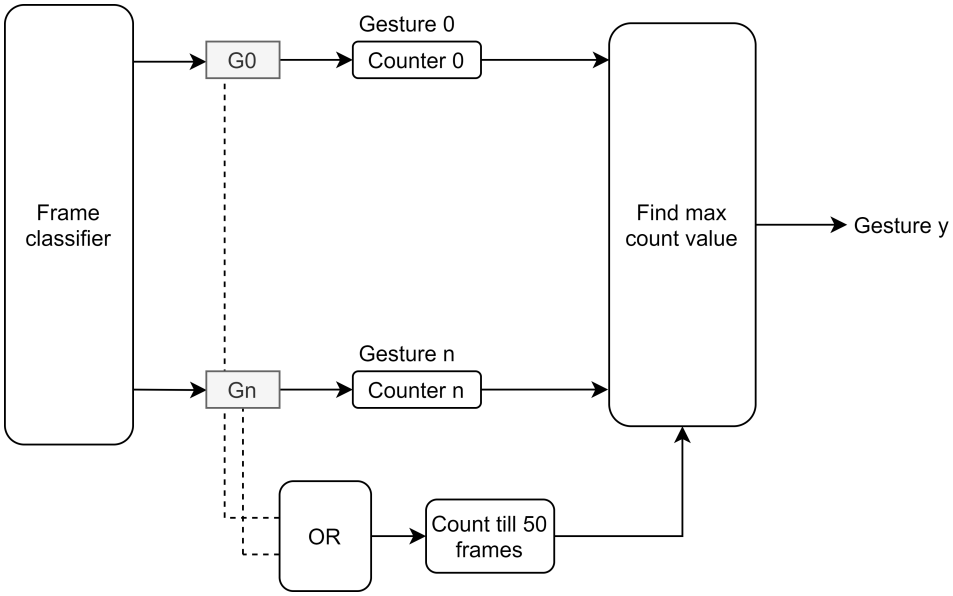

Figure 3.8: Here the counting implementation has been shown without the threshold part. The method shows that it will count till 50 frames and then compare each gesture counter for finding the maximal gesture counter.

detection. The main benefit of SNNs is the in-time classification. With the counting method this benefit will not be used.
In summary, the counting method is useful for determining performance. It is not the best solution because it removes the time information. Nevertheless, the solution with a threshold could be practical for implementation, if the temporal classification is not essential.

3.5.3. Array buffer

The array buffer uses a FIFO buffer architecture for classifying a gesture in time. The input for the temporal classifier will be the FIFO buffer implementation. Each time a new frame is being labelled, it will enter the buffer, and the oldest gesture will leave the buffer. This approach will divide the SNN network into two parts, each part using an encoder and decoder.
This approach is useful when the classification cannot be done using an SNN or counting method. The disadvantages of the buffer method are that a digital buffer would be less energy efficient than an SNN approach. The other issue is the classification: continuously classifying each time a new gesture comes in, and if multiple gestures are performed quickly after each other. It will be not known when a particular gesture is finished, when the choice is for a windowed approach where with each n-frames a classification happens. Hence, this buffer should have a variable width. Therefore the size of the buffer is unknown, and the dataset could get more complicated to detect multiple gestures.
The given array buffer could be designed using a layer of SNNs that will decide when is a gesture being found. A second layer after the SNNs can be classified using a network, such as a decision tree [57].

3.5.4. Learning using the Tempotron learning rule

There exist multiple training methods to train an SNN. The training methodology will be temporal-based encoding because the spatial classifier generates spikes. Those spikes can be used as an input for the temporal classifier. There are $m-1$ inputs from the spatial classifier, where the unknown class is not connected to the temporal classification. The counting methods showed already promising results, but it misses the temporal classification and decisions based on multiple classes. Some classes are closely related to each other. Making decisions based on multiple input gestures for a class will improve the final accuracy. The state-of-the-art LSTM has a leaky factor inside of its neuron; the SNN's LIF neuron. Tempotron learning rule [35] uses the gradient-descent learning approach for training the LIF neurons. In the background, equation 2.1 describes the update method of the weights. The LIF neurons are also in the design of the inhouse neuromorphic processor. Tempotron learning rule is limited by having one-layer architecture. Multiple layers can solve the `XOR` problem and be more robust against noise [58]. Using the Tempotron learning rule, the leaky par, decides from multiple inputs, and determines when the neuron is excited enough.
There are other approaches for learning SNNs, but Tempotron is a straightforward approach to train because it uses less hyper-parameters. The other benefit is it does not matter which type of input spike [64] the Tempotron uses.

4

Training methodology

In the following chapter, the training method **Suino** has been discussed for the presented architecture. The previous chapters have described the spatial and temporal part of the architecture what is shown in figure 4.1. The first part of the network is called the spatial classifier or frame classifier, and use a trained ANN. The second part of the network is called the temporal classifier. The temporal classifier is trained using a SNN training method. This classifier uses temporal encoded spikes as an input to classify the hand gesture in time. However, setting all the weights and biases of the network correctly, we first have to determine, what learning steps are taken and the procedure for achieving state-of-the-art results. In next section **??**, an brief overview is given where the followup sections give a more in-depth analysis.

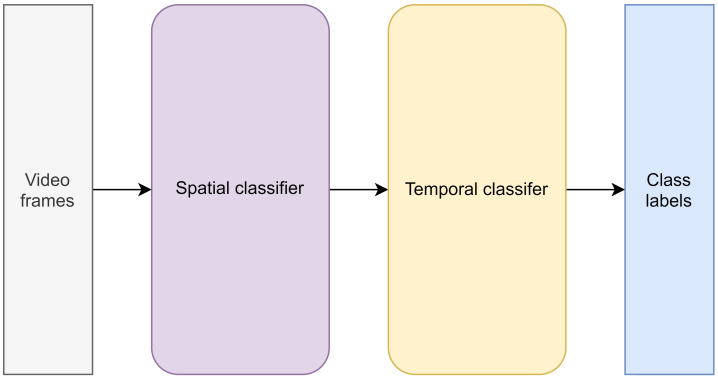

Figure 4.1: For the hand gesture dataset, the video frames enter the spatial classifier one-by-one and the temporal classifier determines if there was a hand gesture being found.

4.1. The rest of the chapter is redacted from the thesis and is explained in the future patent.

5

Results and Analysis

This chapter discusses the experiments and analyses the obtained results. Here, the author also argues the architecture, training methods, and algorithms:

- The author discusses the radar hand gesture dataset and compares the baseline to the presented architecture from this thesis.
- The author shows the spatial and temporal classifier results and the reasoning for specific design steps.
- The researcher analyses the training methodology and its hyper-parameters.

5.1. Radar dataset

The radar that has been used for the detection of hand gestures is a $\sim$60GHz radar sensor, with three channels where each channel represents an antenna. The fourth antenna may not have been given in the dataset. The dataset consists of six different hand gestures, shown in table 5.1. The given dataset did not explain which hand gesture belongs to which class.

Table 5.1: There are six radar hand gestures in the dataset. The type of hand gesture is an estimation because it is not known what the exact gesture where.

Gesture	**type**
Gesture 1	Grab
Gesture 2	Finger Rub
Gesture 3	Finger Waves
Gesture 4	Circle
Gesture 5	Swipe
Gesture 6	Top-Down

Two subjects, each performing six types of hand gestures, a 100 times, make up the radar dataset. The recordings consist of 50 frames each, with a size of 32-by-32 pixels. Only three channels were provided. Each hand gesture video consists of 50 frames, but the gesture recordings do not always start at the first frame. In short: two subjects, six gestures each, 100 videos per subject with 50 frames of 32-by-32 pixels across three channels.

