Radar-based gesture recognition with spiking neural networks

by

Lucie de Ghellinck

Student Name	Student Number
Lucie de Ghellinck	5707137

Supervisors: Charlotte Frenkel and Francesco Fioranelli
Faculty: EEMCS Faculty, EI group, Delft

TU Delft

Abstract

Radar-based sensors are used to perceive their environment and objects of interest in a contactless manner and with robust performance in all weather and light conditions. One of the main drawbacks is the energy needed for the processing of radar data in order to extract its valuable information. Spiking neural networks are an emerging type of neural networks that aim to reduce the energy footprint of their computations while maintaining acceptable performance. To do so, the data is encoded through time in binary spikes to help leverage the low cost of additions. This is in stark opposition to the much higher cost of multiplications that are highly present in conventional artificial neural networks. The drawback of this energy gain is that the rate encoding adds an extra time dimension, hence increasing the latency between the acquisition of the radar data and the recognition of the corresponding gesture class.

More specifically, this work uses an air-marshalling dataset from the literature to exemplify a gesture-recognition problem. The first step is to replicate the well-known radar processing pipeline, and classification approach based on conventional neural networks to reach high classification accuracies. A validation accuracy of 98.5% and a test accuracy of 59.8% are reached on the full dataset (11 classes) and 86.7% on their 5 best classes (test set), which is about the same performance reported in the original dataset baseline.

The following steps propose an adaptation of this non-spiking pipeline to its spiking equivalent by optimising the trade-off between the model's latency, its memory requirements and its accuracy. This work also develops a strategy to tune spiking networks' thresholds to make the process of developing a spiking equivalent more efficient. For example, the spiking network can reach 94.5% validation accuracy using 100 encoding steps and only 5.7% of the initial memory requirements, and reach 46.8% on the test set. However, this trade-off can be shifted towards lower latency, lower memory, or higher accuracy according to the desired requirements.

Contents

List of Figures

List of Tables

Acronyms

ADC analog-digital converter. 27

ANN artificial neural network. 3, 8, 10, 11, 13, 14, 16–19, 24, 54, 55

BPTT back-propagation through-time. 11, 16, 17, 55

CA-CFAR cell-averaging constant false alarm rate. 15, 39

CFAR constant false alarm rate. 15, 39

CuBa current-based. 10

CW continuous wave. 1

DNN deep neural network. 2, 3, 11

FC fully-connected. 46, 61

FLOPS floating-point operations. 33

FMCW frequency-modulated continuous wave. 1, 3, 5, 27

FT Fourier transform. 1, 6, 13, 14, 19, 23, 24, 27, 29–32, 36, 39, 43–52, 54, 55, 59, III

HDL hardware description language. 51

HH Hudgkin-Huxley. 10

I&F integrate-and-fire. 2, 10, 14, 19, 21, 22, 25, 30, II

IF intermediate frequency. 6, 7, 13, 16, 56

LIF leaky integrate-and-fire. 10, 16, 19, 21

LSM liquid state machine. 16

MAC multiply-and-accumulate. 3, 8, 16, 20, 52, 54

MTI moving target indication. 13

OS-CFAR ordered statistics constant false alarm rate. 15, 39

RD range-Doppler. 1, 5, 14–16, 29, 30, 33, 36, 39, 43–46, 51–55, 58, 62

SNN spiking neural network. 2, 3, 5, 8, 10, 11, 13, 14, 16–19, 55

SRAM static random access memory. 53

STDP spike-time dependent plasticity. 11

SVM support vector machine. 16

TTFS time-to-first spike. 8, 10, 17

List of Symbols

α_{att}	attenuation factor between the transmitted signal x_T and the returning signal x_R
α_{cfar}	multiplicative value for CFAR algorithms
β	leakage factor
$\hat{x}$	approximated value of x_{int} after quantisation
λ	wavelength of the center frequency of the FMCW
ϕ_D	instantaneous phase of the intemerdiate frenquecy signal through chirps
ϕ_R	instantaneous phase of the intermediate frequency signal r through samples
ϕ_T	instantaneous phase of the transmitted signal x_T
σ	standard deviation
$A_{i,j}/B_{i,j}$	weights of a neural network
B	bandwidth of the frequencies in a chirp
b	bit width of a quantisation scheme
c	speed of light
d	distance between the radar and the object to be detected
d_{max}	maximum range
f_a	maximum frequency in a chirp
f_C	center frequency
f_c	minimum frequency in a chirp
f_D	frequency of the intermediate frequency signal that contains the velocity of an object
f_N	Nyquist frequency
f_R	frequency of the intermediate frequency signal that contains the raneg of an object
f_s	sampling frequency of the chirps
f_{max}	maximum frequency of the intermediate frequency signal
N_c	Number of chirps per chirp-sample map
N_s	Number of samples per chirp
r	intermediate frequency signal which results from the low pass filtering of the multiplication of the transmitted signal x_T and the returning signal x_R
r_i	r for chirp number i
s	scale factor of a quantisation scheme
$S[t]$	binary value representing the spike (or lack of spike) at time t
$S_x[t]$	binary value at time t representing the encoded version of X_n
T	number of encoding steps
t	time
T_c	time for a full chirp signal to go from a minimum frequency f_c to its maximum frequency f_a
t_d	delay between the transmitted signal x_T and the returning signal x_R
$T_{c,diff}$	time between two chirps

$t_{d,i}$	delay between the transmitted signal x_T and the returning signal x_R for chirp number i
$U[t]$	voltage potential at time step t
U_{th}	threshold voltage
v_{max}	maximum velocity
$W_{\Im}$	imaginary weights of a Fourier transform
$W_{\Re}$	real weights of a Fourier transform
X	analogue value
X_n	normalised analogue value of X
x_R	received chirp
x_T	transmitted chirp
x_{int}	integer version of a number that will be quantised
X_{max}	maximum of the analogue value X
X_{min}	minimum of the analogue value X
z	zero-point of a quantisation scheme
Δd	range resolution
Δf	frequency resolution
Δv	velocity resolution
LPF	low pass filter

1. Introduction

1.1 Background

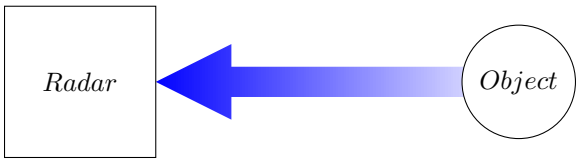
Gesture recognition using radar technologies targets a wide range of applications since such recognition enables human-machine interfaces. This can be seen through devices such as smart TVs, smart homes, virtual reality, etc. [1]. More specifically, radar systems can be used to detect human activities in an even wider variety of applications, ranging from human activity recognition and fall detection to support vulnerable individuals, to the automotive industry [2].

Indeed, autonomous driving requires an understanding of the environment over time, for example, to detect people in the environment to decide how to react. The industry currently uses lidar, radar, and vision cameras as sensors to do so. However, for the moment 10% of the energy usage of an automated car goes to its computing system: more precisely 4% of the added energy in a medium automated vehicle goes to its radar system [3]. Thus, there is a pressing need to optimise computational and energy resources for these sensors.

1.1.1 Radar Sensors

The basic type of radar sensor is based on continuous wave (CW) signals: the radar signal is transmitted and the reflection from objects is received back so that their velocity can be computed. However, this does not allow to find the object's distance. More complex systems use frequency-modulated continuous wave (FMCW). Figure 1 explains this concept: the radar sends a signal of increasing frequency (lighter to darker red) through time, this is called a chirp. This signal is reflected against the object, and the radar captures this attenuated version of the frequency-modulated sinusoid. Repeating a series of chirps through time allows the computation of the object's distance [2].

(a) The red arrow represents a sinusoid with increasing frequency.



(b) A few moments later, the attenuated sinusoid with increasing frequency returns to the radar.

Figure 1: Transmission and reception of one chirp of a FMWC signal.

The common FMCW pipeline is provided in Figure 2, a chirp is one row of the raw map, and consequent chirps are stacked next to each other. The process used to extract the relevant information from the raw input map has two main components. After an optional pre-processing step (some of which are developed in the Literature review in Section 3.1), two Fourier transforms convert the data into a range-Doppler (RD) map, which is a matrix where one dimension is proportional to the distance and the other is proportional to the velocity of the objects in a scene. The second main component is a classification network that extracts the relevant information from the RD map into the most probable gesture class.

The advantages of radar-based sensing are highlighted below ([2], [1], [4] and [5]):

- it is robust under all weather and light conditions since it can detect signals in darkness and through clouds, while vision sensors are unable,
- it can reach accurate range and relative velocity measurements for distances up to 250m,
- it preserves privacy compared to vision sensors that inherently record readable images,
- its processing pipeline is relatively simple, being based on FTs as illustrated in Figure 2,
- its form factor has been miniaturised to fit in smartphones, wristbands, headphone devices, etc.

1.1.2 Neuromorphic Solutions

Although the above factors have made radar signal processing an indispensable technology for object detection, the resolution increase needed for newer applications requires increasingly higher energy costs: for example, higher resolution can be achieved by using a higher sampling frequency and the number of computations will

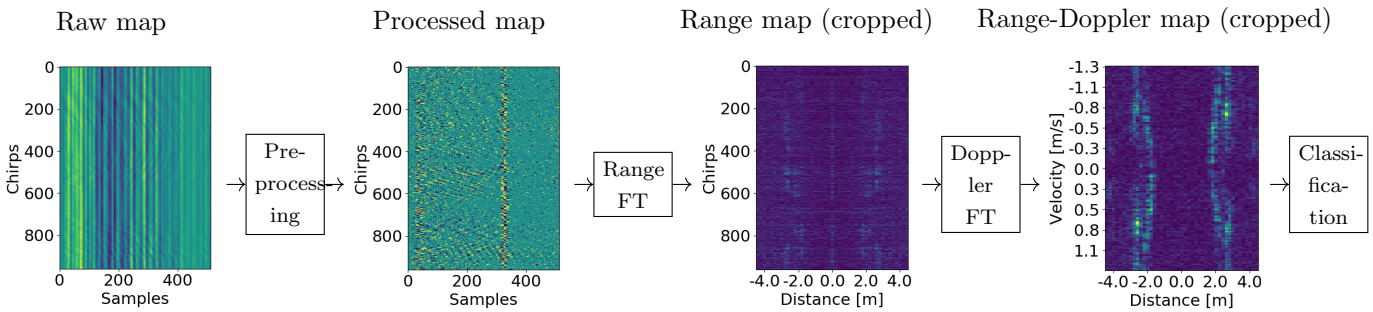


Figure 2: Radar-processing pipeline (with example maps from the dataset used in this thesis): the raw map is first pre-processed, then the frequency components are extracted by a first Range FT to obtain the distance and a second Doppler FT to obtain the velocity, finally this range-Doppler map can be passed in a classification algorithm to obtain the gesture class.

depend on how many more samples there are [3]. A recently proposed solution is to modify the usual radar processing pipeline algorithm into one that can be implemented on neuromorphic hardware.

The rise of spiking neural networks implemented in neuromorphic hardware marks a significant leap beyond deep neural networks (DNN). Although DNNs have excelled in computer vision, speech recognition, and natural language processing, their energy efficiency remains a critical issue. The computational power needed for the best deep learning models has increased by a factor of ten annually from 2012 to 2019. In stark contrast, the human brain operates at approximately 12-20 W, efficiently managing complex tasks and sensory input [6].

Neuromorphic computing aims to design new technologies that resemble the human brain more closely compared to conventional DNNs to attempt to copy its high energy efficiency [7]. The human brain contains about 10^{10} neurons that communicate via electrical signals that are known to have an all-or-nothing behaviour. This implies that if the incoming signal is strong enough, a spike is emitted and transmitted to the following neurons, otherwise, no signal is transmitted [8]. By greatly simplifying the biological process, its principal characteristics can be retained and mathematically modelled as an artificial spiking neuron. Those principal characteristics are the binary nature of the spikes and their sparsity, considered relative to a continuous signal that would always be present.

An example of a spiking neuron is provided in Figure 3: this is an integrate-and-fire (I&F) neuron which is the mathematically simplest artificial neuron, it uses rate encoding where larger numbers are represented by more spikes. There are other spiking neurons and encoding types that are described in Section 2.2, however, the simplest are represented here.

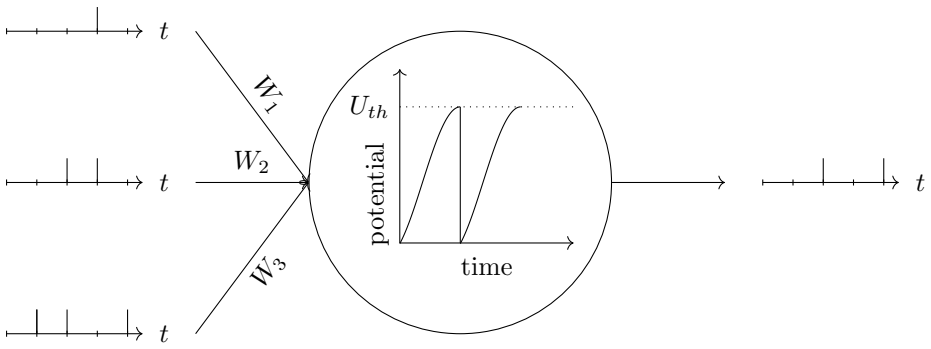


Figure 3: Simplest type of artificial spiking neuron: the integrate-and-fire (I&F) neuron. W_i are the weights, U_{th} is the firing potential. The spikes enter the neuron from the right to the left of the spike train through time and the corresponding weights are added to the potential. When the membrane potential crosses a threshold, the neuron emits a spike, the spikes at the right of the output train are emitted earlier.

In a similar way as in biological neurons, the artificial neurons possess a membrane potential that stores a value over time. Each of these discrete time is called a time/encoding step and the resulting latency is one of the main trade-off of SNNs. The incoming spikes increase the membrane potential by the value of their respective

weights: for example, W_3 is first added to the potential, at the following time step, W_1 and W_2 are added, etc. This addition process happens until the potential reaches a pre-defined firing threshold. The neuron then fires by emitting a spike and resetting its membrane potential. When the number of input spikes is proportional to the magnitude of the analogue input, Section 2.2 explains how the output number of spikes is also proportional to the magnitude of the analogue output in the corresponding non-spiking network [9].

SNNs have the following advantages ([1], [3], [4] and [10]):

- their high energy efficiency is useful for low-power edge applications, [11] shows a leakage power reduction of 34× and a dynamic power reduction of 49× when neurons spike for 10% of time steps (relative to the non-spiking equivalent). This is because they only process addition operations and no multiply-and-accumulate (MAC) operations to the opposite of DNNs where MAC operations consume the majority of their energy,
- their accuracies approach that of a normal ANN with simpler operations. For example, the literature research in Section 3.4 proves that the spiking version can drop less than 1% accuracy,
- they implement highly parallel processes through asynchronous computations,
- they are sparse and can process the data asynchronously,
- they aim at solving the memory bottleneck caused by the von Neumann architectures by making the information processing local, since the memory (synaptic weights) are close to the computing units (neurons).

Due to the requirement for lower energy requirements for radar processing, it is important to find the SNN equivalent for all stages of the processing pipeline to avoid issues linked to the conversion from non-spiking to spiking stages. This conversion challenge includes two aspects. On one side, there is an algorithmic decision: it might be more optimal to modify the way the computation is achieved if it is to be implemented on neuromorphic hardware. On the other hand, a spiking implementation requires the tuning of more parameters since the thresholds of the neurons need to be defined, and an optimal encoding strategy from continuous data to discrete spikes must be chosen. Furthermore, as per [7] and [6]:

- training an SNN is more complicated since the spikes are Dirac deltas which are not differentiable, gradient descent can therefore not be directly deployed without additional techniques such as surrogates,
- since the spike events are discrete, a loss occurs if the data that should be represented in a fraction of a spike is ignored,
- there is an added time dimension to accumulate the spikes that requires the definition of a trade-off between the performance and the time taken for the computation.

1.2 Contributions

As shown in the literature review in Chapter 3, to the best of knowledge, no paper has been found that fully implements a full-body gesture recognition pipeline in its spiking form. Therefore, this thesis attempts to address this gap by presenting an example of this full spiking pipeline by tackling the following objectives:

- a technique to tune the neurons' thresholds,
- a method to decide on the quantised precision to give to each spiking step of the pipeline,
- an analysis on how to reduce the memory footprint.

An example of the final spiking pipeline reaches 94.1% validation accuracy and 46.8% test accuracy while using 100 encoding steps and only 4.7% of the initial memory requirements.

This thesis is organised as follows:

- Chapter 2 lays down the mathematical background of radar sensors, more specifically for FMCW signals, it also develops the general radar-processing pipeline that will be used throughout this work. Following this, it explains the fundamentals of spiking neural networks, it first describes the different encoding strategies and describes the types of spiking neurons available,
- Chapter 3 is a literature research about each step of the radar processing pipeline and spiking neural networks, several gaps are highlighted that will be exploited in the following chapters,
- Chapter 4 selects the techniques from the literature research that will be further investigated, it outlines the methods and mathematical bases upon which the following implementations will build. More precisely, it analyses the procedures used in the literature and expands on them to incorporate SNNs into the initial processing pipeline,
- Chapter 5 applies the previous methods to an air-marshalling gesture dataset published in [12]. This proves the correctness of the methods outlined in Chapter 4 and demonstrates how to define the trade-off

between the desired accuracy and memory/energy performance. It also provides a small example of a hardware implementation of a spiking network in comparison to a non-spiking network,

- Chapter 6 finalises this thesis work and highlights different ways this work could be continued and improved.

2. Background

This chapter outlines the background on three separate subjects. First, Section 2.1 explains the theory behind radar processing. Then, Section 2.2 describes the theory of SNNs. Finally, since the final aim is to reduce the full pipeline's footprint, Section 2.3 outlines how values can be quantised.