Frames In figures 5.1 different frames are given from gesture five. The figures are scaled for making the blobs human visible. The reason to scale the frames if for making them human readable. In equation 5.1 the scaling formula is shown. The im is the absolute taken from a radar frame. For each frame (im) the $vmax$ is the maximum pixel value and the $vmin$ the minimal pixel value of the frame.

$$scaled\_img = \frac{im - vmin}{vmax - vmin} \tag{5.1}$$

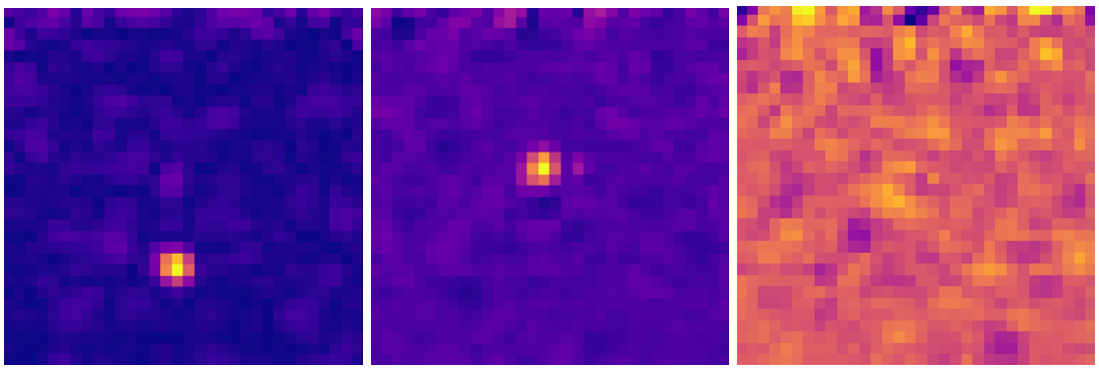

Figure 5.1: On the left frame 1 and the middle frame 50 is given for gesture 5. On the right, an empty frame is given. *The printed version shows a different colour perspective between the frames.*

Differences between recordings Different recordings will produce a slight difference in the representation of the gesture. This variation happens in real life, which needs to be added to the dataset. Otherwise, during training, it overfits on a specific dataset. Figure 5.2 shows two different recordings from the two subjects. Both showing the same gesture, but there is a variation in both recordings. There can be several reasons for this representation because those frames are scaled like figure 5.1. Hence, the rest of the video is not visible due to the scaling. Alternatively, the different circle motions the subjects performed can vary in speed or size.

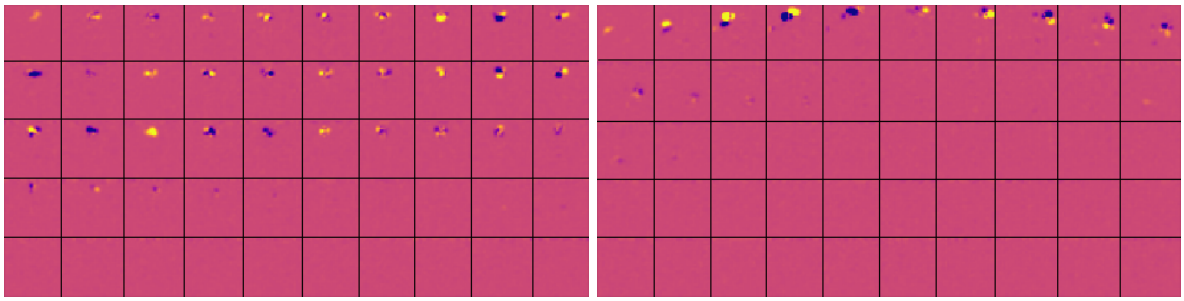

Figure 5.2: Here, gesture four is performed where the frames are stitched behind each other from left till right. There are six frames separated across the rows and columns, where each frame uses 32 rows and columns. In total, there are 50 frames in both images. On the left, person one, and on the right person two is shown.

Evenly distributed The sets should be evenly distributed to improve accuracy and not overfit on both train and test set, as shown in figure 5.3. It is essential to evenly distribute both of these sets; otherwise a small bias to one class from a network will be visible.

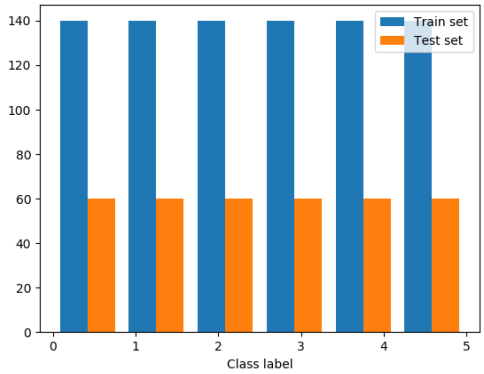

Figure 5.3: Between each class label, the number of samples is evenly distributed.

5.2. Baseline performance

For the baseline, the researcher chose the Convolution LSTM (ConvLSTM) [14]. The ConvLSTM is specialised in video classification because it has convolution filters in each LSTM neuron. Before the experiment starts, the train set and test set are generated. Both are evenly distributed in class distributions, and each class has the same set size. The other comparisons use the same train and test set. The same random seed initialises the network.

Layer (type)	Output Shape	Param #
conv_lst_m2d_1 (ConvLSTM2D)	(None, 32, 32, 50)	92000
batch_normalization_1 (Batch	(None, 32, 32, 50)	200
dropout_1 (Dropout)	(None, 32, 32, 50)	0
flatten_1 (Flatten)	(None, 51200)	0
dense_1 (Dense)	(None, 6)	307206
activation_1 (Activation)	(None, 6)	0

Total params: 399,406
Trainable params: 399,306
Non-trainable params: 100

Figure 5.4: The architecture of the video classifier using a Conv_LSTM

Figure 5.4 shows the ConvLSTM with 50 filters. The learning optimiser is RMSprop. The model uses the ConvLSTM 2D layer with 50 ($ConvLSTM\_50$) or six ($ConvLSTM\_06$) filters and a kernel size of 3-by-3. After this layer, the Batch-normalisation is added with a dropout-layer for protecting overfitting in the network. After that, a flatten-layer and a dense-layer are added . For the activation function, a Softmax layer is used. During training, the batch size of 64 is used with a validation split of 30%. The models chosen were from the Keras example website [65].
This thesis used the stripped-down version, with the LSTM-layer, dense- and dropout layer removed, which did not change the accuracy, but increased the computational complexity, learning the architecture more gently. Table 5.2 shows the accuracies for both networks. With some more tuning, it would be possible to improve the results, but that was out of the scope of the thesis.

Table 5.2: Comparison of two different ConvLSTMs both having different numbers of input filters. The first has six filters the second 50 filters.

Architecture	Parameter count	Test set accuracy
ConvLSTM_06	35K	97.0%
ConvLSTM_50	400K	95.0%

5.3. Spatial classifier

The spatial classifier is the input filter of the network. This section discusses the presented method further and alternatives that can be used with different datasets. The robustness is also measured and compared to the baseline and the training methodology.

5.3.1. How to determine the performance

The CNNs are state-of-the-art spatial classifiers for image classification. This work uses CNNs to label each frame of the gesture video. As no prior work exists on this dataset (i.e. a CNN classifier used to label this dataset), this approach is novel and able to learn on the dataset. The work [6] uses 3D-CNNs for Range-Doppler hand gesture recognition. The state-of-the-art version of the VGG [37], MiniVGG [66], has a small size when compared to works as described in the studies of [41], and achieves an accuracy of 29.47%. If a fully connected network replaced the CNN, the accuracy reached 40.13%. The spatial classification accuracy is good if it is in the range of 30% to 60% because the network is small, and it avoids overfitting. The study avoided using CNNs in the final design due to the unfeasibility of mapping a CNN onto an SNN. Consider the equations 5.2 and 5.3, even though the accuracy is 40.98%, class two is favoured when compared to other classes. This will be a problem for the spatial classifier, and the stages after the spatial classifier, as many frames could potentially be labelled as class two instead as class zero. An ideal confusion matrix would be an evenly distributed, and the diagonal needs to be as high as possible for that data set.

$$A = \begin{bmatrix} T\backslash P & c0 & c1 & c2 \\ c0 & \mathbf{50} & 101 & 1980 \\ c1 & 28 & \mathbf{307} & 1672 \\ c2 & 1 & 10 & \mathbf{2276} \end{bmatrix} \tag{5.2}$$

$$Accuracy\ result: \frac{trace(A)}{sum(sum(A))} * 100\% = 40.98\% \tag{5.3}$$

5.3.2. Multi-Layer perceptron

The spatial classifier uses the Multi-Layer perceptron (MLP). The following sections explore if there are enough data samples to train the MLP. After that, the author discusses two different versions of the MLP. One has a bias, and the other does not have a bias in the neuron model. Then, the weight matrices of the input neurons and the heat maps of the video frames are used to determine respectively, the neurons' focus and the network's performance. The last section looks at the improvements of the retrained network methodology.

Train set size

To verify that there were enough data samples given, the final network performance was tested as a function of different dataset sizes. However, the results showed that using only $\frac{1}{10}$ was enough to achieve 84.54% accuracy. The reason for this result is the high homogeneity of the dataset. This homogeneity could have been introduced either during the recording or during the Range-Doppler transformations.