2.1 Radar Signal Processing

This section describes how FMCW radar signals carry information about objects' distance and velocity. As shown in Figure 4 a FMCW radar works by sending towards and receiving a sequence of electromgnetic waveforms from the objects to detect. Sub-section 2.1.1 mathematically describes these signals called chirps. Sub-section 2.1.2 then explains how the transmitted and received chirps are mixed together so that the output signal contains information about the object's distance and velocity, which is stored in a range-Doppler (RD) map. Sub-section 2.1.3 then outline how to obtain the maximum values of the range and velocity represented in the RD map.

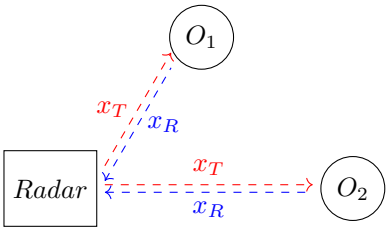


Figure 4: Sketch of a radar setting (O_1 and O_2 are objects to detect), the red x_T represent the transmitted radar signals, these signals are then reflected against the objects and return to the radar as the blue signals x_R.

2.1.1 Chirp

A transmitted chirp $x_T(t)$ is a modulated waveform whose frequency increases or decreases linearly over time t from a minimum f_c to a maximum f_a. Linear modulation can have different waveforms, but the most common is a sawtooth function as shown in Figure 5 [5]. This signal is mathematically represented in Equation 1: B is the bandwidth of the chirp defined as the difference between the minimum and the maximum frequency $B = (f_a - f_c)$ and T_c is the time taken by a chirp.

$$x_T(t) = \cos\left(2\pi f_c t + \pi \frac{B}{T_c} t^2\right) \tag{1}$$

When the chirp bounces off a static object and returns to the radar, it can be modelled as a delayed and attenuated version of $x_T(t)$ [5]. The transmitted signal is attenuated by α_{att} and the round-trip delay is t_d which can be expressed in terms of the distance between the radar and the object d as given in Equation 2 where c is the speed of light. The returning signal is therefore expressed as $x_R(t)$, shown in Equation 3 [5].

$$t_d = \frac{2d}{c} \tag{2}$$

$$\begin{aligned} x_R(t) &= \alpha_{att} x_T(t - t_d) \\ &= \alpha_{att} \cos\left(2\pi f_c (t - t_d) + \pi \frac{B}{T_c}(t - t_d)^2\right) \end{aligned} \tag{3}$$

Figure 5 illustrates the time delay t_d for the received signal x_R obtained from static objects O_1 and O_2

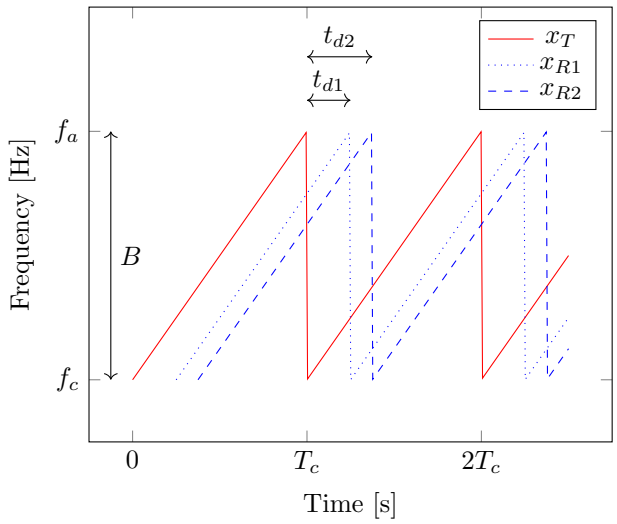


Figure 5: FMCW chirp frequency through time for a transmitted signal x_T and its corresponding received signal x_R when the signal modulation is a sawtooth function.

(from Figure 4). Since object O_2 is further away, the signal will take longer to arrive at the radar, as depicted by the longest delay t_{d2} relative to t_{d1}.

2.1.2 Intermediate Frequency Signal

The returning signal $x_R(t)$ from one object is mixed with the transmitted signal $x_T(t)$ at the in-phase receiver of the radar to create the intermediate signal, which is drawn in Figure 6. When this signal is low-pass-filtered, the frequency component f_c is removed to create the filtered intermediate frequency (IF) signal $r(t)$ shown in Equation 4. More details of this calculation and the following calculations can be found in Appendix 7.1.

$$r(t) = LPF\big[x_T(t)x_R(t)\big] = \frac{\alpha_{att}}{2}\cos\left(2\pi \underbrace{\frac{B}{T_c}t_d}_{f_R} t + \underbrace{2\pi f_c t_d}_{\Phi_R}\right) \tag{4}$$

This signal $r(t)$ can be analysed in two ways to extract both the object's range and its velocity.

Range

The frequency f_R of $r(t)$ in Equation 4 is proportional to the delay t_d. According to Equation 2, the frequency is therefore proportional to the distance or range d, as shown in Equation 5 [5]. This means that one chirp's frequency spectrum, obtainable via a FT, shows peaks at frequencies corresponding to the distances between the radar and objects. When many objects are present, the low-passed-filtered signal contains a series of peaks at frequencies corresponding to the distances of the objects. More details are found in Appendix 7.1.

$$f_R = \frac{B}{T_c}t_d = \frac{B}{T_c}\frac{2d}{c} \tag{5}$$

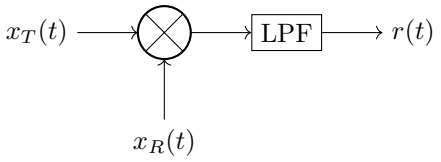


Figure 6: Generation of the IF signal $r(t)$ from the mixing and low-pass-filtering og the transmitted signal $x_T(t)$ and its reflection $x_R(t)$.

Velocity

The above highlighted how to obtain the range from the frequency of the IF signal of a single chirp. Given that the distance travelled between chirps is Δd and that the corresponding time delay is Δt_d according to Equation 2, then the velocity is expressed as $v = \Delta d/T_{c,diff} = (c\Delta t_d)/(2T_{c,diff})$ where $T_{c,diff}$ is the time between chirps. By renaming the IF signal of each chirp i by $r_i(t)$, then Equation 6 shows how this extra distance Δd is proportional to the frequency f_D of the IF signal across the chirps [3]: λ is the wavelength of the radar signal. The details of the calculation are found in Appendix 7.1.

$$\begin{aligned} r_i &= \frac{\alpha_{att}}{2}\cos\left(2\pi \underbrace{f_c\Delta t}_{f_D}\cdot i + \underbrace{2\pi t_{d,0} f_c}_{\Phi_D}\right) \\ v = \frac{\Delta d}{T_{c,diff}} &= \frac{c\Delta t_d}{2T_{c,diff}} = \frac{cf_D}{2f_cT_{c,diff}} = \frac{\lambda}{2T_{c,diff}}f_D \end{aligned} \tag{6}$$

To summarise, a Fourier transform applied along the samples will provide the objects' distances, which is called the range Fourier transform. Another applied along the chirps will provide their velocity, which is called the Doppler Fourier transform.

2.1.3 Maximum Range and Velocity

Range

In the range dimension, the maximum frequency that can be obtained is the Nyquist frequency f_N, which avoids aliasing, which is equal to half of the sampling frequency. Using Equation 5, the maximum range detected is therefore expressed in Equation 7 [4].

$$d_{max} = \frac{cT_c}{2B} \cdot f_N = \frac{cT_c}{2B} \cdot \frac{f_s}{2} \tag{7}$$

Velocity

The velocity information is saved in the IF signal's phase Φ_R and, by definition, phases are bounded $\Phi_R \in [-\pi, \pi]$, the equivalent frequency is hence bounded by $f_D \in [\frac{-1}{2}, \frac{1}{2}]$. This enables the maximum velocity to be found in Equation 8.

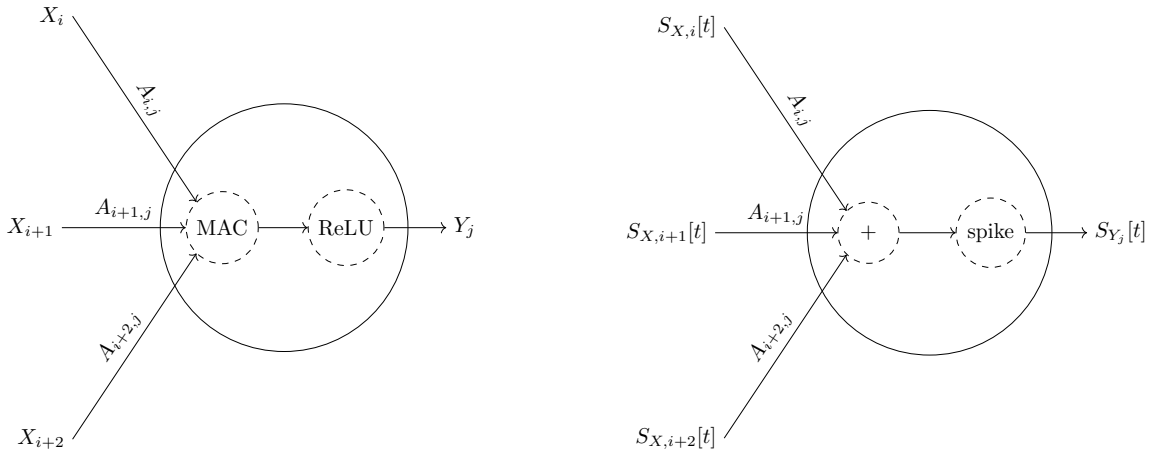
$$v_{max} = \frac{\lambda}{2T_{c,diff}} f_{D,max} = \frac{\lambda}{2T_{c,diff}} \frac{1}{2} = \frac{\lambda}{4T_{c,diff}} \tag{8}$$

2.2 Spiking Neural Networks

Neuromorphic algorithms work with binary spikes that encode information through time instead of using the value's magnitude. This is an inspiration from the brain since biological neurons communicate through electrical impulses of about the same amplitude. Figure 7 illustrates the difference between a non-spiking ANN neuron in Figure 7a and a spiking neuron in Figure 7b. The output of the ANN is a series of MAC operations between the inputs X_i and the weights $A_{i,j}$: the inputs X_i are analogue and the output Y_j too. On the other hand, the spiking neuron only sums the weights $A_{i,j}$ when there is an incoming spike S_{X_i} coming from the corresponding neuron: both the inputs $S_{X,i}$ and output $S_{Y,j}$ are binary spikes. The advantage of spike-based calculations is, therefore, that there are no multiplications of the input. This process will be further developed in Sub-section 2.2.2 [6].

In this work, it is assumed that the reader is aware of the theory behind non-spiking ANNs. For more information, the reader can read the book "Dive into Deep Learning" [13] which explains the mathematics of neurons, neural layers, weight training strategies, etc.

The first step in generating an SNN is to encode the data, there are two main ways to do so which are explained in Sub-section 2.2.1. As stated above, Sub-section 2.2.2 outlines the different spiking neurons and their mathematical expressions. Following this, Sub-section 2.2.3 explains how the weights of an SNN are selected.



(a) Non-spiking neuron

(b) Spiking neuron

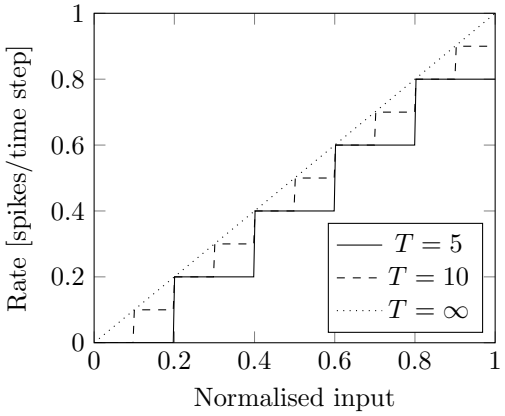
Figure 7: Example non-spiking and spiking neurons: $A_{i,j}$ are the weights between the neurons of the previous layer and the illustrated neurons, X_i are the un-encoded values and $S_{X,i}[t]$ are the encoded spikes of X, Y_j is the output of the non-spiking neuron and $S_{Y_j}[t]$ are the spikes representing Y_j.

2.2.1 Encoding and Normalisation

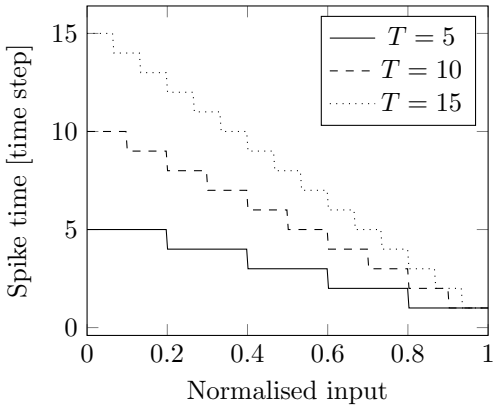
Since SNNs require the inputs to be binary, there needs to be an encoding process to generate the discrete spikes from the continuous inputs. Decoding is then used to extract the information from the output spikes [3]. There exists two types of such coding that are compared in Table 1. The following describes how they work in more detail.

Coding	Advantages	Disadvantages
Time-to-first-spike (TTFS)	• Fewer spikes. • Faster inference. • Better information disentangling due to fewer spikes.	• Only reaches high accuracies for simple tasks and shallow architectures. • More sensitive to noise since the redundancy is low.
Rate	• Higher performance/accuracies • More robust to noise since there is more redundancy due to the larger number of spikes.	• Requires more encoding steps containing a higher density of spikes which makes it more computationally intensive.

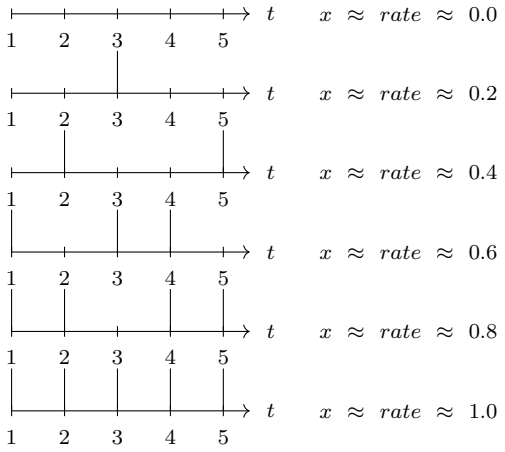
Table 1: Encoding comparison ([3], [4], [14], [15] and [16])



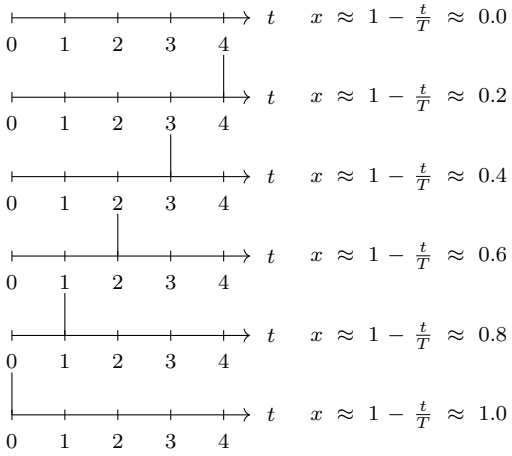
(a) Rate encoding of a normalised input for $T = 5$ and $T = 10$ steps, using an increasing number of steps approaches the non-encoded value. Using $T = \infty$ steps is equivalent to the non-encoded value.



(b) TTFS encoding of a normalised input for $T = 5$, $T = 10$ and $T = 15$ steps



(c) Rate decoding for a range of values when the normalised input had been encoded with 5 time steps, the decoded value is given as x.



(d) TTFS decoding for a range of values when the normalised input had been encoded with 5 time steps, the decoded value is given as x.

Figure 8: Rate and TTFS encoding and decoding comparative examples.

Rate Coding

Rate encoding translates the continuous input into the firing rate of an associated spike source according to Equation 9 [1] where T is the number of encoding steps: the higher the magnitude of the input X the higher the rate [3]. This rate defines the probability that there is a spike at each time step $t \in [0, T[$. For example, if $rate = 0$ then no spikes are fired, if $rate = 1$ then all T spikes are fired, if $rate = 0.5$ then $\frac{T}{2}$ spikes are fired, etc. To ensure that all values can be represented by the number of chosen steps, the input is first normalised to X_n using the maximum and minimum values of X: X_{max} and X_{min}. Figure 8a shows how the normalised input is encoded in the rate: the higher the number of time steps, the smaller the error between the encoded value and the input.

$$rate = \frac{\lfloor X_n T \rfloor}{T} \tag{9}$$

Decoding is achieved by taking the average number of spikes. If $S_x[t]$ is the encoded value at time t, then the normalised decoded signal X_n and its denormalised decoded signal X are shown in Equation 10. Figure 8c

[1] The floor operation $b = \lfloor a \rfloor$ transforms the number a to its integer value b by rounding it down.

illustrates how the decoded value is obtained from the rate when 5 time steps are used.

$$X_n = \frac{1}{T}\sum_{t=1}^{T} S_x[t] = \frac{X - X_{min}}{X_{max} - X_{min}}$$
$$X = X_{min} + (X_{max} - X_{min}) \cdot \frac{1}{T}\sum_{t=1}^{T} S_x[t] \tag{10}$$

Latency Coding

Temporal coding translates the analogue input X into a spike timing. The most common is time-to-first spike (TTFS) coding where higher values are typically mapped to shorter onset times from a given time reference, which is illustrated in Figure 8b. Other temporal codings include rank-order coding where the order of spikes from different neurons encodes information, and phase coding where an internal oscillatory signal provides a reference signal [3]. Equation 11 shows TTFS spike encoding where the analogue input X is first normalised to X_n: the higher the magnitude of the input X, the faster the spike appears.

$$time = T - \lfloor X_n T \rfloor \tag{11}$$

Decoding is achieved by the timing of the output spikes. Figure 8d illustrates how the decoded value is obtained from the spike time when 5 time steps are used.