Bias in the neurons

The weights will determine the steepness of the activation function, but the bias will do something different. The biases make the neuron's activation function more flexible, but they allow neurons to produce spikes without any input activation. The biases come from the spatial classifier . Each neuron had one bias and many weights. In the LIF-neurons, there is a threshold value that determines when the neuron should fire. The bias is a threshold value that makes a neuron more or less critical to the network, as it can add a negative bias to suppress that neuron (if a negative bias is supported).
The performance improvement obtained by adding a bias is considerable. Without the biases, the average accuracy was 27%. With the biases, the average accuracy was 53%. It was clear from the confusion matrix (equation 5.4) that the model was overfitted on class four and five. The accuracy for this confusion matrix was 29.15% without the biases. Equation 5.5 shows the confusion matrix with bias. In this case, the accuracy of was 48.23%. Both setups had the same learning parameters.

$$A = \begin{bmatrix} T\backslash P & c0 & c1 & c2 & c3 & c4 & c5 \\ c0 & \mathbf{30} & 475 & 0 & 4 & 160 & 331 \\ c1 & 2 & \mathbf{716} & 0 & 8 & 425 & 2849 \\ c2 & 0 & 101 & \mathbf{4} & 19 & 1249 & 2627 \\ c3 & 0 & 132 & 3 & \mathbf{229} & 421 & 3215 \\ c4 & 2 & 196 & 1 & 3 & \mathbf{2034} & 1764 \\ c5 & 0 & 13 & 0 & 1 & 2 & \mathbf{3984} \end{bmatrix} \rightarrow accuracy = 29.15\% \tag{5.4}$$

$$A = \begin{bmatrix} T\backslash P & c0 & c1 & c2 & c3 & c4 & c5 \\ c0 & \mathbf{2210} & 247 & 169 & 560 & 133 & 681 \\ c1 & 1034 & \mathbf{1194} & 544 & 532 & 259 & 437 \\ c2 & 597 & 198 & \mathbf{2242} & 543 & 199 & 221 \\ c3 & 861 & 277 & 795 & \mathbf{1532} & 84 & 451 \\ c4 & 724 & 284 & 308 & 301 & \mathbf{2204} & 179 \\ c5 & 1063 & 111 & 79 & 524 & 29 & \mathbf{2194} \end{bmatrix} \rightarrow accuracy = 48.23\% \tag{5.5}$$

Show weight graphs

Why is the MLP learning on the dataset? Can the weight graphs determine where it looks? Each input neuron fully connects to the input. From each neuron, the input weights present as an image. After some scaling the inputs can be made visible. Figure 5.5 shows the input weights of an input layer with 16 neurons for the given radar dataset. Not all 25 neurons of the MLP_A model are shown. It is essential to pinpoint the focus of the network when determining if the network is not overfitted. To find a pattern from the dataset and in some frames, it also looks at background movements. The frames are essentially blobs moving in time, but not for all hand gestures. For some hand gestures, the network also looks to smaller velocity changes in the Range-Doppler images. It is using more input neurons for the radar dataset results in empty neurons or neurons that respond to noise in the frames.

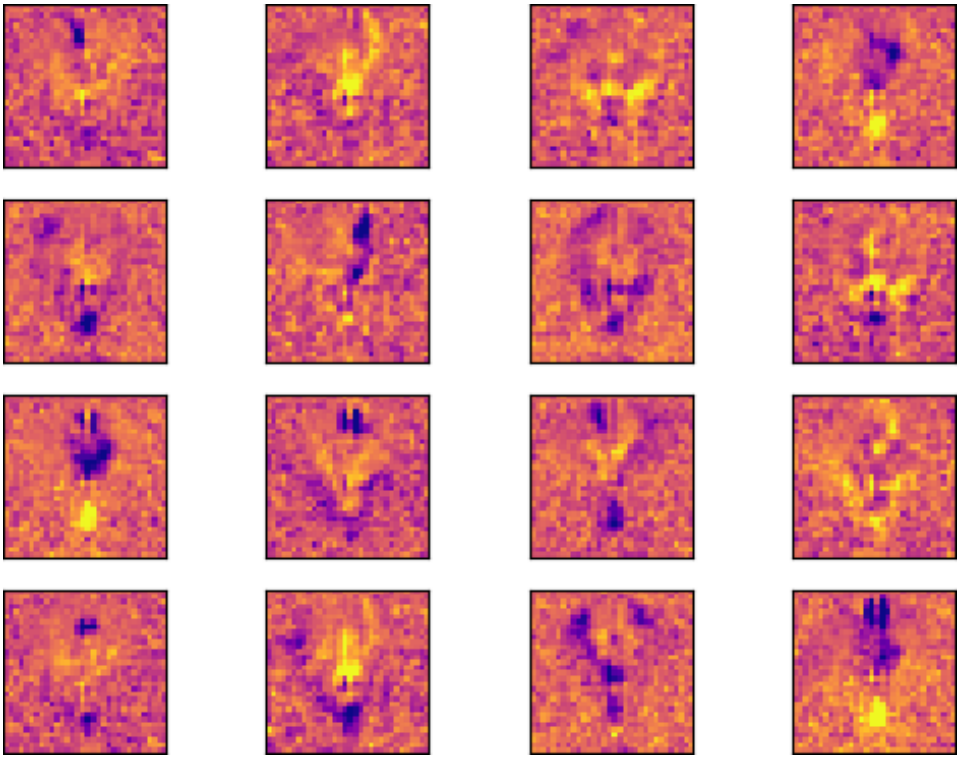

Figure 5.5: Here, the input neurons weigh images. It shows to which areas a specific class responds and to which it will not respond. On the top left is the first neuron. The second neuron is on the right of it. Blue means inadequate response, where yellow means maximum response and red is in-between those responses.

heatmap

The spatial classifier will label each frame of the hand gesture video with a number from zero till five. After filtering (as explained in Chapter 4), the output of the train set and test set is being labelled. Figure 5.6 shows the train set, where Figure 5.7 shows the test set. These figures are small screenshots from the full set. In the train set, many frames are labelled with different classes when in the ideal circumstances this would be label seven (unknown class) with zero (this part of the dataset is class zero). Appendix A.4 shows the full, labelled test set. It also shows that different classes have more frames labelled in time than others.

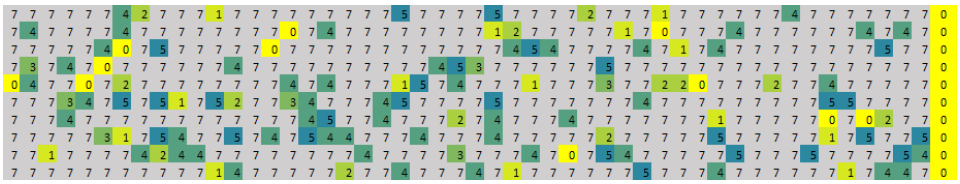

Figure 5.6: The figures show a part of the training set with the class label in the last column. Each row represents the hand gesture video frame. Class 7 represents the unknown frame.

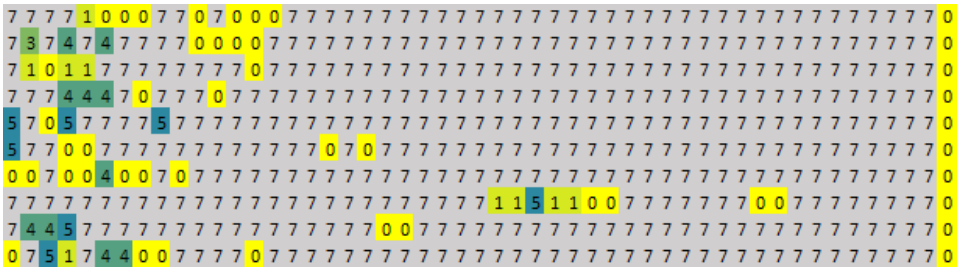

Figure 5.7: The figures show a part of the testing set with the class label on the last column. Each row represents the hand gesture video frame. Class 7 represents the unknown frame.

5.3.3. Feedforward threshold

After the first MLP network, the predictions are filtered using a simple threshold value. In the following figure 5.8, a prediction plot shows the accuracy versus the true label. Each frame of the dataset is labelled, and the prediction accuracy tracked for the correct label despite incorrect classification. In the figure, there is a small blob found near 0.2 (=20%). Here, the network misclassified the frames by the network; the frames did not contain enough information to indicate to which class it belonged. The tipping point of the graph is around 60%.
After the first MLP network, the predictions are filtered using a simple threshold value. In the following figure 5.8, a prediction plot shows the accuracy versus true label. Each frame of the dataset is labelled and kept track what the prediction accuracy was for the correct label even if it was not correctly classified. In the figure, there is a small blob found near 0.2 (=20%). Here the frames are miss classified by the network, hence the frames did not contain enough information to show that it belongs to the certain class. The tipping point of the graph is around 60%.