2.2.2 Types of Neurons

The main structural difference between ANNs and SNNs is the type of neuron used. While an artificial neuron sums its weighted inputs and passes them through an activation function, an SNN neuron integrates its weighted input over time and saves it as a membrane potential. It then spikes once its potential reaches a certain threshold voltage U_{th}, which makes it work similarly to biological neurons.

A range of models exist to define those neurons. Figure 9 shows the main spiking neurons in order of biological realism. While realistic neurons are good at reproducing electrophysiological results, they are more complex and are hence much more difficult to implement in hardware [6].

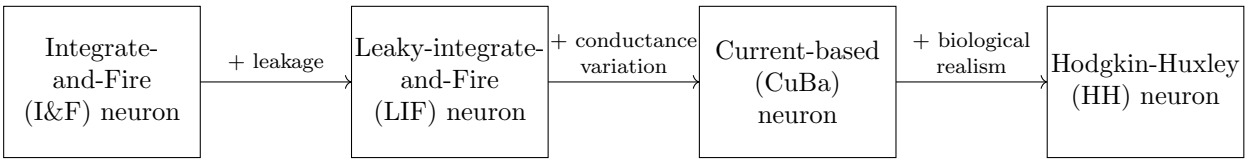


Figure 9: Spiking neurons from least complex to most complex.

The leaky integrate-and-fire (LIF) neuron is the standard choice: although much simpler than conductance-based models, it is computationally efficient and easy to train with gradient-based techniques. Similar accuracy levels can be reached by LIF-based SNNs compared to ANNs as later shown in the accuracies reached by ANNs and SNNs in Section 3.4 of the literature research.

Equation 12 describes the mechanism of a LIF neuron where β represents the leakage factor, the extreme cases being $\beta = 0$ with the maximum leakage without memory from time step to time step, and $\beta = 1$ without leakage and a fully preserved membrane potential between time steps. In this latter case, the LIF neuron is simplified to a an I&F neuron. The new membrane potential $U[t]'$ is given by the leaked potential of the previous step $\beta U[t-1]$ plus the input current $I[t]$. The input current $I[t]$ is the sum of the weights when their corresponding input neuron spikes. Then, if the membrane potential reaches a threshold U_{th} it emits a spike $S[t]$ [6].

$$U[t]' = \beta U[t-1] + I[t]$$
$$S[t] = \begin{cases} 1 \text{ if } U[t]' > U_{th} \\ 0 \text{ if } U[t]' < U_{th} \end{cases} \tag{12}$$

Once the spike has been emitted, there are two reset mechanisms shown in Equation 13. Soft reset subtracts the threshold voltage U_{th}, and hard reset sets the membrane potential back to zero. In general, soft reset leads

to higher performance [6], this is because there is no loss of information and the remaining membrane potential will be used to influence future spikes.

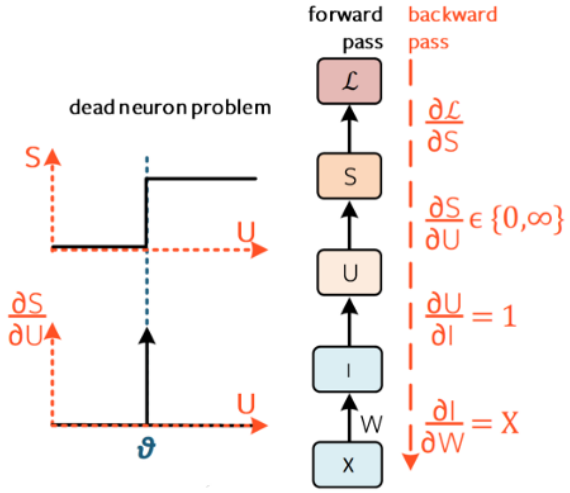
$$\text{Soft reset: } U[t] = U'[t] - U_{th} \cdot S[t]$$
$$\text{Hard reset: } U[t] = \begin{cases} 0 \text{ if } U[t]' > U_{th} \\ U'[t] \text{ if } U[t]' < U_{th} \end{cases} \tag{13}$$

2.2.3 Training

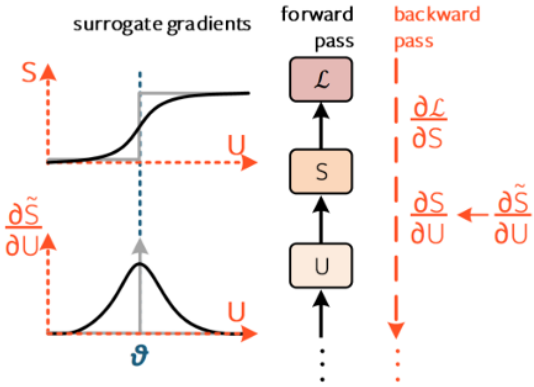
One main issue with SNNs is to find the best way to train their weights. Standard error back-propagation is not applicable to SNNs. Figure 10a illustrates the issue: the derivative of the spike with respect to the membrane potential $\delta S/\delta U$ is either 0 or ∞ which makes the gradient meaningless for weight update [6].

Three methods that counter this problem are highlighted in Table 2. Two of them train the network from scratch: unsupervised learning with spike-time dependent plasticity (STDP) and supervised learning using surrogate gradients during back-propagation through-time (BPTT). BPTT replaces the spike function by a surrogate function that smoothens the function $\delta S/\delta U$, an illustration is provided in Figure 10b. Secondly, an unsupervised bio-inspired alternative to BPTT is STDP, which enforces correlation between two neurons by increasing the weight of their connection for neurons that fire causally within a time window [6].

The last one trains an ANN first and then converts the weights to an equivalent SNN. This requires the activation function to be a ReLU and the SNN to follow a rate-based encoding, since the spiking action inherently cuts off all negative values. Since there is no training done on the SNN, this method leverages all advances found for traditional DNNs. It is especially meaningful to use it if the training efficiency is not important and that the focus is on reducing inference costs. However, the conversion process might not benefit from all advantages of SNNs since more steps are usually needed compared to network trained through BPTT [6], which is because the converted SNN might have benefited from different weights that would have been found by training the SNN for a lower number of encoding steps.



(a) Standard error back-propagation



(b) Surrogate gradients

Figure 10: Training a SNN, both figures are taken from [6]: $\mathcal{L}$ is the loss, S is the spike, $\tilde{S}$ is a surrogate spike function to be used during back-propagation, U is the membrane potential), I is the current, W is the weight

Training type	Biological plausibility	Achieves state-of-the-art accuracy	Ease of use	Stable	Scalable
STDP	High	No	No	No	No
BPTT with surrogate gradients	Low	Yes	Medium	Yes	Yes
ANN-SNN conversion	Low	Yes [2]	Yes	Medium	Medium

Table 2: SNN weight training comparison [9] [10]

[2] although it is now outperformed by BPTT with surrogate gradients.

2.3 Quantisation

According to [17], using quantisation is one of the most effective ways to reduce computational time and energy consumption of neural networks: for example, a simple reduction from 32 to 8 bits reduces the memory needs by $\times 4$ and the cost for multiplication by $\times 16$.

It is possible to directly train the neural networks with fewer bits, which is called quantisation-aware training. However, it is also possible to train the network with full-precision and then reduce the number of bits, which is called post-training quantisation [17].

Another point to consider is to choose which type of representation to use. Although floating-point numbers are able to represent a wider range of values with more precision than fixed-point numbers, floating-point arithmetic uses more energy for the same operation [17].

Asymmetric quantisation is expressed in Equation 14 [3]: x_{int} represents the integer version of x in terms of the scale factor s, the zero-point z and the bit-width b. The zero-point aims to map all numbers to positive values so that they can be represented with integers. The scaling factor s sets the step size of the quantiser, so it characterised the precision held in one bit of information. The approximated real-value is given by $\hat{x}$ which shows that quantising limits the values to $\hat{x} \in [-sz, s(2^b - 1 - z)]$. There is a trade-off between this clipping error and the rounding error caused by the scale factor s [17].

$$\begin{aligned} x_{int} &= \text{clamping}\left(\text{round}\left(\frac{x}{s}\right) + z; 0; 2^b - 1\right) \\ \hat{x} &= s(x_{int} - z) \end{aligned} \tag{14}$$

Symmetric quantisation is a special case of the previous quantisation method where $z = 0$. However, this prevents negative values from being represented with the previous method, it can hence be modified to incorporate signed integers as shown in Equation 15 [17].

$$\begin{aligned} x_{int} &= \begin{cases} \text{clamping}\left(\text{round}\left(\frac{x}{s}\right); 0; 2^b - 1\right) & \text{for unsigned integers} \\ \text{clamping}\left(\text{round}\left(\frac{x}{s}\right); -2^{b-1}; 2^{b-1} - 1\right) & \text{for signed integers} \end{cases} \\ \hat{x} &= s \cdot x_{int} \end{aligned} \tag{15}$$

Another special case is power-of-two quantisation which is when the scale factor s is restricted to a power of 2. In that case operations can be implemented as bit shifting which are more efficient in hardware. The drawback is that it is harder to optimise for the error between the un-quantised and quantised value [17].

The choice on the quantisation type depends on the data type. If the output can only be positive such as at the output of a ReLU layer, then unsigned symmetric quantisation is meaningful. If the distribution is centered around zero, then it is better to use signed symmetric quantisation.

Those quantisation schemes can be applied to a neural network with different granularity where different sets of quantisation parameters can be defined for different parts of the network. Per-tensor quantisation has one set of parameters for the weights and another for the activations and per-channel quantisation contains the two sets of parameters for each channel. Per-channel quantisation is especially useful if the magnitude of the weights varies from channel to channel. However, not all hardware is designed to be able to incorporate different quantisation parameters.

[3] The notation $clamping(a; b; c)$ clamps the value a between b and c.

3. Literature Review

This chapter is a literature review to summarise what has been achieved so far to transform the radar-processing pipeline for gesture recognition into its spiking equivalent. Figure 11 illustrates the different steps in the radar-processing pipeline and states which section explains which step. Section 3.5 reviews these findings to identify the gaps and paths worth exploring to address such gaps.

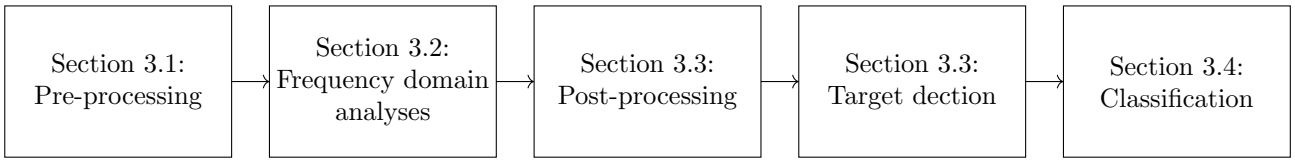


Figure 11: Gesture recognition radar pipeline stating how the sections of the literature review are organised.

3.1 Pre-processing Steps

Once the low-passed-filtered IF signal (Equation 4) is received, it needs to be pre-processed before the FTs are applied. Table 3 shows a series of strategies that are used in common radar processing pipelines in the literature. As shown by [4] which also implements a full spiking hand-gesture recongition pipeline, the effects of pre-processing are much more significant for SNNs than they are on ANNs of the same topology in terms of reaching higher classification accuracy.

Pre-processing type	Explanation	Benefits	Costs	Source
Delta filter	Subtracting each chirp from the one preceding it to remove the signal appearing in both.	Removes static objects since the DC components cancel each other out.	This can increase AC noise.	[12] [4]
Moving target indication (MTI) filtering	Substracting from each chirp a running average of the previous chirps to remove constant signal.	Also removes static objects but with averaging over more chirps to obtain a more accurate DC baseline.	More parameters needs to be defined such as the forget factor of the running average.	[12] [18]
Level crossing/thresholding	Thresholding to fixed values of -1 below a certain value, 1 above another and 0 in between.	The output is binary which saves on memory and removes noise while maintaining all frequency spectrum information.	Loses information but this loss does not impact the classification accuracy significantly.	[12]
Hann windowing	Multiplication by a factor to minimise the effects of the non-idealities of a finite FFT over an infinite one.	Outputs a more ideal FT with reduced side lobe levels.	Additional computations (except if the FT coefficients are pre multiplied by the Hann coefficients)	[4] [12]

Table 3: Pre-processing steps comparison

Filtering and level-crossing are easily implemented in neuromorphic hardware since they only need subtraction and comparison operations. Hann windowing adds a correction factor to the FT, the FT weights can be

multiplied by those factors with no added computations during inference. For coherent pulse integration, there is a division due to the averaging, however it could be transformed by a simple sum to avoid the multiplication.

3.2 Frequency-domain Analyses

The next step is to feed the pre-processed signals into the FTs to extract the distance and velocity components of the gesture contained in the signal. The main ways to do so are RD maps and μDoppler signatures.

In both cases, the first FT is done along the sample dimension to extract the range as has been explained in Sub-section 2.1.2. The second FT is applied along the chirps for both methods but in a different manner. To construct a RD map, this FT is applied along consecutive chirps. However, to construct a μDoppler signature, the FT is applied along the chirps in consecutive frames where a frame is a fixed number of chirps [4]. The choice greatly depends on the dataset type, for example, if most gestures occur repeatedly in consecutive frames, and that the aim is to detect this periodic motion, then using μDoppler signatures is more meaningful. [5]. Examples of usages of both methods are provided in Table 4.

Algorithm type	Explanation	Training type	Source
μDoppler signature	Distance FT along the sample dimension followed by another FT on the frame dimension	Mathematically defined for an SNN using I&F neurons and rate encoding	[4]
RD map	Distance FT along the sample dimension followed by a Doppler FT on the chirp dimension		[4] [19]
		Mathematically defined for an SNN using I&F neurons and latency encoding, using a spiking FT	[20] [20]
		Resonate and Fire neurons where each neuron spike contains information about their eigenfrequency, amplitude and phase	[3]

Table 4: Frequency-domain analyses comparison

3.3 Post-processing

A certain number of papers have added techniques that are implemented before the RD map is fed into the classification network. Table 5 highlights two of them.

Processing type	Explanation	Source
Doppler axis cropping.	Keeps only the portion of the Doppler axis whose corresponding velocity is within possible values for the target object.	[4][12]
Normalisation of the absolute value of the RD map between 0 and 1, they can be normalised one at a time by taking each RD map's maximum value, or the full dataset can be normalised at once by taking its maximum value.	Makes the maps more comparable between each other. While this is optional for an ANN implementation, it is mandatory for an SNN as explained in Section 2.2.1	[4]

Table 5: Further processing comparisons

Another step that can be included between the FT and the classification are target detection algorithms. These are used to identify the cells or pixels containing target signatures and reject those containing noise or clutter.

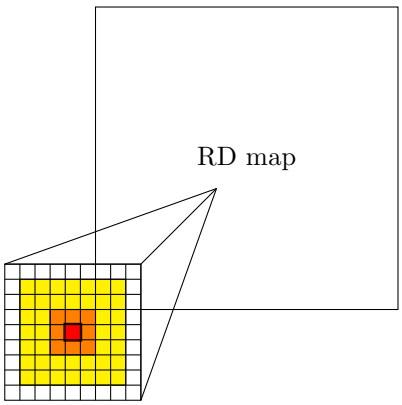
The most basic method is to simply apply a detection constant threshold on the data defined by a constant number, such as the weighted mean [18], zero [5] or the k^{th} largest value [4]. This works well if the data is homogeneous and the noise value is constant on the full RD map. It is also a computationally efficient method since there are no addition/multiplication operations, there are only a number of comparison operations equal to the number of pixels.

Constant false alarm rate (CFAR) algorithms are more complex since the threshold depends on the immediate surroundings of the pixel being thresholded. Figure 12a shows that for each pixel under test (red), $N_{guarding}$ cells (orange) are ignored and the following $N_{neighbouring}$ cells (yellow) are taken for comparison. Cell-averaging constant false alaram rate (CA-CFAR) takes the average of those neighbouring cells as the threshold, whereas ordered statistics constant false alarm rate (OS-CFAR) takes their k^{th} largest value where k has to be found [3]. Figure 12b illustrates a possible output for a CFAR algorithm given the provided RD map.

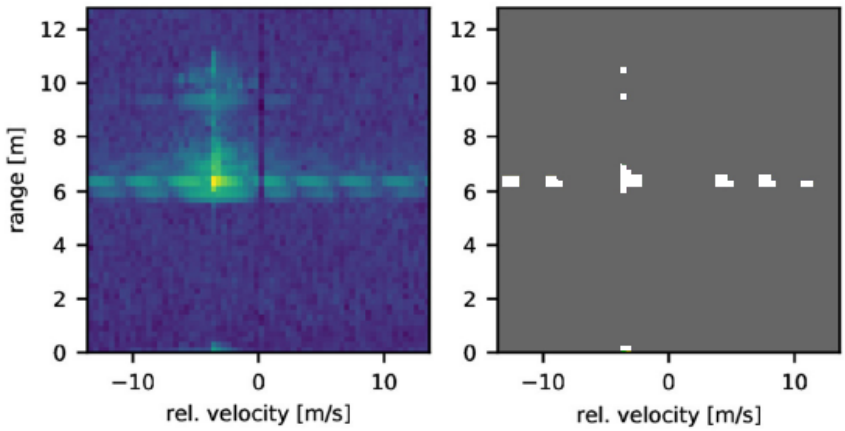
Table 6 shows examples where these target detection techniques are implemented in their spiking form, both of which are latency encoded. To the best of the author's knowledge, no rate encoded version has been found in the literature. The study in [3] shows that both spiking CFAR algorithms can reach near-perfect performance compared to their non-spiking version when enough time steps are used. These spiking version are, however, much more computationally intensive than their non-spiking equivalents as illustrated by the second column.