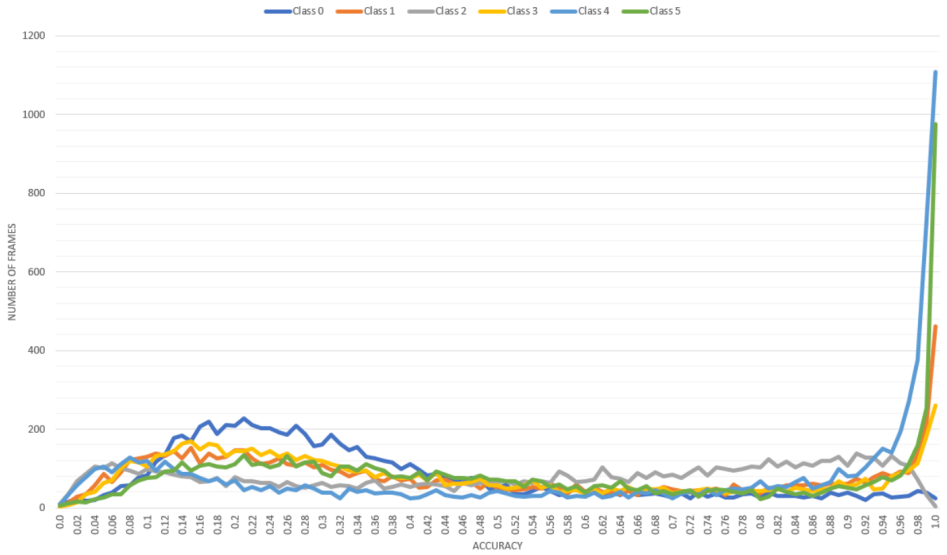

Figure 5.8: The prediction accuracy for the correct class label versus the number of frames the accuracy was the correct class.

If the threshold value was lower than 60%, more incorrect frames were labelled. If the accuracy was higher than 80%, fewer frames were labelled. In earlier experiments, some videos were fully labelled to 'unknown'. The closer the accuracy values were set to 100%, the more videos were completely labelled as unknown. Figure 5.9 shows that after the threshold accuracy was set at 85%, the final accuracy of the network dropped.

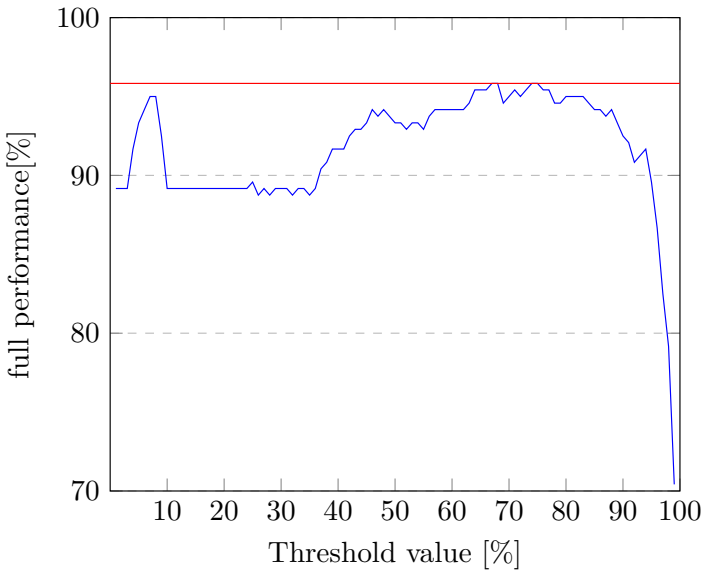

Figure 5.9: The threshold value of the feeedforward is compared with the full network performance.

5.3.4. Retraining

The retraining is an essential step in the learning process. In this step, the unknown label is added in the training architecture, and the network can be made smaller. The first-trained MLP network has 1024 inputs, with two hidden layers of each 25 neurons, and six outputs. The feed-forward pass generates new labels used in the retraining phase of the network. Chapter 4 shows the feed-forward, pass-filtering method. The retrained network size could be increased, as shown in table 5.3. The accuracy of the retrained networks is above 75%, but this is inaccurate. This statistic ignores the unknown frames. The section of the dataset frames, which the network labelled, was above 75%. Going to a bigger network size does not mean that the final network accuracy increased, it decreased. The accuracy of each frame was labelled more thoroughly, and more frames belonged to the unknown class. The test set accuracy compared with the originally given labels. These test labels were not filtered in the feed-forward step. All the frames of each video were labelled to the class to which the video belonged.

Table 5.3: Different network architectures before and after retraining. The final accuracy is based on the counting method.

Network	**Network size**	**Para. cnt**	**Accuracy**	**Unkn. frame**	**Final Acc.**
MLPA_20	1024x20x20x6	21K	49.12%	00.00%	90.43%
MLPA_25	1024x25x25x6	26K	50.44%	00.00%	89.17%
Retrained MLPA_20	1024x20x20x7	21K	79.42%	60.97%	87.50%
Retrained MLPA_25	1024x20x20x7	21K	75.28%	58.88%	93.33%
Retrained MLPA_25	1024x20x20x7	21K	74.63%	63.24%	94.33%

Both retrained MLPA_20 and retrained MLPA_25 are converted to an MLP with two hidden layers of 20 units. Table 5.3 shows that the network size was decreased and learned to label frames as unknown. This mimicking of the network increased the final performance by 5.16%. The *Final ACC* is done using the counting method, so this percentage is determined over all frames. Table 5.4 shows the most important hyper-parameters of the network. The first hidden layers were kept small to lower the number of neurons. When the first hidden layer had 100 neurons, the parameter count increased to 105K, where the final accuracy dropped. The threshold value for the retrained MLPA_25 with 93.33% accuracy had a threshold value set to 67%, and 94.33% had the accuracy set to 63%. These statistics indicate that during the retraining phase, the full potential of the filter is not used. During the sweep of many different setups, the value 63% was chosen to be the threshold value.

Table 5.4: The network parameters the retraining phase.

Parameter	MLP_25	Retrained MLPA_25
Input	1024	1024
Output	6	7
epoches	200	200
activation	RELU	RELU
solver	Adam	Adam
Learning rate	1e-3	1e-3
hidden-layers	(25, 25)	(20, 20)

5.3.5. Clustering

There are diverse types of clustering methods used to validate the performance of the network. This section explains these methods. Appendix A.1 gives a table is with the number of clusters each method has generated.

Self-Organizing map The SOM is an unsupervised learning method determined from the data sample, the closest neuron that best matches with that class. The results for the SOM layer was 23.64%. The reason for this would be the same as for the other clustering methods, as explained in the next paragraph.

Meansift The following figure 5.10 shows the variation of dataset sizes versus the clustering sizes. This dataset was obtained by adding the training and testing set, where each data sample has labelled frames. The *Red* line is the smoothed result of the *Blue* line, and the *Blue* line is the actual result. There were many tests done to understand this behaviour, but there was no clear explanation. The researcher calculated the bandwidth with the Scikit-learn *estimate_bandwith* function.

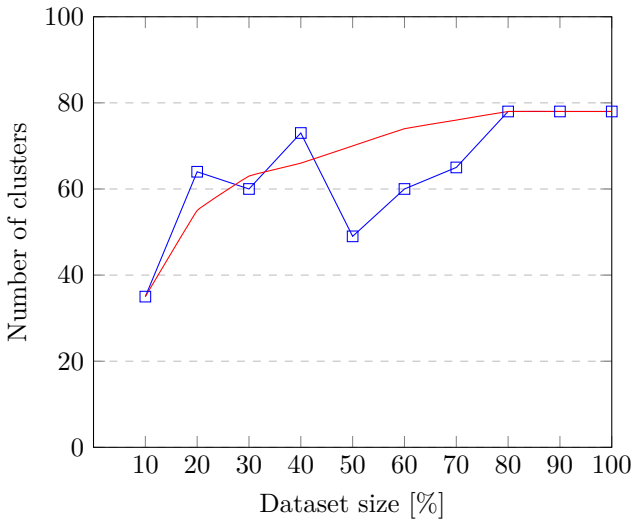

Figure 5.10: The dataset size versus the founded Meansift clusters.