Algorithm type	Number of operations per output pixel	Performance
CA-CFAR	$N_{cfar} + T$ 'add' operations and T 'compare' operations (N_{cfar} 'add' operations and 1 'compare' operations for the non-spiking version)	Quickly rising sensitivity and precision (with number of time steps) $\rightarrow$ 99% sensitivity and 99% precision for 500 time steps.
OS-CFAR	N_{cfar} 'add' operations and T 'compare' operations (0 'add' operations and $T + 1$ 'compare' operations for the non-spiking version)	Perfect precision even for few time steps, worse sensitivity than the CA-CFAR $\rightarrow$ 100% precision even for few time steps and 99% sensitivity for 100 time steps (with logarithmic conversion, 95% sensitivity for 800 time steps otherwise)
		Able to detect all three targets with $N_{guarding} = 48$, $N_{neighbouring} = 176$, $k = 9$ and a scale factor $\alpha = 0.2$

Table 6: Target detection comparison, N_{cfar} is the number of cells taken for the comparison ($N_{cfar} = (N_{neighbouring} + N_{guarding})^2 - N_{guarding}^2$) and T is the number of encoding steps [3].



(a) Schematic of a CFAR algorithm, the red cell is the one being thresholded, the orange cells are the guarding cells, the yellow cells are the neighbouring cells and the white cells are not considered for thresholding. In this specific example $N_{guarding} = 1$ and $N_{neighbouring} = 2$.



(b) Example of a CFAR algorithm where the left figure represented the RD map while the right figure is the RD map after being thresholded. This is a modified version of a figure taken from [3].

Figure 12: Illustration of a CFAR using a graphical explanation and an example with a real RD map.

3.4 Classification

Classification networks can be implemented in many ways. For $2D$ radar images, it is meaningful to at least use a convolutional layer to extract $2D$ features, and it will also at least require a fully-connected layer for the output. Table 7 shows the results from a certain number of classification architecture where the data is rate encoded. Table 8 does the same but for data that is latency encoded. The following provides a short explanation for the datsets used in those tables:

- CARRADA: automotive dataset containing RD maps recorded by a $77GHz$ radar. Its drawback is that it is limited in size, complexity and variety since it is recorded with low environmental noise [3],
- Soli: RD maps obtained from a $60GHz$ radar containing 11 different hand gestures performed by 10 different people, each person repeated each gesture 25 different times to reach a total of 2 750 samples [5],
- Dop-NET: μDoppler signatures containing 4 different gestures performed by 6 different people at about 30cm from the radar,
- PSCAL VOC: visual image with annotation around the target, it contains 20 object categories for a total of about 3 000 images [21],
- MS COCO: visual images with annotations abou the type of target, it contains 164 000 images [22],
- Iris: 150 images of iris flowers containing three iris species,
- MNIST: handwritten digits with 70 000 samples [23].

Algorithm type	Training type	Indicative performance	Computational efficiency	Dataset	Source
Extract the regions that have detected objects and feed them into an SNN with two convolutional layers, one fully-connected layer and one output layer.	BPTT with surrogate gradients	90% on the SNN version compared o 94% on the ANN version	$4k$ SNN synaptic events compared to $1M$ ANN MAC operations	CARRADA	[3]
Feed the RD/Doppler signatures into an SNN with IF neurons that has one convolutional layer, one max-pooling layer and 2 fully connected layers.	BPTT with Gaussian surrogate gradient			Custom arm gestures at 2m of the 8Ghz radar	[4]
Liquid state machine + simple classifier (logistic regression, random forest, SVM)	Random weights for the LSM neurons and training for the readout neurons with cross-validation	98.8% with SVM classifier	460 neurons (but the accuracy can be nearly as good even with about 150 neurons)	Soli and Dopl-NET	[5]
Feed the RD maps into one convolution layer, one fully connected layer and one output layer, all with LIF neurons	BPTT with surrogate gradients (SoftLIF activation)	99.50% compared to $86.25 - 99.63\%$ for equivalent ANNs	$75kB$ for compared to $375k - 12MB$ for equivalent ANNs	Custom hand gestures collected from 5 different people for a total of 4 800 gestures.	[18]
Spiking-YOLO: spiking version of Tiny YOLO	ANN-SNN conversion	51.83% compared to a target 53.01% 25.66% compared to a target 26.24%	2000 times less energy than its non spiking equivalent	PASCAL VOC and MS COCO	[9]

Table 7: Spiking classification comparison - rate encoded

Algorithm type	Training type	Indicative performance	Computational efficiency	Dataset	Source
Feed the latency-encoded inputs into a network with one input, one hidden and one output layer using a sTTFS (synchronous TTFS) encoding	Based on the equivalent ANN	98% accuracy	$\approx$ 40% of the area and 55% of the power of the equivalent ANN, but $15-18$ times higher energy due to higher latency	Iris	[16]
Feed the latency-encoded inputs into a network with one input, one hidden and one output layer using a TTFS encoding	BPTT	96% accuracy	$\approx$ 40% of the area and 50% of the power of the equivalent ANN, but $\approx$ 5 times higher energy due to higher latency		
Feed the latency-encoded inputs into two fully connected layers	BPTT	96.90% accuracy	3.5 times more power efficient than the rate-encoded equivalent	MNIST	[24]

Table 8: Spiking classification comparison - latency encoded

The previous tables provide indicative performances that give information about the order of magnitude of how well the papers' algorithm behaves for their given dataset. It is not possible to perform a direct comparison between them since they do not classify data from the same distributions, but the indicative performance helps to make informed guesses about the type of architectures that could be used in this work. From this literature search, it can be concluded that rate-encoded networks are more common and are more easily implemented that their latency-encoded versions. Furthermore, using an ANN-SNN conversion appears to be the easiest way to implement the spiking implementation, as it can make use of all advantages of the training with a non-spiking network.

3.5 Gaps and Solutions

From the literature study presented in this chapter, the following challenges are extracted:

- the papers present parts of the pipeline, which means that some outline rate-encoded methods, others show latency-encoded ones, but the impact of the loss from one to the next is not systematically investigated in a step by step manner on the same dataset. Furthermore, the data needs to be normalised before being encoded, however, it might not be necessary to be able to represent all values, especially larger ones. There might be normalisation methods that are better than others, but there is no comprehensive description of this process,
- there is very little explanation about how to tune the spiking versions of the algorithms, in particular this includes the thresholds and the number of encoding steps,
- many papers emphasise the advantage of using temporal encoding to become sparser by using fewer spikes, and hence saving on the cost of computations. However, few implement algorithms using it since rate encoding provides better results,
- the comparison between the different model's computational efficiencies is complicated due to different metrics, for example: the number of encoding steps, the memory requirements, the number of neurons regardless of the other memory needs, the energy, the area, the power.

Based on the above open challenges, the work outlined in the following chapter of this thesis focuses on developing a strategy to optimise the conversion of the non-spiking pipeline to its spiking equivalent from start to end. It aims to define the trade-off between the number of encoding steps, the memory requirements and the resulting accuracy by:

- ensuring that the spiking accuracy resembles the non-spiking one, including by selecting thresholds resulting in a good ANN-SNN correspondence.
- reducing the memory footprint, for example by implementing quantisation strategies,
- attempt to design both the ANN and SNN hardware.

4. Methodology

This chapter describes the theoretical steps to be taken to reach a pipeline that balances the trade-off between the accuracy of the gesture classification and the resources needed to reach this accuracy. The aim of this chapter is to develop the mathematics behind the equations referenced in Chapter 5 when these mathematics are applied to a real dataset.

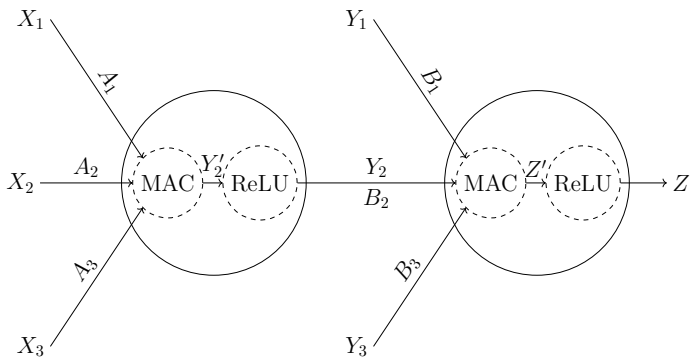
Section 4.1 extends the theory from Chapter 2 to explain the mathematical bases for the conversion between any non-spiking network to its spiking equivalent. Section 4.2 explains how to implement a spiking FT. Finally, Section 4.3 describes how to merge a batch normalisation layer with its preceding layer.

4.1 Spiking Neural Networks

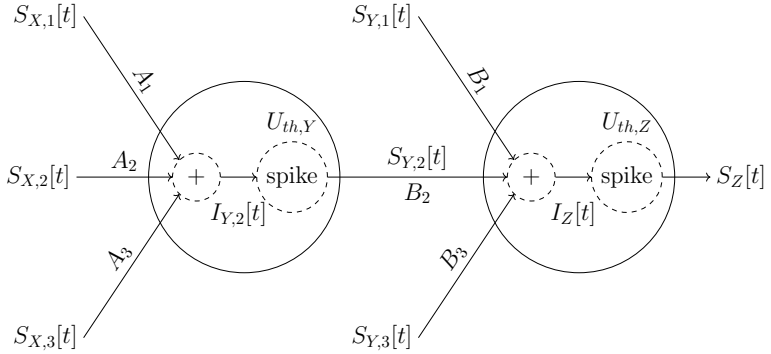
Section 2.2.2 explained that the LIF neuron is usually used which adds an additional parameter: the leak term. This work uses the simplified I&F neuron since it is the spiking neuron with the fewer hyper-parameters to find, and because it results in an easy ANN-SNN conversion as is explained in this section [25].

In non-spiking ANNs, the only parameters to optimise during training are the weights. However, in SNNs using I&F neurons, there are two more parameters to find: the thresholds and the number of encoding steps. When implementing an ANN-SNN conversion, the weights are already decoupled from the spiking parameters since they are simply copied from the independent ANN. However, the thresholds, the number of encoding steps, and the quantisation scheme should also be decoupled from each other so they can be selected independently from each other.

Sub-section 4.1.1 develops a mathematical relationship of the non-spiking output in terms of the spikes of the equivalent spiking neuron. From there, Sub-section 4.1.2 explains how to find good thresholds. Sub-section 4.1.3 then describes how to estimate the performance of a spiking neural network with different numbers of encoding steps.



(a) Non-spiking neurons using a ReLU activation function: X_i represent the inputs to the first layer, Y_j the outputs of this first layer fed to the second layer and Z is the output of the second layer, Y'_j and Z' represent the values between the MAC operation of the neuron and the ReLU activation function.



(b) Spiking neurons: $S_{X,i}[t]/S_{Y,j}[t]/S_Z[t]$ represents the encoded version of $X_i/Y_j/Z$ at time t of the encoded spike train, the current added at time t to the membrane potential is provided as $I_{Y,j}[t]/I_Z[t]$, and the neuron spikes when its membrane potential reached $U_{th,Y}/U_{th,z}$.

Figure 13: Spiking/non-spiking neurons equivalence: each neuron represents one layer where the weights A_i are those of the first layer and the weights B_j are those of the second layer

4.1.1 Neurons

Figure 13a shows two non-spiking neurons using a ReLU activation function and weights A_i/B_j for the first/second layer. The analogue inputs are X_i where i iterates over all values incoming the neuron, the outputs of the first layer are Y_j where j iterates over the inputs of the neurons of the second layer, and the output of the second layer is Z. Those are expressed in Equations 16 and 17. It is assumed that the non-spiking neurons have no biases: as will be described in Section 4.2, the FT implementation does not require biases, and the non-spiking classification network is trained without any.

$$Y_j = ReLU(Y'_j) = ReLU\left(\sum_i X_i A_{i,j}\right) \tag{16}$$

$$Z = ReLU(Z') = ReLU\left(\sum_j Y_j B_j\right) \tag{17}$$

Figure 13b shows two spiking neurons. The following steps provide an equivalence between the spiking neuron outputs $S_{Y,j}[t]/S_Z[t]$ and the non-spiking ones Y_j/Z for the two layers.

1. **Spiking currents in terms of the incoming spikes**: The current $I_{Y,j}[t]$ modifies the membrane potential of the neuron j of the first layer at time step t. $I_Z[t]$ does the same for the neuron of the second layer.

   These currents are expressed in Equations 18 and 19 as the MAC operations of the incoming spikes $S_{X,i}[t]/S_{Y,j}[t]$ with their respective weights $A_{i,j}/B_j$. Since the spikes are either worth 0 or 1, in practice, this is equivalent to adding the weights when the corresponding spiking input is a 1. While the non-spiking networks do not have biases, spiking biases are necessary for the spiking networks to be equivalent. These biases are expressed as a_i for the first layer and b for the second layer, their values are found in step 2.

   $$I_{Y,j}[t] = \sum_i S_{X,i}[t] A_{i,j} + a_j \tag{18}$$

   $$I_Z[t] = \sum_j S_{Y,j}[t] B_j + b \tag{19}$$

2. **Pre-ReLU non-spiking output in terms of the sum of the spiking currents**: For the first layer, the current from Equation 18 is summed through time to provide an equivalence between the current sum $\sum_t I_{Y,j}[t]$ and the non-spiking pre-ReLU value Y'_j, the result is provided in Equation 20. It is assumed that the spiking values are rate encoded which allows Equation 10 to be used in the simplification. For simplicity, the upper and lower bounds of the sum through time t are omitted, but they always represent the sum from the first time step $t = 1$ to the final time step $t = T$.

$$\begin{aligned}
\sum_t I_{Y,j}[t] &= \sum_t \underbrace{\left(\sum_i S_{X,i}[t] A_{i,j} + a_j\right)}_{\text{Equation 18}} \\
&= \left[\sum_i \left(\sum_t S_{X,i}[t]\right) A_{i,j}\right] + T \cdot a_j \\
&= \left[\sum_i \left(\underbrace{T \cdot \frac{X_i - X_{min}}{X_{max} - X_{min}}}_{\text{by Equation 10}}\right) A_{i,j}\right] + T \cdot a_j \\
&= \frac{T}{X_{max} - X_{min}} \cdot \underbrace{\sum_{n_x} X_{n_x} A_{i,j}}_{\text{Equation 16}} - T \cdot \underbrace{\frac{X_{min}}{X_{max} - X_{min}} \cdot \sum_i A_{i,j}}_{a_j} + T \cdot a_j \\
&= \frac{T}{X_{max} - X_{min}} \cdot Y'_j \\
Y'_j &= (X_{max} - X_{min}) \frac{1}{T} \sum_t I_{Y,j}[t]
\end{aligned} \tag{20}$$

A similar calculation can be applied to the second layer to obtain the relation in Equation 21 between the spiking current through time $\sum_t I_Z[t]$ and the pre-ReLU non-spiking Z'.

$$Z' = (Y_{max} - Y_{min}) \frac{1}{T} \sum_t I_Z[t] \quad \text{with} \quad b = \frac{Y_{min}}{Y_{max} - Y_{min}} \cdot \sum_{n_y} B_j \tag{21}$$

In the case where $Y_{min} = 0$, then Equation 21 is simplified as shown in Equation 22. This hypothesis is satisfied for all layers that contain a ReLU in the non-spiking form, which is the case here.

$$Z' = \frac{Y_{max}}{T} \sum_t I_Z[t] \text{ with } b = 0 \tag{22}$$

3. **Sum of spikes in terms of the sum of spiking currents**: Equations 20 and 22 provided the non-spiking pre-ReLU output in terms of the sum of the currents, the aim is now to replace the sum of the currents wit the sum of the spikes. Equation 23 provides the relationship between the sum of currents and the sum of spikes for any spiking neuron. This uses Equation 12 that provided the equation for a LIF neuron (β is set to 1 for an I&F neuron) and Equation 13 that provided the reset mechanisms (soft reset is used here).

   This general equation assumes a total number of time steps T, a threshold voltage of U_{th}, a membrane potential at time $t \in]0, T]$ of $U[t]$, and a spike value at time $t \in]0, T]$ of $S[t]$.

$$\begin{aligned}
\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} I[t] &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \underbrace{\Big(U[t] - U[t-1] + U_{th} \cdot S[t] \Big)}_{\text{by Equations 1213}} \\
&= \underbrace{\lim_{T\to\infty} \frac{\overbrace{U[T] - U[0]}^{\text{finite}}}{T}}_{=0} + \lim_{T\to\infty} \frac{U_{th}}{T} \cdot \sum_{t=1}^{T} S[t] = \lim_{T\to\infty} \frac{U_{th}}{T} \cdot \sum_{t=1}^{T} S[t] \\
\frac{1}{T} \sum_{t=1}^{T} I[t] &\approx \frac{U_{th}}{T} \cdot \sum_{t=1}^{T} S[t]
\end{aligned} \tag{23}$$

   Adapting the general Equation 23 to the example spiking neurons from Figure 13b provides the relationships in Equation 24 and 25 where $U_{th,Y}$ is the threshold voltage of the first layer and $U_{th,Z}$ the threshold voltage of the second layer.

$$\sum_t S_{Y,j}[t] = \frac{1}{U_{th,Y}} \sum_t I_{Y,j}[t] \tag{24}$$

$$\sum_t S_Z[t] = \frac{1}{U_{th,Z}} \sum_t I_Z[t] \tag{25}$$

4. **Non-spiking output in terms of the sum of the spikes**: Equation 26 incorporates Equation 24 into Equation 20 to obtain a relationship between the non-spiking output of the first layer and the output of the spiking neuron. The ReLU is automatically included since the spiking neuron will only spike for positive values if its threshold is positive. The same is achieved for the second layer by incorporating Equation 25 into Equation 22.

$$Y_i = \frac{U_{th,Y} \cdot (X_{max} - X_{min})}{T} \sum_t S_{Y,i}[t] \tag{26}$$

$$Z = \frac{U_{th,Z} \cdot Y_{max}}{T} \sum_t S_Z[t] \tag{27}$$

4.1.2 Thresholds

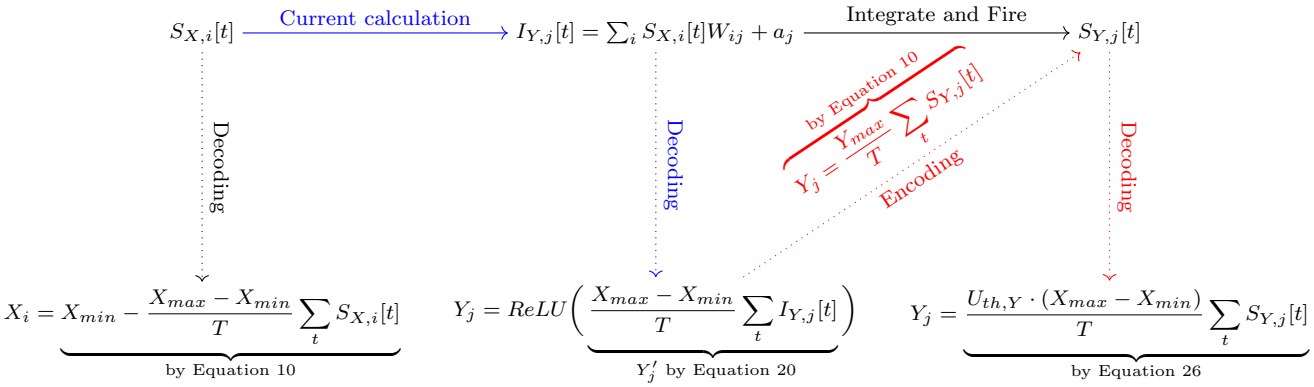


Figure 14: Diagram showing the relations between the different steps of an I&F spiking neuron and its non-spiking equivalent. The top part represents the spiking operations, and the bottom part represents the non-spiking equivalents that can be obtained through the vertical dotted "Decoding" lines. The "Encoding" line shows how the non-spiking Y_j can be rate encoded to the spikes $S_{Y,j}[t]$.

The previous section describes the spiking equivalence of a two-layer non-spiking network. However, the thresholds are not yet defined. Figure 14 illustrates how to find the best threshold value for the first layer. The top of the diagram with solid lines represents the spiking network. The current $I_{Y,j}[t]$ is calculated from the inputs spikes $S_{X,i}[t]$ by Equation 18. Then this current is added to the membrane potential, and after the integrate and fire actions of the I&F neuron, the spikes $S_{Y,j[t]}$ are emitted.

The black vertical dotted line provides the equation to decode the spiking inputs $S_{X,i}$ into their non-spiking values X_i. The blue vertical dotted line extracts the non-spiking output Y_j from the current $I_{Y,j}[t]$. The red vertical dotted line also calculates the non-spiking output Y_j but using the spikes $S_{Y,j}$. The red diagonal dotted line shows how to encode the values Y_j if they were encoded from the non-spiking values (as a reminder, it is hypothesised that $Y_{min} = 0$).

In the perfect case, the relationship between the non-spiking Y_j and its spikes $S_{Y,j}[t]$ should be the same for its encoding from its non-spiking values to its spikes (across the red diagonal dotted line), and for its decoding from its spikes $S_{Y,j}[t]$ to its non-spiking values Y_j (across the red vertical dotted line). This equivalence is stated in Equation 28 to extract the corresponding threshold voltage.

$$Y_j = \frac{Y_{max}}{T} \sum_t S_{Y,j}[t] \Leftrightarrow Y_j = \frac{U_{th,Y} \cdot (X_{max} - X_{min})}{T} \sum_t S_{Y,j}[t]$$
$$U_{th,Y} = \frac{Y_{max}}{X_{max} - X_{min}} \tag{28}$$

An equivalent diagram for all following layers is given in Figure 46 in Appendix 7.2, and the same logic can be applied to find any layer's threshold. In the case of the second layer's whose output is Z_k, the equivalence and thresholds are provided in Equation 29.

$$Z_k = \frac{Z_{max}}{T} \sum_t S_{Z,k}[t] \Leftrightarrow Z_k = \frac{U_{th,Z} \cdot Y_{max}}{T} \sum_t S_{Z,k}[t]$$
$$U_{th,Z} = \frac{Z_{max}}{Y_{max}} \tag{29}$$

More generally, the general form for the n^{th} threshold is provided in Equation 30.

$$U_{th,1} = \frac{\max(\text{output layer } 1)}{\max(\text{input}) \text{ - } \min(\text{input})}$$
$$U_{th,n} = \frac{\max(\text{output layer } n)}{\max(\text{output layer } n-1)} \tag{30}$$

4.1.3 Pseudo-spiking Equivalent

A pseudo-spiking model of the network is defined as an intermediary version between the non-spiking and the spiking networks. In Figure 14, the pseudo-spiking equivalent is expressed by the blue path: it maintains the ReLU activation function, but it iterates over steps without spiking. An example is given below in the case of a convolutional layer:

- non-spiking in Algorithm 1: the output `y` is simply the `ReLU` of the convolutional layer applied to the input `x`,
- spiking in Algorithm 2:
  - the input `x` is first rate encoded with `T` time steps to generate the spike train `encoded_x`,
  - this spike train is sent one time step at a time through the convolution layer to output the current `t_current`,
  - the current `t_current` is fed into the function `lif` to update the membrane potential and output the spikes `t_spikes`,
  - the output `y` is decoded as shown by Equation 26,
- pseudo-spiking in Algorithm 3:
  - the input `x` is first rate encoded in the same way as the spiking case,
  - the spike train is also sent one time steps at a time through the convolution layer to output the current `t_current`,
  - instead of feeding the current `t_current` into the lif function, it is directly added to `pseudo_spikes`,
  - the pseudo spikes are decoded by using the equation from Figure 14.

Algorithm 1 Non-spiking

```
y = ReLU(conv(x))
```

Algorithm 2 Spiking

```
encoded_x = rate_encoding(x, steps=T)
pseudo_spikes = []

for t in range(steps):
    t_current = conv(encoded_x[t])
    pseudo_spikes.append(t_current)

sum_pspikes = sum(pseudo_spikes)
decoded_x = (X_max - X_min) / T
            * sum_pspikes
y = ReLU(decoded_x)
```

Algorithm 3 Pseudo-spiking

```
encoded_x = rate_encoding(x, steps=T)
spikes = []

for t in range(steps):
    t_current = conv(encoded_x[t])
    t_spikes, mem = lif(t_current, mem)
spikes.append(t_spikes)

sum_spikes = sum(spikes)
y = Uth * (X_max - X_min) / T * sum_spikes
```

Since this pseudo-spiking equivalent contains the steps but not the thresholds, it models the loss of information obtained for fewer steps. By analysing how the performance improves with increasing number of encoding steps, an optimal number of steps can be chosen so that it performs well with reasonable latency.

4.2 Fourier Transforms

The following explains how to implement a spiking FT as developped by [19]. The discrete FT $R[k]$ of a signal $r[n]$ is given in Equation 31 where k is the frequency bin index, n is the time bin index, N is the total number of bins, and j is the imaginary number.

$$\begin{aligned} R[k] &= \sum_{n=0}^{N} r[n] e^{-2\pi j \frac{k \cdot n}{N}} \\ &= \sum_{n=0}^{N} r[n] \cos\left(-2\pi \frac{k \cdot n}{N}\right) \\ &\quad + j \cdot \sum_{n=0}^{N} r[n] \sin\left(-2\pi \frac{k \cdot n}{N}\right) \end{aligned} \tag{31}$$

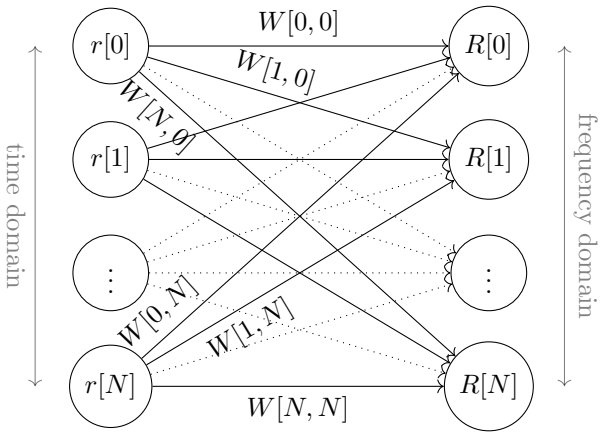


Figure 15: Neural network for the FT. The input neurons are the sampled inputs of the signal expressed through time $r[n] \ \forall \ n \in [0, N]$. The output neurons represent the frequency components of the signal $R[k] \ \forall \ k \in [0, N]$ where N is the number of input samples. The weights $W[k, n]$ are defined in Equation 32: the network represents the real part of the FT if the weights are cosines ($W[k, n] = W_{\Re}[k, n]$), and it represents the imaginary part if the weights are sines ($W[k, n] = W_{\Im}[k, n]$).

The FT is given as the sum of its real and imaginary parts. For both parts, the FT is equivalent to a weighted sum of the samples of the signal r: the weights are cosines for the real part and sines for the imaginary part. Figure 15 exemplifies how such a weighted sum is equivalent to a fully-connected layer in an ANN where the weights are pre-defined and not learned, which is better expressed in Equation 32.

$$\begin{aligned}
\Re[R[k]] &= \sum_{n=0}^{N_s} r[n] W_{\Re}[k,n] \\
\Im[R[k]] &= \sum_{n=0}^{N_s} r[n] W_{\Im}[k,n] \\
W_{\Im}[k,n] &= \sin\left(-2\pi \frac{k \cdot n}{N_s}\right) \\
W_{\Re}[k,n] &= \cos\left(-2\pi \frac{k \cdot n}{N_s}\right)
\end{aligned} \tag{32}$$

The input chirp-sample map is called X, the first FT outputs a Range map Y, which is expressed in a matrix form in Equation 33. To obtain the Range-Doppler map Z from the Range map Y, the Range map first needs to be transposed so that the FT is applied along the chirps rather than along the samples. Then it can be fed back into a similar fully-connected layer that contains its corresponding weights, this is highlighted in Equation 34. Figure 17 shows a graphical representation of these equations with the sizes of each matrix when the chirp-sample map has a size $N_c \times N_s$ (N_c is the number of chirps and N_s is the number of samples per chirp).

$$\begin{bmatrix} \Re(Y) & \Im(Y) \end{bmatrix} = X \begin{bmatrix} W_{\Re} & -W_{\Im} \end{bmatrix} \tag{33}$$

$$\begin{bmatrix} \Re(Z) & \Im(Z) \end{bmatrix} = \begin{bmatrix} \Re(Y)^T & \Im(Y)^T \end{bmatrix} \begin{bmatrix} W_{\Re} & W_{\Im} \\ -W_{\Im} & W_{\Re} \end{bmatrix} \tag{34}$$

4.2.1 Spiking Fourier Transform

To transform the FTs into their spiking version, each neuron of the fully-connected layer becomes an I&F neuron. However, since a neuron can only spike or not spike, it cannot represent both positive and negative values. As a consequence, the number of neurons needs to be duplicated so that, for a single value, there is a neuron to represent positive values, and another to represent negative values. An example for the Range FT is shown in Figure 16.

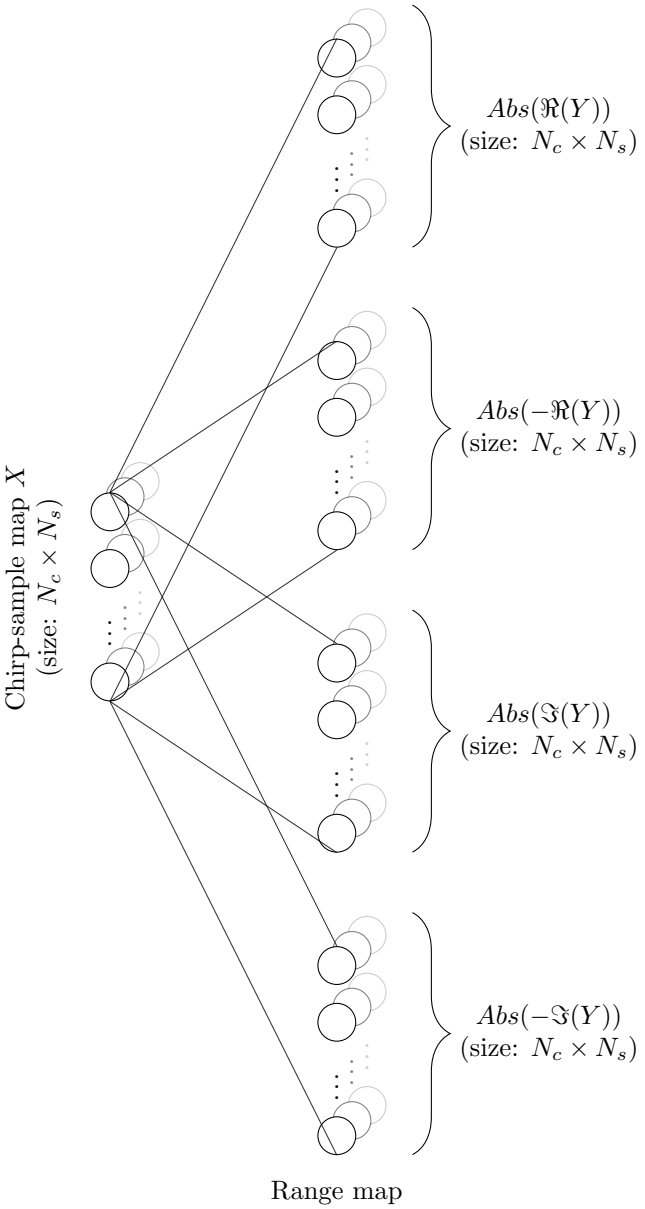
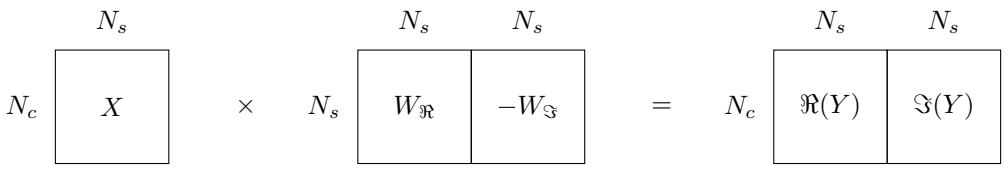


Figure 16: SNN for the Range FT, there are 4 convolutional layers to convert the chirp-sample map to its frequency components for the real-positive $Abs(\Re(Y))$, real-negative $Abs(-\Re(Y))$, imaginary-positive $Abs(\Im(Y))$ and imaginary-negative $Abs(-\Im(Y))$ values. $Abs(x)$ represents the absolute value of x. The lines represent the multiplication by the matrix $\begin{bmatrix} W_{\Re} & W_{\Im} \end{bmatrix}$ as shown in Equation 33: $Abs(\Re(Y))$ and $Abs(-\Re(Y))$ use the same weights $W_{\Re}$ and $Abs(\Im(Y))$ and $Abs(-\Im(Y))$ use the same weights $W_{\Im}$.

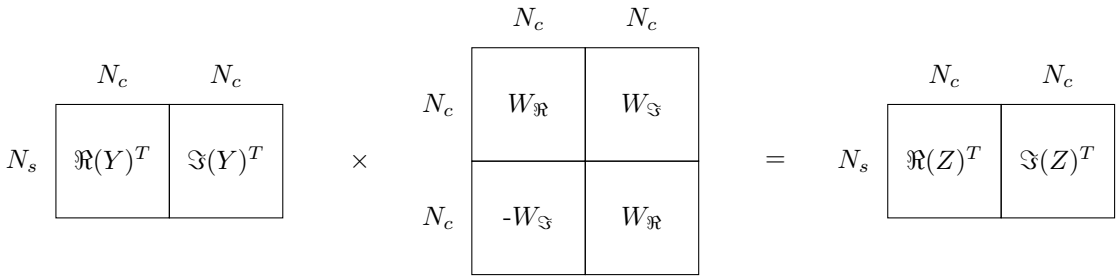
For the spiking version of the Range and Doppler Fourier transforms, the following claims can be made:

- since there is no learning as the weights are known, the precision loss between a spiking and a non-spiking RD map is only dependent on the number of time steps used,
- the thresholds of the I&F neurons are the only tweakable parameters,

- a 'substract' reset mechanism is used so that the integrate step of the I&F neuron has no losses. This means the sum of all integrated membrane voltages pre-spike is exactly equal to the true output value, hence the number of spikes is directly proportional to the true output value (assuming that the last time step's remaining membrane potential is negligible),
- normalising the input between 0 and 1 to rate encoded it forces a meaningless DC component to appear, a bias is added to each I&F neuron of the Range fully-connected layer the cancel it.



(a) Matrix Range Fourier transform: the chirp sample map X is multiplied by $\begin{bmatrix} W_\Re & W_\Im \end{bmatrix}$ to obtain the range map Y as expressed more compactly in Equation 33



(b) Matrix Doppler Fourier transform: the transposed range map $\begin{bmatrix} \Re(Y)^T & \Im(Y)^T \end{bmatrix}$ is multiplied by $\begin{bmatrix} W_\Re & W_\Im \\ -W_\Re & W_\Re \end{bmatrix}$ to obtain the transposed RD map $\begin{bmatrix} \Re(Z)^T & \Im(Z)^T \end{bmatrix}$ as expressed more compactly in Equation 34

Figure 17: Visual matrix representation of the Range-Doppler Fourier transforms with the shape of each matrix

4.3 Batch Normalisation Layer

Most architectures contains batch normalisation layers to help it converge faster. In a spiking network, such a layer is complicated to implement since the output of each I&F neuron would need to be multiplied by definition of the normalisation.

However, once a network is trained, the weights from the convolutional and the following batch normalisation layer can be merged so that the batch normalisation layer is no longer necessary [17]. If the input of the convolutional layer is X and its output Y, Equation 35 provides the convolutional operation between them where K is the matrix of kernel weights and b is the matrix of biases.