The number of clusters is converted to 78 clusters on the full dataset size, which is promising. However, the study evaluated the final performance with a second MLP network and AdaBoost using a decision tree that never went higher than 22.62%. Plotting the number of frames, shows that almost the whole dataset (60K frames), 41K was labelled as cluster zero. Figure 5.11 shows how the clusters are distributed.

Figure 5.12 Here, each row represents the video of a gesture. It shows that many frames are labelled as unknown, and some did not have any other label zero. This small part of the dataset is the same

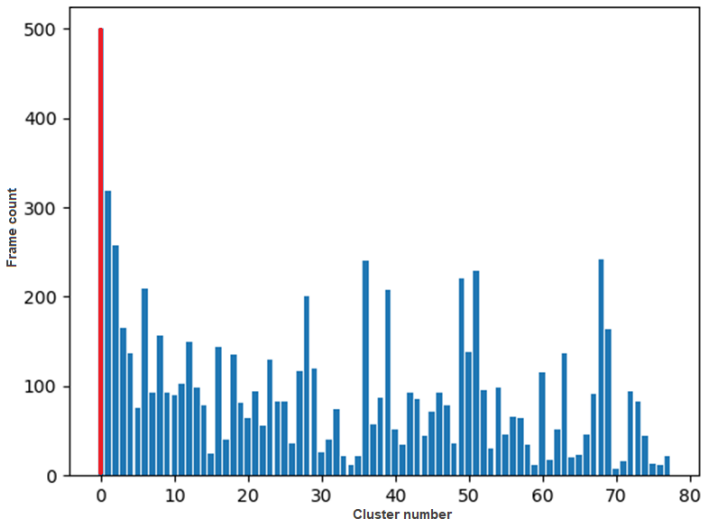

Figure 5.11: The distribution of each Meansift cluster versus the radar frames. Small note to make is that label 0 (red line) was originally 40973, but here set to 500 for visualization.

as the rest of the videos for other classes. Figure 5.2 shows that there are many different blobs at the beginning of the recordings. Figure 5.12 shows specific clusters at the beginning and the end. In many hand-gesture videos, the starting point of a gesture is at the start and not in the end.

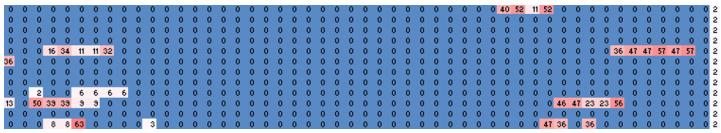

Figure 5.12: Each row represents the video of a gesture the last column is the class. Labelled using the Meansift method.

Summary The issue shown with Meansift is that many frames are labelled to a specific class. In other clustering methods, such as the SOM layer, the version of scaling and orientation feature extractor and the K-means that the few outliers of the dataset are clustered. When those clustering methods were classified, the temporal network could only label them with an average accuracy of 23% compared with the flipping-coin percentage of 16.67%. This is not usable. Even if empty frames are removed, or other methods used to lower the frameset, it did not help to get a more evenly distributed clustering.

The complexity involved to scale the number of clusters is the other issue with clustering methods. The SNN will run on an analog chip with room for hundreds of neurons; using over 5K of features will not be practical.

5.4. Temporal classifier

The temporal part of the network is where the videos are labelled to the class they belong. The unknown class was added to ignore frames that held no hand-gesture movement or common movements between classes. The next sections explain in-depth the reasons for training on the test set, which is divided into a newer test set. The author also discusses the temporal layer parameters.

5.4.1. Trained on the test set

There was a problem with the temporal classification; it did not achieve near state-of-the-art performance because it was trained on the training set. The training set should generally be overfitted on the data and achieve high accuracy, but this was not the case for the temporal classifier. The final performance was below 50% trained on the training set.
In the previous section on the heatmap, Figure 5.6 shows part of the train set and in Figure 5.7 shows the test set. The train set highlights numerous misclassifications, while more frames are labelled as unknown in the test set. The sets are differently classified, and therefore, would affect the performance of the temporal classifier. Hence, the final results should be compared to the test set that is generated from the new dataset. There are 240 videos for the original test set. In the temporal classifier, 140 videos are being used for training and 100 for testing. The absolute accuracy from the baseline is only done on these 100 videos.

5.4.2. Tempotron learned layer

The test set learns the Tempotron. This result is not directly comparable with the other methods as it uses less data for the training set, involved in a lower degree of accuracy. A few parameters are set in advance for the Tempotron learning-rule. The τ_s to a small value is only needed during training but is not present in the simulator. In table 5.5 are the parameters shown during training and table 5.6 gives the confusion matrix for the results. During the training of the temporal classifier, small changes of 10^{-20} or 10^{-19} could already be the difference between success and failure. Six inputs spike for the input classes and six output classes for the gesture classification in time. The in-house simulator defines the time step, V_rest, and threshold.
The accuracy achieved for the networks is 90.87%, using the retrained MLP_25 in the spatial classifier converted to spikes. The unknown class is replaced with no spike at that time step. There were some videos, which did not have an output response or multiple output responses simultaneously. These were ignored and labelled as unknown.

Table 5.5: The network parameters for training Tempotron

Parameter	**Value**
Input	6
Output	6
Time step	1μs
Learning rate	1e-19
V_rest	0
τ	$1e-4$
τ_s	$32e-8$
Threshold	$0.450e-13$

5.5. Robustness of the network

The many papers [5, 49, 67] on the Range-Doppler approaches did not discuss false detection of the datasets. The paper of [6] shows that the detection of datasets is essential to embedded information of an unknown gesture, but they did not determine if their approach is robust. The robustness test is a complex problem [68, 69]. It is difficult be confident when the architecture works in different environments, and perfect performed gestures do not bias the generated dataset. The author argues that robustness is more important than having the best accuracy. The approach of this thesis differs from other state-of-the-art approaches because it teaches the network to recognise the unknown gestures. This differs from other approaches [6], where final ANNs indicates the certainty about a complete gesture video instead of determining the frames.

Dataset setup The author removed a class from the dataset during training and added it in the testing phases to measure the robustness of the network. The selection of removed class was random as many frames were labelled as a different class. The removed class had the most frames predicted belonging to different classes. In the confusion matrix, in equation 5.6, the class is removed. The temporal classifier has labelled the removed class four, as unknown. The accuracy of the confusion matrix is 93.72%. The accuracy calculation ignores the unknown column for comparison between the setups. The confusion matrix in equation 5.7 showed it when the removed class was present during the training. The accuracy is calculated using the trace of the matrix divided by the sum of all values, but the unknown column ignored.

Presented architecture In the confusion matrix and heat-maps, it appears that there were a few wrongly labelled in that removed class. The class was firstly removed and then added again in the dataset. The following results are from the presented architecture giving the following confusion matrix 5.7 an accuracy of 95.61%.
The confusion matrix 5.6 shows that the classes not trained, were almost classified as unknown: meaning that random movements or noise can be filtered from the network, and the network's output will not spike to those hand-gesture videos.

$$A = \begin{bmatrix} True\backslash Predict & c0 & c1 & c2 & c3 & c4 & c5 & Unknown \\ c0 & \mathbf{36} & 0 & 0 & 0 & 0 & 0 & 4 \\ c1 & 3 & \mathbf{35} & 0 & 0 & 0 & 0 & 2 \\ c2 & 0 & 0 & \mathbf{32} & 0 & 0 & 0 & 8 \\ c3 & 0 & 1 & 4 & \mathbf{0} & 0 & 0 & 35 \\ c4 & 1 & 0 & 0 & 0 & \mathbf{39} & 0 & 0 \\ c5 & 1 & 0 & 1 & 0 & 0 & \mathbf{37} & 1 \end{bmatrix} \tag{5.6}$$

$$A = \begin{bmatrix} True\backslash Predict & c0 & c1 & c2 & c3 & c4 & c5 & Unknown \\ c0 & \mathbf{34} & 0 & 0 & 0 & 0 & 1 & 5 \\ c1 & 1 & \mathbf{39} & 0 & 0 & 0 & 0 & 0 \\ c2 & 1 & 0 & \mathbf{35} & 1 & 0 & 0 & 3 \\ c3 & 0 & 0 & 3 & \mathbf{35} & 0 & 0 & 2 \\ c4 & 1 & 0 & 0 & 0 & \mathbf{39} & 0 & 1 \\ c5 & 2 & 0 & 0 & 0 & 0 & \mathbf{37} & 1 \end{bmatrix} \tag{5.7}$$

ConvLSTM The expectation of the baseline, the ConvLSTM, was that the samples not trained on would be predicted to different classes. Equation 5.8 shows such a confusion matrix of the ConvLSTM. Here, the expectation was that the removed classes would have a low prediction accuracy. In practice, this was not the case; the prediction accuracy for the classes not trained on was still 100%. Some videos had a lower accuracy than 99% but were the wrongly classified videos from other classes.
The accuracy for the dataset where all classes were added and comprised of 100 videos had an accuracy of96%. When the class was removed from the training set and added again during testing, the accuracy

was 77% for 100 videos.

It is possible to train a different dataset for the LSTM, but this approach has some caveats. This dataset would have a different class added containing videos that should be ignored . The solution is probably robust against movements that are false detections or noisy frames. Nevertheless, it can fail with cross-class frames. The LSTM weighs the cross-class frames to a specific class for these frames, when they could have information that is relevant to random movements and should be ignored. The other limitation of this dataset is that it will not overfit on the provided extra class containing videos that should be ignored.

$$A = \begin{bmatrix} True \backslash Predict & c0 & c1 & c2 & c3 & c4 & c5 \\ c0 & \mathbf{17} & 0 & 0 & 0 & 0 & 0 \\ c1 & 0 & \mathbf{18} & 0 & 0 & 1 & 0 \\ c2 & 0 & 0 & \mathbf{13} & 0 & 0 & 0 \\ c3 & 0 & 0 & 11 & \mathbf{0} & 0 & 7 \\ c4 & 0 & 0 & 0 & 0 & \mathbf{15} & 0 \\ c5 & 1 & 0 & 0 & 0 & 0 & \mathbf{14} \end{bmatrix} \tag{5.8}$$

5.6. Comparison of the architectures

This section discusses the final accuracies and why the accuracies are not the only metric to consider.

The presented architecture developed with the training method Suino, learned to detect when the networks are unsure about a frame of a video. It is more robust against false detection than a state-of-the-art ConvLSTM because of the use of the temporal classifier that leaked in time. Random movements close to a class will be ignored. It could also make decisions if other classes were present in the dataset to use this in the decision. The final scores compared with the F1 metric was evenly distributed.

Table 5.6: The final results: comparing state-of-the-art with the presented architecture. Where robustness is determined using the method, one class was removed from the original dataset.

Network	**Parameter Count**	**Robust**	**Final Accuracy**
ConvLSTM_06	35K	No	97.00%
Retrained MLP + counting method	21K	No	94.58%
Retrained MLP + Tempotron	21K	Yes	90.87%

Table 5.6 presents the final model accuracies. The presented architecture learned with Suino is the retrained MLP_25 with Tempotron. The counting method shows a higher accuracy, then the SNN layer trained using the Tempotron learning rule. The lower accuracy is because of videos where the network did not respond. Those were wrongly labelled because the network needed to label these, but there was no class excited enough. It is hard to determine the robustness of the architecture and how well the network performs in a real-life environment. Looking only at the final network architecture, it does not portray the full picture. The presented architecture is, therefore, the winner: it is more robust to false detections and converts easier to the analog SNN.

6

Conclusions

6.1. Thesis contributions

This thesis explored a subset of ANNs with a focus on the radar dataset of hand gestures. Following this exploration, the researcher developed an architecture for hand gesture recognition. The analog neuromorphic spiking neural network processor constrained the design.

This study subdivided the presented architecture into two components: the spatial and temporal classification. An MLP trained the spatial part, which can be converted to an SNN. The temporal part used an SNN training method. Hence, the architecture was a pure SNN block that contained no digital circuits in between. The spatial classifier has multiple layers for robustness, where the temporal classifier was trained to classify the temporal aspect of the hand gestures.

The presented architecture was not only energy-efficient compared with a state-of-the-art deep-learning approach, but also small in size, fitting on the neuromorphic chip. The architecture can continuously classify the radar frames and remain energy-efficient.

This study presented a new training methodology for training such an architecture. The Suino-method is a stable training technique using proven classical methods to ensure stability and speed. It learns to classify which frames to ignore and classifies time-series data. Compared with the state-of-the-art Suino-method, an architecture that is not only small in size but more robust against false detections. The network will reject classes absent from the training set. The network size was decreased and taught to ignore frames; a smaller network was mimicking the larger network.

This thesis discusses different feature extraction methods and how these methods could be used in Suino. Using The Suino-method can convert the extraction methods to the SNNs. The given dataset did not perform well on the clustering methodologies and feature extractions.

The ConvLSTM had an accuracy of 97.0% compared to 90.87% of the presented architecture of Suino. Although the accuracy is lower than that of the ConvLSTM, Suino is more robust against false detections, can perform continuous classification , and is more energy efficient. The ConvLSTM without modification is not robust compared to unknown classes. The counting method showed promising results of 94.58% but is a time-slotted approach that needs gesture thresholds for the prevention of false detections.

6.2. Future Work

- In this thesis, SpikeProp is not further being discussed, but it uses multiple layers that help the architecture to be more robust. Both the spatial- and temporal classifier could benefit from this.
- To validate robustness, a new dataset should be generated containing more noise, different persons, different environments, and gestures that are incorrectly performed. Also, the speed of the gesture should be a variable in the recording. The dataset could also exists of vector gestures, where each gesture is not entirely performed and in the temporal classifier stitched together.
- Different representation such as the micro-Doppler should be explored. The current architecture uses range-Doppler frames for classification. Methods from the sound classification should be explored. A significant difference between radar and sound signals is the frequency.
- A study that focused on improving the robustness of the fixed-threshold choice in Suino.
- A broader search in different datasets has to be done to understand how well Suino can be generalised.
- Currently, it is not known if the neurons can remember the 20ms between frames. Therefore, a feasibility study for the LSTM could be conducted, to determine if it can remember time reliably.
- Is it possible to make the counting method more robust and energy-efficient?

6.3. Patent

The training methodology presented in this thesis is being filed for a patent application. At the time of writing this thesis, the patent number is unknown.

Bibliography

[1] D. Sterratt, B. Graham, A. Gillies, and D. Willshaw, *Frontmatter,* in *Principles of Computational Modelling in Neuroscience* (Cambridge University Press, 2011) pp. i–iv.

[2] B. P. Halpern and R. F. Thompson, *Foundations of Physiological Psychology,* (1967).

[3] 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 **117**, 500 (1952).

[4] J. Vreeken, *Spiking neural networks , an introduction,* Computing **7**, 1 (2002).

[5] J. Lien, N. Gillian, M. E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev, *Soli: Ubiquitous gesture sensing with millimeter wave radar,* in *ACM Transactions on Graphics*, Vol. 35 (2016) p. 142.

[6] S. Hazra and A. Santra, *Short-Range Radar-Based Gesture Recognition System Using 3D CNN with Triplet Loss,* IEEE Access **7**, 125623 (2019).

[7] T. J. Sejnowski, *The unreasonable effectiveness of deep learning in artificial intelligence,* Proceedings of the National Academy of Sciences , 201907373 (2020).

[8] E. Iranmehr, S. B. Shouraki, M. M. Faraji, N. Bagheri, and B. Linares-Barranco, *Bio-Inspired Evolutionary Model of Spiking Neural Networks in Ionic Liquid Space,* Frontiers in Neuroscience **13** (2019), 10.3389/fnins.2019.01085.

[9] E. M. Izhikevich and G. M. Edelman, *Large-scale model of mammalian thalamocortical systems,* Proceedings of the National Academy of Sciences of the United States of America **105**, 3593 (2008).

[10] E. Musk, *Tesla's approach to building an autonomy platform is focused on high compute efficiency machine learning. primarily vision...* (2017).

[11] H. Paugam-Moisy and S. Bohte, *Computing with spiking neuron networks,* Handbook of Natural Computing **1-4**, 335 (2012).

[12] S. Oh, D. Kwon, G. Yeom, W.-M. Kang, S. Lee, S. Y. Woo, J. S. Kim, M. K. Park, and J.-H. Lee, *Hardware Implementation of Spiking Neural Networks Using Time-To-First-Spike Encoding,* (2020), arXiv:2006.05033 .

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors,* Nature **323**, 533 (1986).

[14] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting,* Advances in Neural Information Processing Systems **2015-Janua**, 802 (2015), arXiv:1506.04214 .

[15] S. Zhou and W. Wang, *Object Detection based on LIDAR Temporal Pulses using Spiking Neural Networks,* (2018), arXiv:1810.12436 .

[16] R. V. Florian, *The chronotron: A neuron that learns to fire temporally precise spike patterns,* PLoS ONE **7** (2012), 10.1371/journal.pone.0040233.