If $\tilde{Y}$ is the output of the batch normalisation layer, Equation 36 shows how $\tilde{Y}$ can be re-expressed as a fully convolutional layer where the weights and bias are only scaled and shifted with the following parameters: μ is the average of the batch, σ is the standard deviation of the batch, β and γ are learnable factors, $\tilde{K}$ is the new kernel weight matrix and $\tilde{b}$ is the new bias matrix. Equation 37 provides these new weights and bias.

$$Y[i,j] = \sum_{m=-k/2}^{k/2} \sum_{n=-k/2}^{k/2} X[i+m, j+n]K[m,n] + b[i,j] \tag{35}$$

$$\begin{aligned} \tilde{Y}[i,j] &= \gamma \cdot \frac{Y[i,j] - \mu}{\sigma} + \beta \\ &= \gamma \cdot \frac{\left(\sum_{m=-k/2}^{k/2} \sum_{n=-k/2}^{k/2} X[i+m, j+n] \cdot K[m,n] + b[i,j]\right) - \mu}{\sigma} + \beta \\ &= \sum_{m=-k/2}^{k/2} \sum_{n=-k/2}^{k/2} X[i+m, j+n] \cdot \underbrace{\frac{\gamma}{\sigma} K[m,n]}_{\tilde{K}[m,n]} + \underbrace{\frac{\gamma}{\sigma}\left(b[i,j] - \mu\right) + \beta}_{\tilde{b}[i,j]} \end{aligned} \tag{36}$$

$$\Rightarrow \tilde{K}[m,n] = \frac{\gamma}{\sigma} K[m,n] \quad \text{and} \quad \tilde{b}[i,j] = \frac{\gamma}{\sigma}\left(b[i,j] - \mu\right) + \beta \tag{37}$$

5. Results

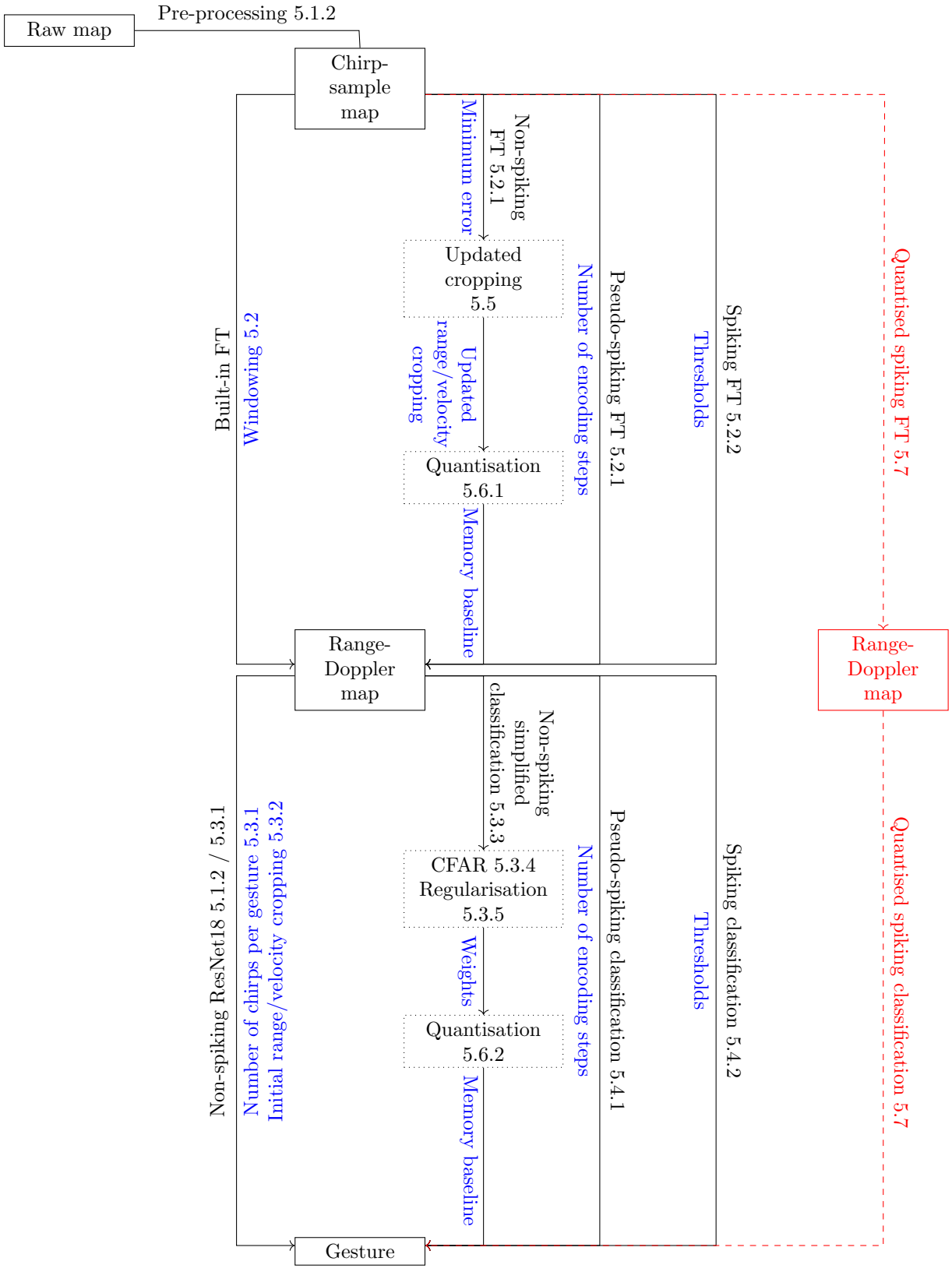


Figure 18: Complete pipeline schematic stating which step is explained in which section. The blue text represents the information extracted from the step, the solid black lines represent the steps that are analysed independently from one another, and the red dotted line represents the complete final pipeline. Reading from left to right follows the conversion from non-spiking to spiking.

This chapter presents the implementation and analysis of the proposed gesture-recognition pipeline from start to end. Section 5.1 provides information about the dataset used from [12] and the results that the authors obtain on a non-spiking pipeline. Sections 5.2, 5.3 and 5.4 provide the individual steps of the pipeline where the spiking output is compared to the non-spiking equivalent. Section 5.5 suggests a way to optimise the obtained pipeline in terms of memory usage. Section 5.6 describes how the pipeline can be quantised, and finally Section 5.7 brings together the spiking FT and classification to create the complete spiking pipeline. Figure 18 illustrates which section addresses which part of the pipeline.

5.1 Dataset

The dataset used has been published by [12], it contains ADC signals generated by an $8GHz$ FMCW radar for a diverse range of air-marshalling gestures. Further work has been performed on the same dataset by [26], this paper's results will therefore also be considered as a baseline for comparison. This dataset has been created with the following characteristics:

- low temporal resolution ADC signals (the signal has a bandwidth of $750MHz$ sampled only 512 times per chirp) to enable the exploration of compression strategies and low-precision encodings,
- 11 aircraft marshalling signals (10 movements plus a 'no movement' class): complex whole-body gestures that engage more than one fundamental feature for recognition,
- a range of 8 different distances from the radar at three different locations (conference room, foyer, and open-space), which makes the dataset rich in spatio-temporal content and allows for high generalisation,
- 13 different people are performing the gestures.

The initial dataset comes as two folders: a test dataset and a 'training' dataset that is then subdivided into train and validation sets for this thesis. For a more in-depth representation of the dataset, Figure 48 in Appendix 7.4 shows the chirp distribution of who made the gestures, in which location and at which distance. An interesting point to make is the high variation between the train and the test sets.

Each gesture is performed by a specific person at a specific location. The radar signal is then recorded for 192 chirps. These chirps are then grouped in frames, an example of a frame had been provided as the raw map in Figure 2. The data comes as a series of frames organised in 3D tensors: the first dimension represents the frames, then the chirps and finally the number of samples per chirp.

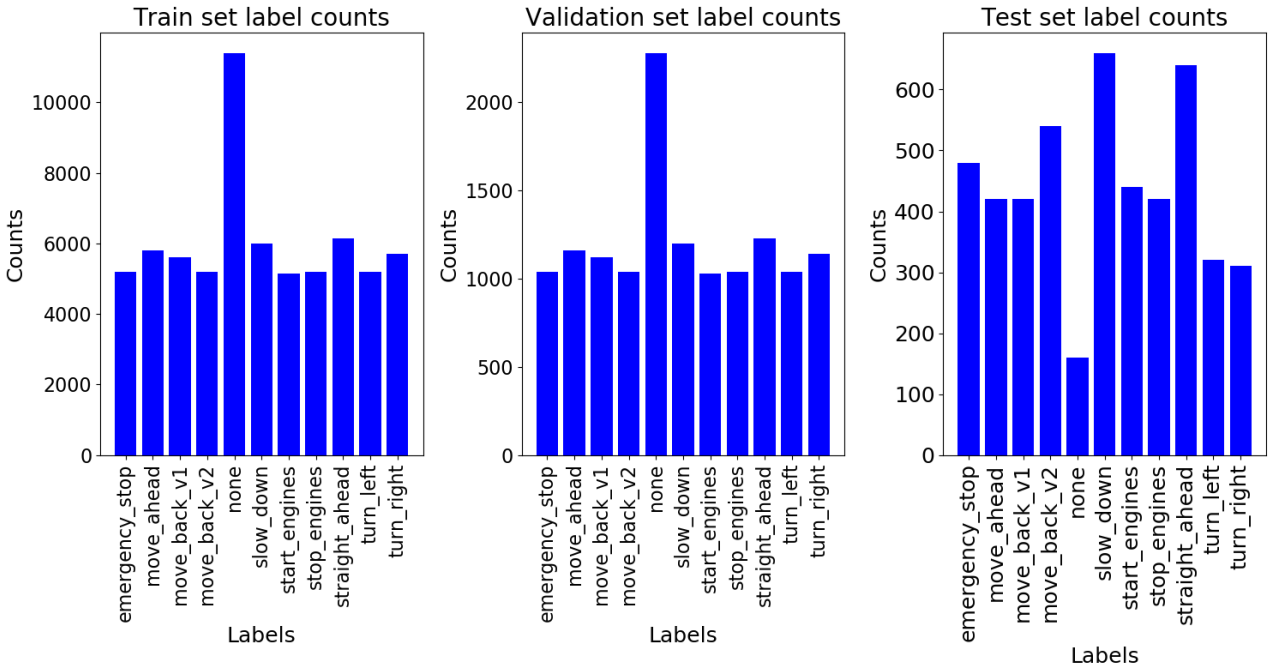


Figure 19: Label distribution per frame for the train, validation and test sets. The train and validation sets used in this thesis are made from a $80\% - 20\%$ distribution of the train dataset from [12].

Since different gestures take different amounts of time, it is not possible to know in advance how many chirps represent one gesture. Furthermore, different gestures might require different amounts of time. Nevertheless, as will be shown in Section 5.3.1, using 5 frames (equivalent to 960 chirps) represents enough for each gesture to give accurate results. For clarity, all chirps in a set of 5 frames will hence be considered one data point.

When using 5 frames per data point, the total number of training data is reduced by 5× compared to the case where each frame is considered one data point. To counter this, a data augmentation technique is used: a window of size 960 chirps (equivalent to 5 frames) is slid across the 3D data matrix with a step of 192 chirps (equivalent to 1 frame) so that each data point is made out of 5 frames and differs from its neighbour by one frame. The gestures are performed repetitively within a 3D block of frames which makes this augmentation meaningful.

Figure 19 shows the number of frames used in each of the train, validation, and test sets when 960 chirps per frame are used. The dataset initially contains a train and a test folder with respectively 66 560 and 4 736 frames. The initial train folder is separated into a train and a validation set with an $80\% - 20\%$ split.

Sub-section 5.1.1 highlights the specifications of the dataset while Sub-section 5.1.2 provides the results obtained on the dataset by its authors as well as highlighting the pre-processing steps used.

5.1.1 Specifications

As explained above, the data contains frames where each row represents a chirp, the columns are the chirp's samples. They have the following specifications:

- centre frequency: $f_C = 7.3GHz$,
- carrier wavlength: $\lambda = 41.1mm$,
- bandwidth: $B = 750MHz$,
- chirp time: $T_c = 40.96\mu s$,
- time between chirps: $T_{c,diff} = 1.3ms$,
- chirps per frame: $N_c = 192$,
- ADC samples per chirp: $N_s = 512$,
- sampling frequency: $f_s = 12.5MHz$,
- range maximum: $d_{max} = 76.288m$,
- range resolution: $\Delta d = 0.149m$,
- velocity maximum: $v_{max} = 15m/s$,
- velocity resolution: $\Delta v = 0.5m/s$.

Using Equations 7 and 8 as well as the above information, the maximum distance and velocity are calculated in Equations 38 and 39 [4].

$$d_{max} = \frac{(3 \cdot 10^8 m/s) \cdot (40.96\mu s)}{2 \cdot (750MHz)} \frac{12.5MHz}{2} = 51.2m \tag{38}$$

$$v_{max} = \frac{3 \cdot 10^8 m/s}{7.3GHz} \frac{1}{4 \cdot 1.3s} = 7.9m/s \tag{39}$$

5.1.2 Processing Pipeline and Results

The authors in [12] apply two different non-spiking processing pipelines to the dataset resulting in the 11-gestures test accuracy results shown in Table 9. The accuracy of the ResNet18 model will be considered since the EfficientNet-B1 is a more complex architecture which hence requires many more parameters to be tuned.

Paper [12] pre-processes the data by applying a first-order delta filter on the raw frames shown in Equation 40: the processed chirp $x_{delta}(n_{chirp}, n_{sample})$ is found by substracting the raw chirp $x(n_{chirp}, n_{sample})$ by the previous raw chirp $x(n_{chirp} - 1, n_{sample})$. This is followed by a thresholding described in Equation 41 where the output $x_{binary}(n_{chirp}, n_{sample})$ is obtained by thresholding $x_{delta}(n_{chirp}, n_{sample})$ with the standard deviation σ.

$$x_{delta}(n_{chirp}, n_{sample}) = x(n_{chirp}, n_{sample}) - x(n_{chirp} - 1, n_{sample}) \tag{40}$$

$$x_{binary}(n_{chirp}, n_{sample}) = \begin{cases} 1 \text{ if } x_{delta}(n_{chirp}, n_{sample}) > \sigma \\ -1 \text{ if } x_{delta}(n_{chirp}, n_{sample}) < -\sigma \\ 0 \text{ otherwise} \end{cases} \tag{41}$$

[4] There are some discrepancies between the maximum range and maximum velocity compared to what is stated in [12], which is, to the best of the author's knowledge, probably due to some correction factor used in that paper.

Paper [26] demonstrates that its classification algorithm is performing twice as well on the raw data that has been pre-processed with delta filtering (50.0% test accuracy) compared to using pure raw data (25.1% test accuracy). This makes sense since the filtering has removed the static objects by removing the DC components, leaving out only the relevant signal.

This work therefore uses delta filtering for increased accuracy. It also implements thresholding, since this involves a binary input to the spiking Fourier transform which means fewer encoding steps are necessary.

Processing	Classification network	11-class test accuracy
• Range-Doppler map with Hann windowing. • Doppler axis cropping to the maximum velocity.	ResNet18	59.1%
	EfficientNet-B1	64.6%
• First-order delta filter. • Level-crossing. • Range-Doppler map with Hann windowing • Doppler axis cropping to the maximum velocity.	EfficientNet-B1	63.31%-64.58%

Table 9: Accuracies obtained by [12] on their dataset for different classification networks and processing steps.

5.2 Frequency-domain Analyses

The literature research in Section 3.2 shows that there exist two main ways of extracting the frequency content of the radar signals: μDoppler signatures and RD maps. This work uses RD maps since the study done in [26] on the same dataset shows 57.5% 11-gestures test accuracy for μDoppler-signatures compared to 67.7% for RD maps.

Furthermore, the authors of the initial paper [12] use Hann windowing to minimise the non-idealities of the finite FT. However, paper [26] shows that omitting windowing on the same dataset results in comparable accuracies, which is true for Hanning, Hamming and Blackman Harris windowing, hence it is also dropped in this work. However, if another dataset would make use of a windowing technique, the windowing coefficients could be easily multiplied by the FT coefficients with no added memory needs nor added operations.

Sub-section 5.2.1 provides the results from the pseudo-spiking equivalent of the FT which is used to approximate the error that will be obtained on the spiking network. Following this, Sub-section 5.2.2 calculates the theoretical thresholds. It also proves their relevance by showing that the spiking FT using those thresholds achieves about the lowest error between the spiking FT and the Python built-in FT. Furthermore, this lowest error is the one predicted by the pseudo-spiking network, proving the validity of the definition of the pseudo-spiking network.

5.2.1 Non-spiking and Pseudo-spiking Fourier Transforms

The methodology behind the design of the spiking (Range and Doppler) FTs has been explained in Section 4.2. The pseudo-spiking equivalent of the FT is built according to the explanations in Sub-section 4.1.3. Figure 20 shows how the per-pixel normalised error of the pseudo-spiking network evolves with the number of encoding steps. This is used as a prediction for the number of encoding steps to use for spiking network, the validity of this error curve with respect to the spiking error is explained later.

Throughout this work, the error is considered as the per-pixel average difference between the modulus of the spiking RD map and the modulus of the PyTorch FT-generated RD map. Furthermore, since the dataset does not provide units for the raw data, all pixel values for the raw map, the pre-processed map, the Range map and the RD map are considered as unitless intensities.