[17] S. Woźniak, A. Pantazi, T. Bohnstingl, and E. Eleftheriou, *Deep learning incorporating biologically-inspired neural dynamics,* (2018), arXiv:1812.07040 .

[18] J. Mes, E. Stienstra, X. You, S. S. Kumar, A. Zjajo, C. Galuzzi, and R. Van Leuken, *Neuromorphic self-organizing map design for classification of bioelectric-timescale signals,* Proceedings - 2017 17th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2017 **2018-Janua**, 113 (2018).

[19] M. B. Buller, *SpringerReference*, Ph.D. thesis, TU Delft (2020).

[20] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, *Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks,* Pattern Recognition **94**, 87 (2019), arXiv:1804.00227 .

[21] E. M. Izhikevich, *Which model to use for cortical spiking neurons?* IEEE Transactions on Neural Networks **15**, 1063 (2004).

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation. In: Rumelhart D E, McClelland J L et al. (eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge, MA **1**, 318 (1986).

[23] A. Tavanaei, M. Ghodrati, S. Reza Kheradpisheh, T. Masquelier, and A. Maida, *Deep Learning in Spiking Neural Networks*, Tech. Rep., arXiv:1804.08150v4 .

[24] N. K. Kasabov, *Time-Space, Spiking Neural Networks and Brain-Inspired Artifical Intelligence* (2019).

[25] T. Masquelier and S. J. Thorpe, *Unsupervised learning of visual features through spike timing dependent plasticity,* PLoS Computational Biology **3**, 0247 (2007).

[26] T. V. Bliss and T. Lømo, *Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path,* The Journal of Physiology **232**, 331 (1973).

[27] B. Gardner and A. Grüning, *Supervised Learning in Spiking Neural Networks for Precise Temporal Encoding,* (2016), 10.1371/journal.pone.0161335.

[28] P. U. Diehl and M. Cook, *Unsupervised learning of digit recognition using spike-timing-dependent plasticity,* Frontiers in Computational Neuroscience **9**, 99 (2015).

[29] J. H. Lee, T. Delbruck, and M. Pfeiffer, *Training deep spiking neural networks using backpropagation,* Frontiers in Neuroscience **10** (2016), 10.3389/fnins.2016.00508, arXiv:1608.08782 .

[30] S. M. Bohte, H. L. Poutré, and J. N. Kok, *SpikeProp: Error-Backpropagation for Networks of Spiking Neurons,* in *Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2000)* (2000) pp. 419–425.

[31] N. Caporale and Y. Dan, *Spike Timing–Dependent Plasticity: A Hebbian Learning Rule,* Annual Review of Neuroscience **31**, 25 (2008).

[32] J. Wu, Y. Chua, M. Zhang, Q. Yang, G. Li, and H. Li, *Deep Spiking Neural Network with Spike Count based Learning Rule,* Proceedings of the International Joint Conference on Neural Networks **2019-July** (2019), arXiv:1902.05705 .

[33] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, *A solution to the learning dilemma for recurrent networks of spiking neurons,* bioRxiv , 738385 (2019).

[34] D. Spessot, *Thesis Design Space Exploration of a Spiking LSM Classifier for RADAR applications,* , 63 (2019).

[35] R. Gütig and H. Sompolinsky, *The tempotron: A neuron that learns spike timing-based decisions,* Nature Neuroscience **9**, 420 (2006).

[36] B. Rueckauer, I. A. Lungu, Y. Hu, M. Pfeiffer, and S. C. Liu, *Conversion of continuous-valued deep networks to efficient event-driven networks for image classification,* Frontiers in Neuroscience **11**, 1 (2017).

[37] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, *Going Deeper in Spiking Neural Networks: VGG and Residual Architectures,* Frontiers in Neuroscience **13**, 1 (2019), arXiv:1802.02627 .

[38] H. Hazan, D. Saunders, D. T. Sanghavi, H. Siegelmann, and R. Kozma, *Unsupervised Learning with Self-Organizing Spiking Neural Networks,* in *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2018-July (Institute of Electrical and Electronics Engineers Inc., 2018) arXiv:1807.09374 .

[39] R. Nusselder, *Spike-based Long Short-Term Memory networks,* (2017).

[40] M. Weiler, F. A. Hamprecht, and M. Storath, *Learning Steerable Filters for Rotation Equivariant CNNs*, Tech. Rep., arXiv:1711.07289v3 .

[41] M. J. A. Van Wezel, L. J. Hamburger, and Y. Napolean, *Fine-grained Classification of Rowing teams,* (2019) p. 8, arXiv:1912.05393v1 .

[42] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, *Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware,* in *2016 IEEE International Conference on Rebooting Computing, ICRC 2016 - Conference Proceedings* (Institute of Electrical and Electronics Engineers Inc., 2016) arXiv:1601.04187 .

[43] R. P. Costa, Y. M. Assael, B. Shillingford, N. de Freitas, and T. P. Vogels, *Cortical microcircuits as gated-recurrent neural networks,* Advances in Neural Information Processing Systems **2017-Decem**, 272 (2017), arXiv:1711.02448 .

[44] A. Shrestha, K. Ahmed, Y. Wang, D. P. Widemann, A. T. Moody, B. C. Van Essen, and Q. Qiu, *A spike-based long short-term memory on a neurosynaptic processor,* in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, Vol. 2017-Novem (Institute of Electrical and Electronics Engineers Inc., 2017) pp. 631–637.

[45] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, *Long short-term memory and learning-to-learn in networks of spiking neurons,* Advances in Neural Information Processing Systems **2018-Decem**, 787 (2018), arXiv:1803.09574 .

[46] I. Pozzi, R. Nusselder, D. Zambrano, and S. Bohté, *Gating sensory noise in a spiking subtractive LSTM,* in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 11139 LNCS (2018) pp. 284–293.

[47] X. Shi, Z. Chen, H. Wang, D. Y. Yeung, W. K. Wong, and W. C. Woo, *Convolutional LSTM network: A machine learning approach for precipitation nowcasting,* in *Advances in Neural Information Processing Systems*, Vol. 2015-Janua (2015) pp. 802–810, arXiv:1506.04214 .

[48] S. Skaria, A. Al-Hourani, M. Lech, and R. J. Evans, *Hand-Gesture Recognition Using Two-Antenna Doppler Radar with Deep Convolutional Neural Networks,* IEEE Sensors Journal **19**, 3041 (2019).

[49] S. Wang, J. Song, J. Lien, I. Poupyrev, O. Hilliges, E. Zurich, and G. Atap, *Interacting with Soli: Exploring Fine-Grained Dynamic Gesture Recognition in the Radio-Frequency Spectrum,* in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (ACM, New York, NY, USA).

[50] I. Nasr, R. Jungmaier, A. Baheti, D. Noppeney, J. S. Bal, M. Wojnowski, E. Karagozler, H. Raja, J. Lien, I. Poupyrev, and S. Trotta, *A highly integrated 60 GHz 6-channel transceiver with antenna in package for smart sensing and short-range communications,* IEEE Journal of Solid-State Circuits **51**, 2066 (2016).

[51] W. Ponghiran, G. Srinivasan, and K. Roy, *Reinforcement Learning With Low-Complexity Liquid State Machines,* Frontiers in Neuroscience **13**, 883 (2019).

[52] J. Chrol-Cannon and Y. Jin, *Learning structure of sensory inputs with synaptic plasticity leads to interference,* Frontiers in Computational Neuroscience **9**, 103 (2015).

[53] J. Materzynska, G. Berger, I. Bax, and R. Memisevic, *The jester dataset: A large-scale video dataset of human gestures,* in *Proceedings - 2019 International Conference on Computer Vision Workshop, ICCVW 2019* (2019) pp. 2874–2882.

[54] J. Wan, S. Z. Li, Y. Zhao, S. Zhou, I. Guyon, and S. Escalera, *ChaLearn Looking at People RGB-D Isolated and Continuous Datasets for Gesture Recognition,* in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (2016) pp. 761–769.

[55] G. Mesnil, Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, P. Vincent, A. Courville, and J. Bergstra, *Unsupervised and Transfer Learning Challenge: a Deep Learning Approach*, Tech. Rep. (2012).

[56] A. Nere, U. Olcese, D. Balduzzi, and G. Tononi, *A Neuromorphic Architecture for Object Recognition and Motion Anticipation Using Burst-STDP,* PLoS ONE **7**, 36958 (2012).