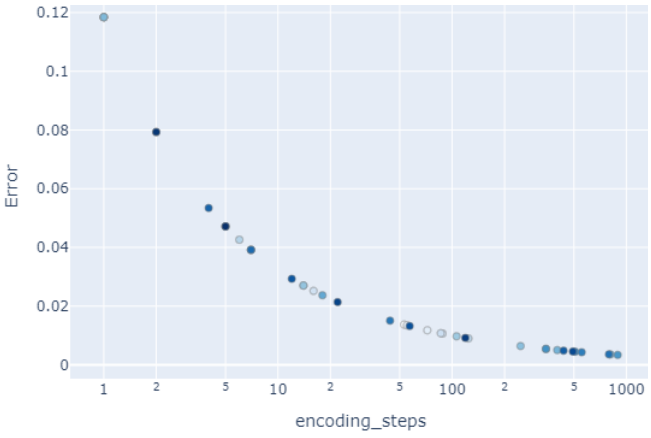
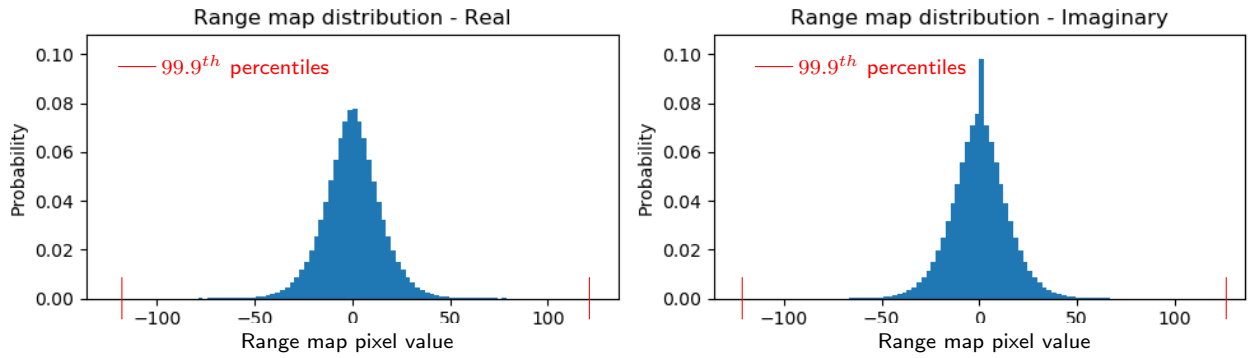


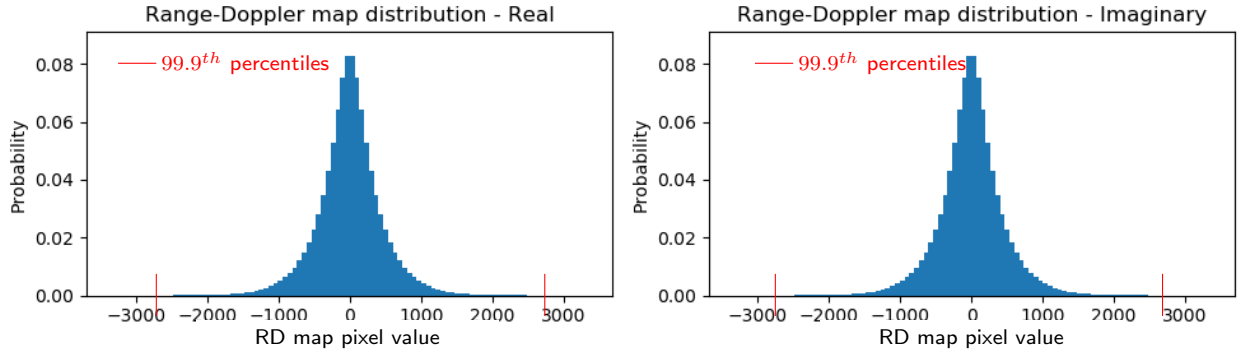
Figure 20: Normalised error per pixel of the RD maps created by the pseudo-spiking network relative to a built-in Python FT for a range of encoding steps.

The normalised average error between the built-in PyTorch FT and a non-spiking FT neural network is $6 \cdot 10^{-6}$ per RD map pixel for the dataset from [12]. This is caused by the non-idealities of the discrete implementation causing side-lobes. In theory, an infinite number of encoding steps would therefore result in an error of $6 \cdot 10^{-6}$ per pixel compared to a continuous FT.

5.2.2 Spiking Fourier Transforms



(a) Range maps distribution, the 99^{th} percentile of these distributions is taken as the maximum clamping value Y_{max}, which is here 123.5.



(b) Range Doppler maps distribution, the 99^{th} percentile of these distributions is taken as the maximum clamping value Z_{max}, which is here 2880.0.

Figure 21: Distribution of the maps used to find the maximum value to be represented by the spiking FT network based on the train set.

The next step is to decide on the thresholds for the Range and Doppler FTs fully-connected layer I&F neurons by using the theory developed in Section 4.1.2.

Equation 30 shows that the theoretical threshold of the first layer of a spiking neural network is expressed in terms of the minimum X_{min} and maximum X_{max} of the input of that layer and of the maximum output Y_{max} of that layer. In this case, the first layer calculates the range FT. The input's maximum and minimum are $X_{min} = -1$ and $X_{max} = 1$ due to the level-crossing pre-processing of the raw map. To find Y_{max}, Figure 21a shows the distribution of the real and imaginary parts of the range maps. Since the distributions contains very large/small values that are very improbable, Y_{max} is taken as the 99.9^{th} percentile of the distributions which is $Y_{max} = 123.5$. The percentile could be chosen as other values, but an analysis of which percentile to choose is kept as future work. As a reminder, some neurons are in charge of the positive values, and others of the negative values. The latter ones also use $Y_{max} = 123.5$ since they are processed in the absolute form. Equation 42 then calculates the theoretical range threshold $U_{th,Y}$ from these values.

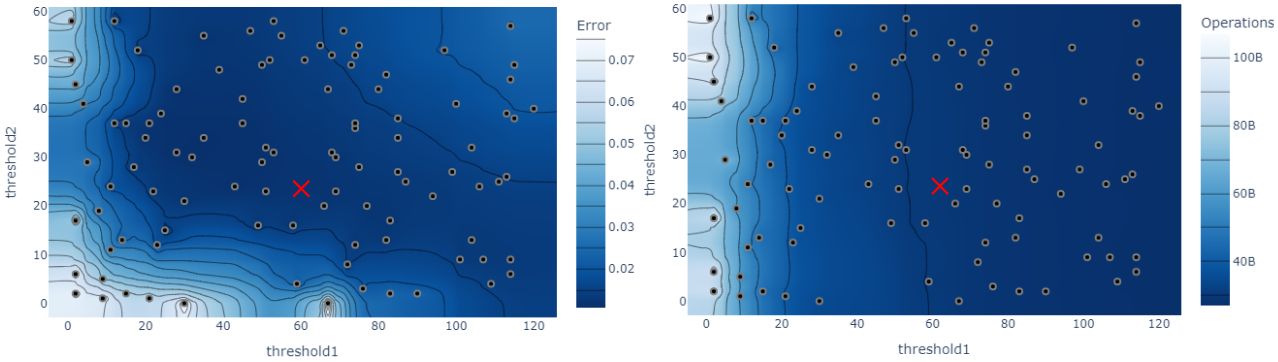
The same process is achieved for the second spiking layer representing the Doppler FT. As discussed, the input of the layer is set as $Y_{max} = 123.5$. To find the maximum output of the second layer Z_{max}, the distribution of the pixel values of the RD maps are provided in Figure 21b. Taking again the 99.9^{th} percentile results in $Z_{max} = 2880.0$, which enables the calculation of the Doppler threshold $U_{th,Z}$ as presented in Equation 43.

$$U_{th,Y} = \frac{Y_{max}}{X_{max} - X_{min}} = \frac{123.5}{2.0} = 61.8 \tag{42}$$

$$U_{th,Z} = \frac{Z_{max}}{Y_{max}} = \frac{2880.0}{123.5} = 23.3 \tag{43}$$

Figure 22a shows the error when 100 steps are used for a range of different thresholds. The following can be noted:

- when the thresholds are those theorised, the normalised per-pixel error is predicted by the pseudo-spiking curve in Figure 20: the error for the theorised thresholds is 0.0112 per pixel, which is nearly the minimum obtained error on the spiking network (0.0107 per pixel), and very close to the error predicted by the pseudo-spiking curve (0.0100 per pixel),
- the predicted thresholds are not the only ones where the minimum error is achieved. There are two reasons for this:
  - the theoretical thresholds assume certain maximum values (Y_{max} and Z_{max}), here chosen as the 99.9^{th} percentiles. While a datapoint for which it is important to represent those maximum values will perform the best with those predicted thresholds, another datapoint which has lower maximum points would have performed better with its corresponding thresholds. Overall, since not all data points have the same significant maximum values, a range of threshold pairs can perform as well as each other given that the data points with high errors are compensated by others,
  - as shown on Figure 14, the decoded spiking output depends on a factor $U_{th} \cdot \sum_t S[t]$. Decreasing the threshold generates more spikes (hence increasing $\sum_t S[t]$) and vice versa, this factor can hence stay constant for a range of thresholds while maintaining low error. This could have been a tuning knob to increase sparsity, however, Figure 22b shows that the number of operations is about constant for thresholds reaching low error. Furthermore, it must be noted that the resulting encoding is no longer 'perfect' (in the sense that a number encoded with full spikes represents Y_{max}/Z_{max} as described in Sub-section 4.1.2), which means that if more layers following this double FT were to be added the theoretical thresholds of the following layers would no longer be well defined,
- as explained in the previous point, low sparsity is obtained for low error, there is hence no trade-off to be reached, this is further highlighted in Figure 49a in Appendix 7.4 which provides the error-number of operations Pareto front of the data in Figure 22.

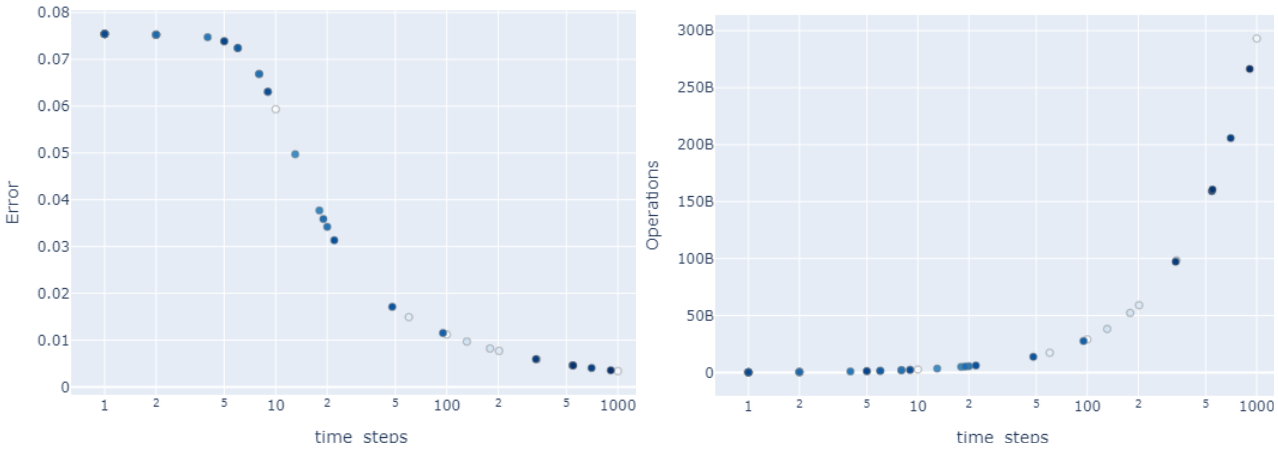


(a) Average normalised error per RD map relative to the built-in Python FT

(b) Number of add operations per RD map

Figure 22: Threshold tuning for the spiking FT neural network using 100 encoding steps: threshold1 = $U_{th,Y}$ and threshold2 = $U_{th,Z}$, those values are averaged over all training data. The red crosses represent the theoretical thresholds: $U_{th,Y} = 61.8$ and $U_{th,Z} = 23.3$.

The last step to characterise the FTs fully is to express how the number of encoding steps impacts both the number of operations and the error as shown in Figure 23, which uses when the theoretical thresholds. These graphs show the trade-off between the error and the number of operations.



(a) Normalised error

(b) Number of add operations

Figure 23: Number of operations - error trade-off of the spiking Fourier Transforms using the theoretical thresholds $U_{th,X} = 61.8$ and $U_{th,Y} = 23.3$.

A comparison between the error in the pseudo-spiking network (Figure 20) and the spiking network (Figure 23a) reveals that the pseudo-spiking method effectively approximates the minimum achievable error and the thresholds required to attain it. It must be noted that it only works for enough encoding steps, which is because the spiking neurons will retain some part of the information in their membrane potential at the last step that will not lead to any spikes. If fewer steps are used, then this information is more significant than for the cases with more steps. This is not the case for the pseudo-spiking equivalent since it does not spike.

In Figure 18, the current section has found the following:

- no windowing should be used,
- the minimum theoretical FT error which is $6 \cdot 10^{-6} V$,
- the number of encoding steps per acceptable error for the FT network is predicted in Figure 20,
- the FT theoretical thresholds are validated as $U_{th,Y} = 61.8$ and $U_{th,Z} = 23.3$.

5.3 Non-spiking Classification

This section focuses on how the non-spiking classification neural network is designed so that it reaches a good trade-off between its complexity and accuracy. Starting from the ResNet18 from [12], each of the following sub-sections provides a simplification of the network to reduce the computational requirements. Table 10 provides the baseline from [12], as well as this work's baseline defined as the replication of the ResNet18 from [12], and summarises the results obtained in each sub-section.

Architecture	Val. accuracy	Test Accuracy		Computational requirements		Sub-section
	11-gestures	11-gestures	Top 5-gestures	Memory (number of weights)	FLOPS [5]	
ResNet18 from [12]	-	59.1%	86.9%	-	-	-
ResNet18 replication - baseline	98.5%	59.8%	86.7%	$11.2M$	$17.1G$	5.3.1
ResNet18 with RD map cropping (input size reduced to 0.7%)	98.3%	58.3%	82.8%	$11.2M$	$154.7M$	5.3.2
Architecture simplification to one convolutional and one fully-connected layer	97.0%	48.1%	79.6%	$138k$	$7.5M$	5.3.3
Normalised input with regularisation	97.0%	50.7%	77.6%	$138k$	$7.5M$	5.3.5
Total change	-1.5%	-8.4%	-9.3%	1.2% of baseline	0.04% of baseline	-

Table 10: Accuracies and computational resources for the non-spiking classification

5.3.1 ResNet

The authors in [12] test their dataset on a ResNet18 architecture which obtains 86.9% on their five best classes and 59.1% on all eleven classes. More detailed explanation about the ResNet18 architecture can be found in Appendix 7.3: Table 21 shows the parameters of a ResNet18 architecture and Table 22 shows the sizes of the outputs of each layer.

Since there is no information about good hyper-parameters and regularisation methods in [12], a search is performed to tune the batch size, the learning rate, dropout and weight decay. Figure 24 shows the training curve the ResNet18 achieves for the values given in Table 11. These parameters are chosen by looking for a high maximum validation accuracy as well as for a smooth curve to ensure that the learning rate is well-tuned. Early-stopping can be applied at 45 epochs where the validation accuracy is 98.5%.

Figure 25a shows how well each gesture from the test set is classified, the 11-classes accuracy is 59.8% which is nearly equal to the accuracy reached in [12] (59.1%). The 5-best-classes accuracy is 86.7% (also very similar to the accuracy in [12] which is 86.9%) where the 5 best classes are: 'move ahead', 'move back v1', 'move back v2', 'start engines' and 'stop engines'.

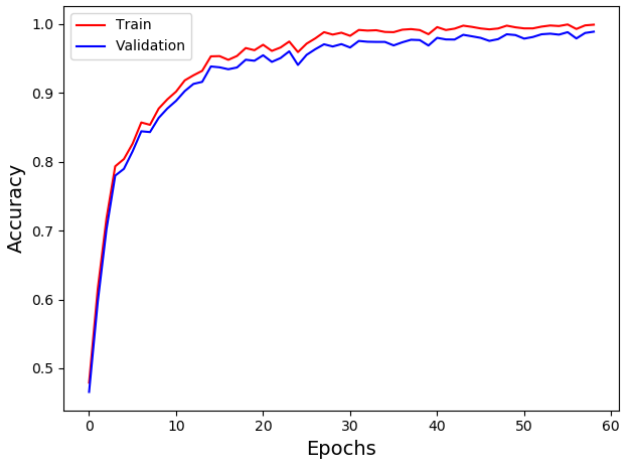
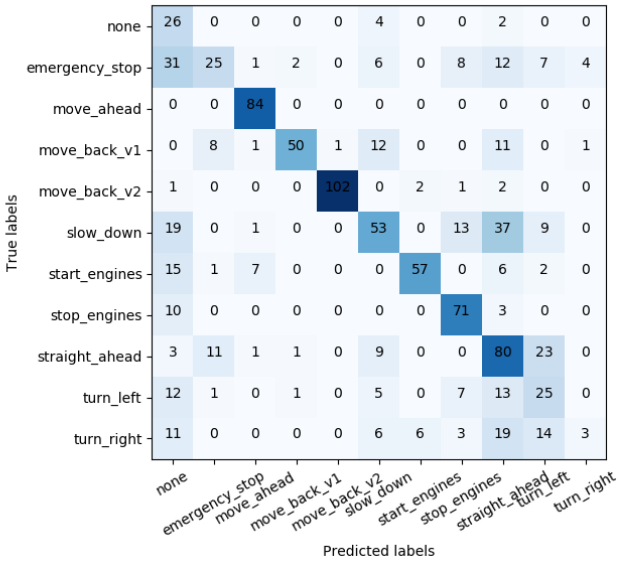


Figure 24: Accuracies through the training epochs (non-cropped maps)

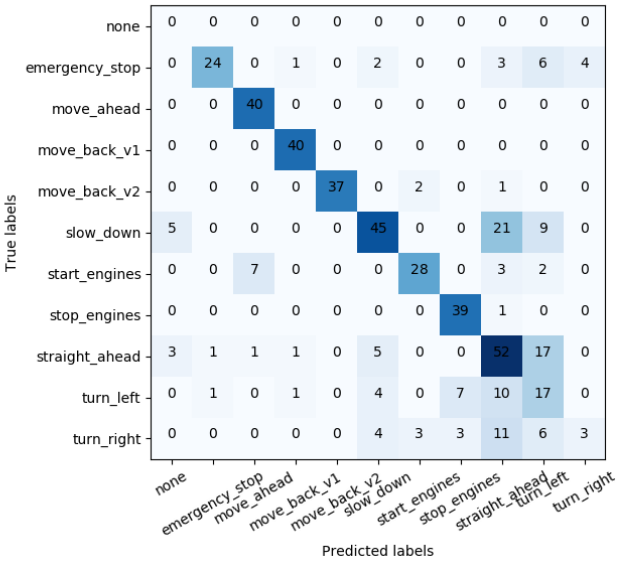
Batch size	Learning rate	Dropout	Weight decay
120	$1 \cdot 10^{-5}$	0.05	$4 \cdot 10^{-5}$

Table 11: ResNet18 hyperparameters

[5] Approximation ignoring the max-pooling layers as the library calculating FLOPS does not include them.