[57] G. O'Leary, J. Xu, L. Long, J. S. Filho, C. Tejeiro, M. Elansary, C. Tang, H. Moradi, P. Shah, T. A. Valiante, and R. Genov, *A Neuromorphic Multiplier-Less Bit-Serial Weight-Memory-Optimized 1024-Tree Brain-State Classifier and Neuromodulation SoC with an 8-Channel Noise-Shaping SAR ADC Array,* in *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, Vol. 2020-Febru (Institute of Electrical and Electronics Engineers Inc., 2020) pp. 402–404.

[58] S. Marsland, *Machine Learning - An Algorithmic Perspective.*, Chapman and Hall / CRC machine learning and pattern recognition series (CRC Press, 2009) pp. I–XVI, 1–390.

[59] T. Kohonen, *The Self-Organizing Map,* Proceedings of the IEEE **78**, 1464 (1990).

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in python,* Journal of Machine Learning Research **12**, 2825 (2011).

[61] D. G. Lowe, *Distinctive image features from scale-invariant keypoints,* International Journal of Computer Vision **60**, 91 (2004).

[62] Y. Cheng, *Mean Shift, Mode Seeking, and Clustering,* IEEE Transactions on Pattern Analysis and Machine Intelligence **17**, 790 (1995).

[63] A. K. Jain, *Data clustering: 50 years beyond K-means,* Pattern Recognition Letters **31**, 651 (2010).

[64] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, *Spiking Neural Networks and online learning: An overview and perspectives,* Neural Networks **121**, 88 (2020), arXiv:1908.08019 .

[65] jeammimi, *Keras documentation: Next-frame prediction with conv-lstm,* (2016).

[66] Adrian Rosebrock, PyImageSearch, 11 February 2019, *PyImageSearch - MiniVGGNet: Fashion MNIST with Keras and Deep Learning.*

[67] Y. Sang, L. Shi, and Y. Liu, *Micro hand gesture recognition system using ultrasonic active sensing,* IEEE Access **6**, 49339 (2018).

[68] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen, *Strike (with) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects,* Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition **2019-June**, 4840 (2018), arXiv:1811.11553 .

[69] T. Hagendorff and K. Wezel, *15 challenges for AI: or what AI (currently) can't do,* AI and Society **35**, 355 (2020).

A

Appendix

Information with extra results or other information that is not crucial to the report itself.

A.1. Clustering

In the following table A.1 several types of clustering methods found number of clusters. K-means was generated using 100 clusters because K-means uses the number of clusters as hyper-parameter. However, the result was the same as Meansift, which does not provide enough information about the dataset. Figure A.1 shows that most outliers are placed in clusters. The frames containing movements belonged to just a few clusters.
The *clusty_** is the method obtained from the stripped version of SIFT feature extractor [61]. The PCA showed that lowering the cumulative explained variance, the feature size would exist of 140 components. The *clusty_A* is filtered on the dataset based on the labels. If the label is ignored, and a unique sort is done on this matrix where each row represents a feature, the number of features is less. Therefore, this is called *clusty_B*.

Table A.1: Different network architectures and the number of clusters

Clustering method	**Number of clusters**
K-means	100
PCA	18528
clusty_A	15956
clusty_B	5084

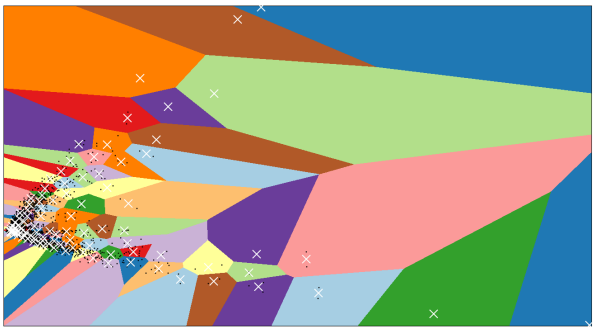

Figure A.1: K-means clustering with 100 clusters. As seen it that the radar datasets outliners are clusters, but not the gesture movements.

A.2. Conversion losses

It is no the core focus of this thesis to run the architecture in the analogue SNN hardware, but there were some performance losses during the conversion. Here follows list of different ways the network performance could be impacted:

- The RC model that has been used in the SNNs does not match the ReLu activation function. This ReLu activation function is used in the Multi-Layer Perceptron. During the conversion, this will have some inaccuracies.
- There exist no negative inputs, so the network can only work when this is not being used. The dataset performed better when the absolute was being taken in the tests. It is still unknown why this had a better result.
- The negative weights and biases are also a problem for the network. There exist no such negative voltage and currents in the analog design (currently), so those weights should be scaled or been set to zero. What lowers the accuracy.
- The chip will have its own unique inaccuracies where a small change in the resistor, capacitor, and current injectors for the bias will have already an impact with slight differences.
- Conversion from the numerical values into a spike encoding should not have a significant impact, but in the hardware itself, some noise can be added.

A.3. Overview of software components

During my thesis I wrote many scrips/programs. Here a small overview is given what languages have been used:

- Python - for the machine learning and dataset scripts (Anaconda for the platform).
- MATLAB - in-house simulator, machine learning, and analysis of results.
- Linux Bash - for running automatically experiments.
- C/C++ - for validating SNN architectures or testing a SNN method.
- Rust - for some quick analyses of results.
- HDL - for the counting method.
- Web-languages - To keep track of report from the experiments for analysing.

The next part will give an overview of the main libraries that have been used. There are many more

- Scikit-learn [60] - machine learning toolbox.
- Keras - deep learning toolbox.
- SNNSIM - the in-house simulator.
- Brian2 - spiking neural network toolbox.
- pyNN - language for building neural networks independent from the simulator.

A.4. Heatmap

In the following figure A.2 part of the dataset is shown how each frame was labelled. Each column represents a frame. Where the last column shows the actual label, each row represents a hand gesture video.

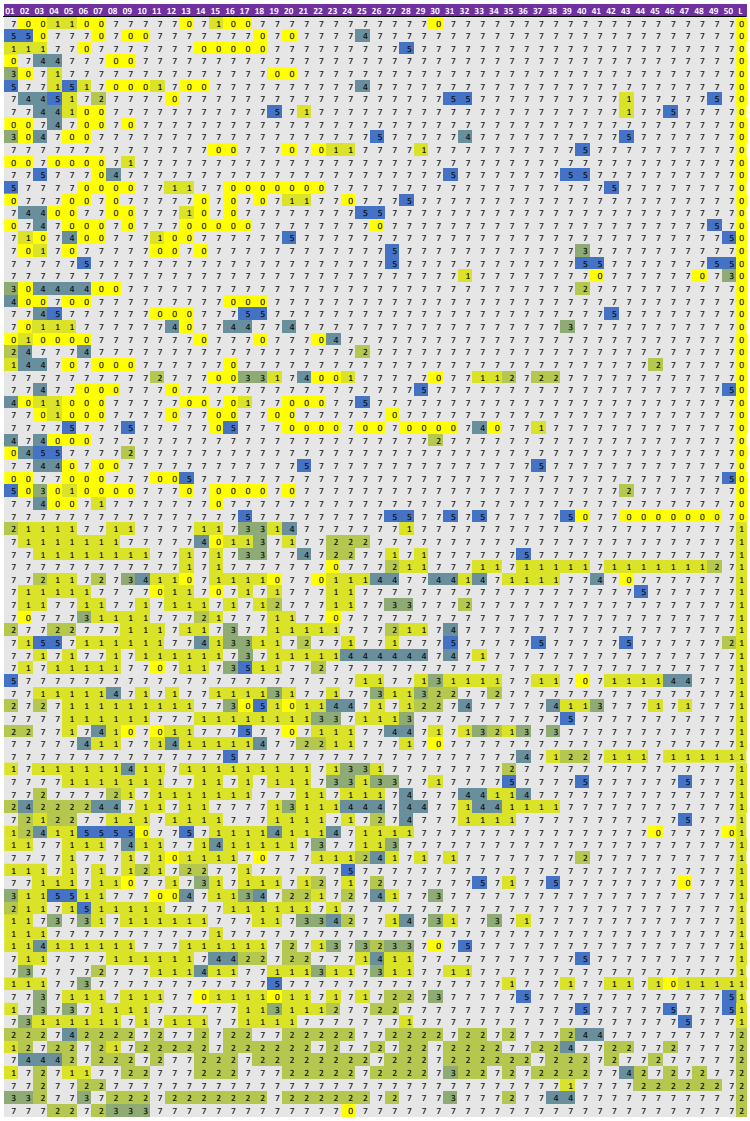

Figure A.2: Part of the labelled frames in the hand gesture radar videos.