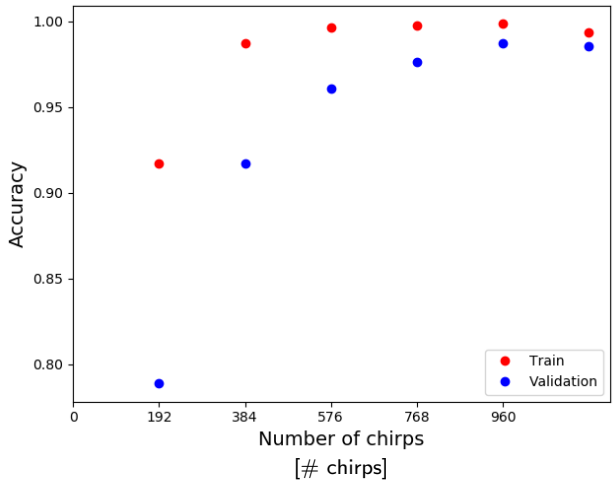
(a) All people and locations (total of 965 test samples)



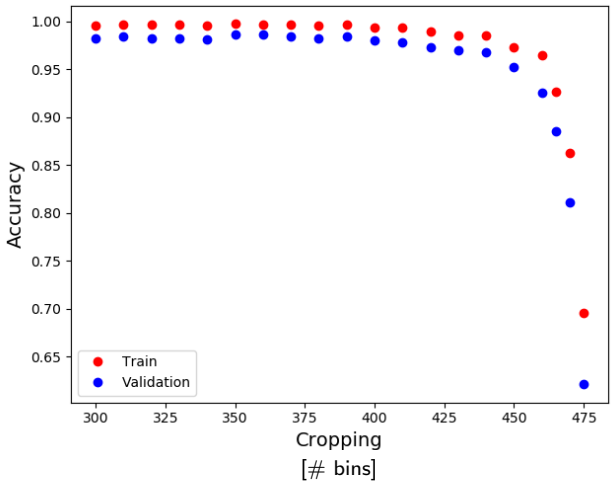
(b) Only the people (2, 3 and 5) and location (conference room) present in the train/validation set (total of 470 test samples)

Figure 25: ResNet18 confusion matrices on the test set

As shown in Figure 48 in Appendix 7.4, the test set contains very different data compared to the training set, hence explaining the large accuracy drop. To exemplify this, Figure 25b only uses test samples that are taken in the same condition as the training and validation set: it samples data done by people and in locations that are present in the training/validation data. In this case, the accuracy increases to 69.1%



(a) Maximum train and validation accuracies obtained for different numbers of chirps per RD map on ResNet18.



(b) Maximum train and validation accuracies obtained by training with inputs of reduced sizes. The RD maps Doppler bins are gradually removed on both the positive and negative Doppler frequencies, the x-value 'cropping' above represents the number of such bins that are removed before training. The number of remaining Doppler bins is hence $960 - 2 \cdot \textit{cropping}$ bins.

Figure 26: ResNet18 accuracies for cropped RD map and RD maps with different number of chirps

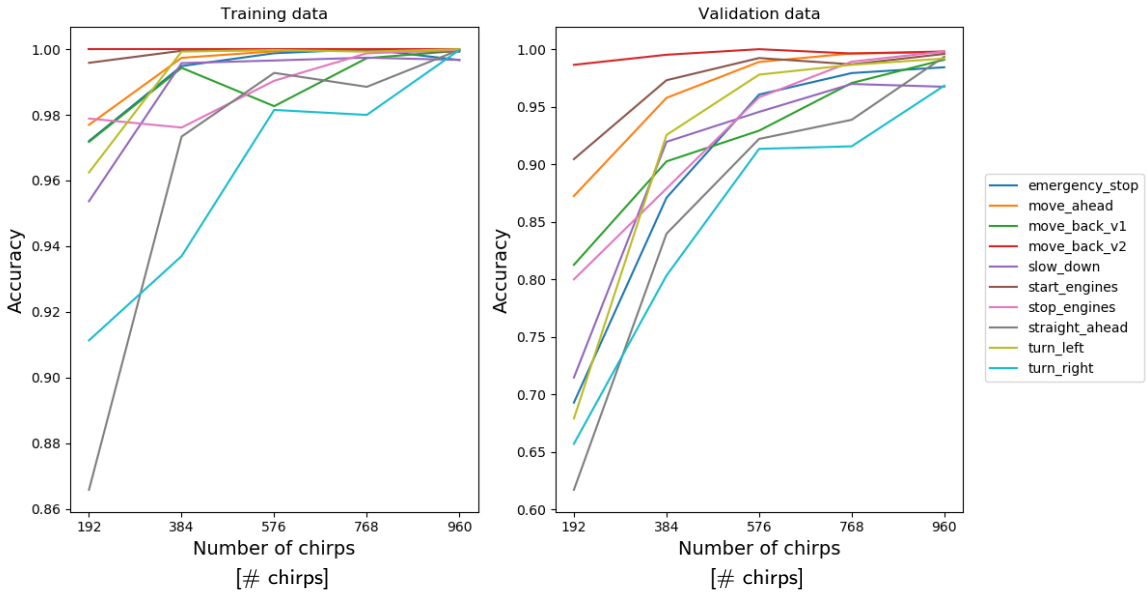


Figure 27: Train (left) and validation (right) recall per gesture class obtained on ResNet18 for different numbers of chirps per RD frame.

The previous training has been achieved by using 960 chirps per data point. Figure 27 shows the maximum train and validation recalls obtained using different numbers of chirps, which validates the choice of using 5 frames (equivalent to 960 chirps) per data point since the gain increase starts to plateau from that value onwards. For a more detailed analysis, Figure 27 shows how the recall of each gesture class evolves with the number of chirps. This graph shows that some gestures are shorter gestures: for example 'move back v2' which has a high recall even when using only 192 chirps. On the other hand other gestures are much longer: for example 'turn right' which requires the 960 chirps to be well classified.

Since the classification network requires an input of constant size, the 960 chirps are hence used for the final architecture. However, it would be interesting to investigate other models that would be able to take advantage of this, such as a having a first model based on fewer frames which would make a first decision, then based on how certain the algorithm is of its prediction, it could decide to trust it or pass the frame along to a more complex classification model. This would reduce the energy usage if most gestures can be classified with fewer chirps, while maintaining the accuracy high by using more resources only when needed.

5.3.2 Cropping

In the range dimension, the authors in [12] specify that the gestures are performed at a maximum of $4.5m$. When the range frequencies are translated to distances, this shows that only 45 out of 512 range bins should be kept. Furthermore, the input data of the FTs are real values which makes the output symmetric. As a consequence, only half of these range bins should be kept. In the Doppler dimension, there is no maximum velocity defined in [12], however, there must be an upper bound since human gestures' speed is limited.

In the following, 'cropping' refers to removing frequency bins in both the positive and corresponding negative frequencies.

An example of an RD map is shown in Figure 28a. To crop the RD map in the Doppler dimension, Figure 26b shows how the train and validation accuracies evolve as the input frame is cropped along the Doppler bins. It can be seen that up to 400 bins can be removed on both sides without significant accuracy drop hence leaving $960 - 2 \cdot 400 = 160$ Doppler frequency bins. The resulting RD map is shown in Figure 28b.

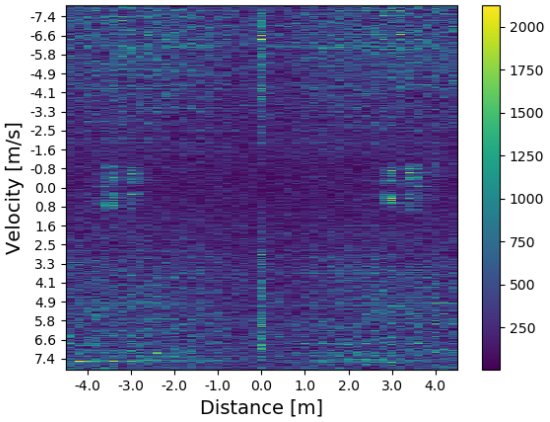
As explained above, since the range map is symmetric, half of the Range bins are kept. Those remaining 23 Range bins are exemplified in Figure 28c. The RD map, initially with 512 range bins and 960 Doppler bins, has been reduced to 23 and 160 bins, respectively. This is equivalent to only 0.7% of the initial RD map.

5.3.3 Architecture Simplification

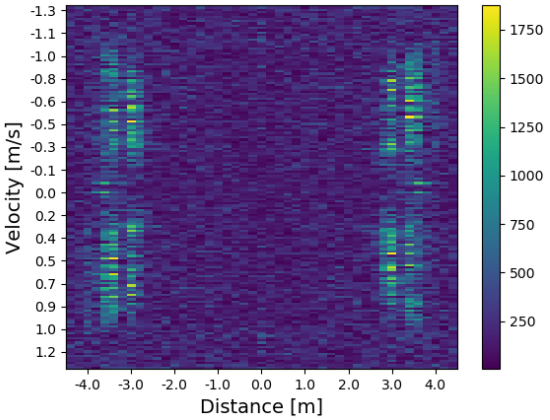
While this ResNet18 architecture reaches good validation accuracies relative to [12], it requires a large number of operations as shown in the number of layers and the output sizes shown in Tables 22 and 21 in Appendix 7.3, Figure 47 provides the architecture.

The particularity of ResNet architectures are to use residual connections, the architecture of ResNet18 can be summarised as follows :

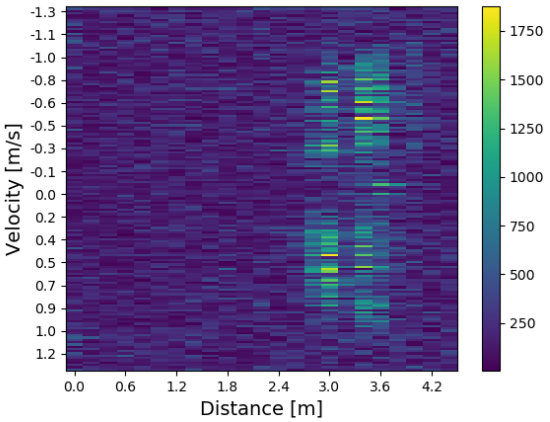
- an input convolutional layer,
- a maxpooling layer,
- a series of sub-blocks containing:
  - convolutional and batch normalisation layers,
  - residual connections between the block's input and output,
- a maxpooling layer,
- a fully connected layer.



(a) Example RD map ($N_c = 960$ and $N_s = 512$).



(b) Example RD map with cropped velocity ($N_c = 160$ and $N_s = 512$).



(c) Example RD map with cropped velocity and half the range ($N_c = 160$ and $N_s = 23$).

Figure 28: Example of an RD map before and after cropping with the correspsing map sizes (N_c is the number of chirps and N_s the number of samples per chirp).

It has been tested that removing the residual connections does not decrease the train and validation accuracies. A series of tests were made by adapting the number of pooling layers and convolutional layers, and it has been found that using as little as one convolutional layer reaches high accuracies. Figure 29 shows an architecture with a first optional average pooling layer, a convolutional layer with its batch normalisation layer, a second optional average pooling layer and the final fully connected layer. It has been decided to use only average pooling layers instead of max-pooling layers since the conversion to the spiking equivalent is direct for the average pooling but not for the max-pooling layer.

A number of experiments are run to test for the best parameters. The pooling layers are added or removed from the architectures in those experiments to test the best arrangement. The batch size and learning rate are also added as hyper-parameters for a first approximation, this will be refined in Sub-section 5.3.5.

In order to take into consideration the trade-off between the number of operations and the accuracy, the Pareto front is extracted for a range of tested architectures. Figure 30 shows the following architecture cases with only one convolutional layer:

- No Avgpool1/No Avgpool2: Input → Conv → FC,
- No Avgpool1/With Avgpool2: Input → Conv → Avgpool2 → FC,
- With Avgpool1/No Avgpool2: Input → Avgpool1 → Conv → FC,
- With Avgpool1/With Avgpool2: Input → Avgpool1 → Conv → Avgpool2 → FC.

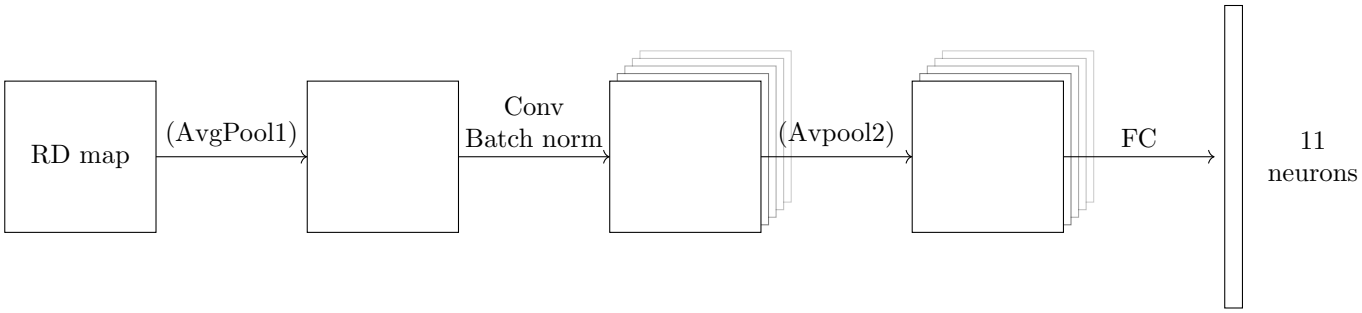


Figure 29: Customisable classification architecture containing: a first optional average pooling layer, a mandatory convolutional layer with batch normalisation, a second optional average pooling layer, and a mandatory fully-connected layer that contains 11 output neurons for the gesture classes.

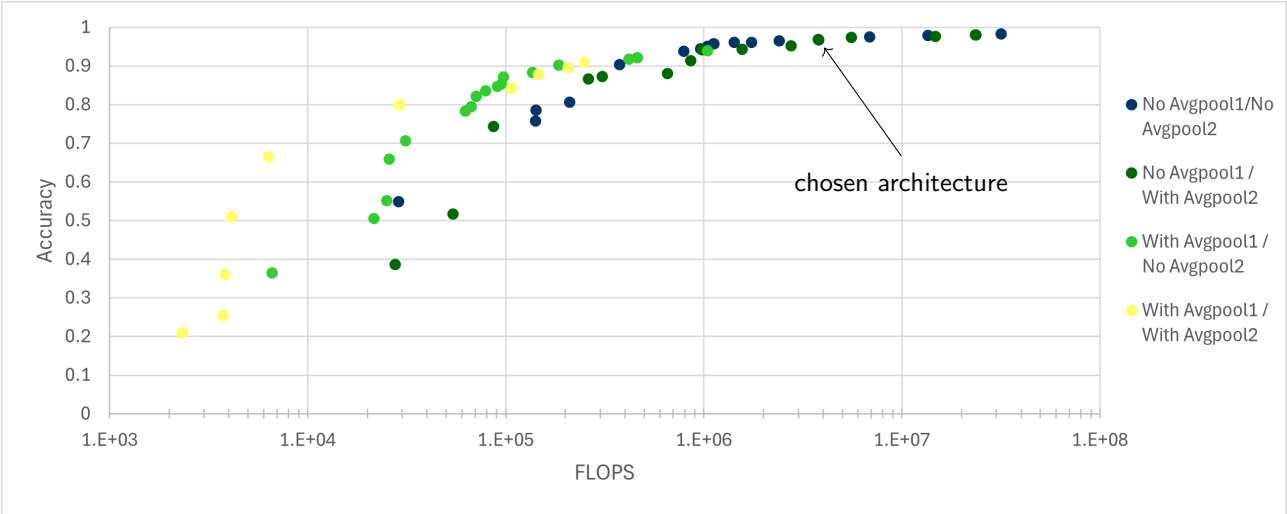


Figure 30: Pareto front for the architecture in Figure 29 with or without the first and second average pooling layers.

Figure 30 shows that it is possible to reach 98.4% validation accuracy, which is as high as was achieved by the initial ResNet18. It is hence more than enough to focus on an architecture using only one convolutional layer.

To obtain more insight into which architecture to choose, Figure 31 shows the distribution of operations across layers for validation accuracies reaching more than 95% (the others can be found in Figure 50 in Appendix 7.4). While the FLOPS-accuracy trade-off shows one side of the argument, it ignores the memory aspect. To satisfy this, Figure 31b illustrates how the number of parameters per layer evolves with the increasing accuracy for validation accuracies above 95% (the full figure can be found in Figure 50 in Appendix 7.4). Both the number of operations and the number of parameters show a logarithmic increase with the validation accuracy.

It is decided that a good accuracy-flops-memory trade-off is achieved by the architecture reaching 97.0% validation accuracy. Its parameters are given in Table 12 and the sizes of the outputs of each layer are given in Table 13. With further inspection, the average pooling is practically selecting one every two pixels from the Doppler bins since it has a kernel size of 1 and a stride of 2. In practice, which means that the pooling layer can be removed and the convolutional layer's stride can be modified.

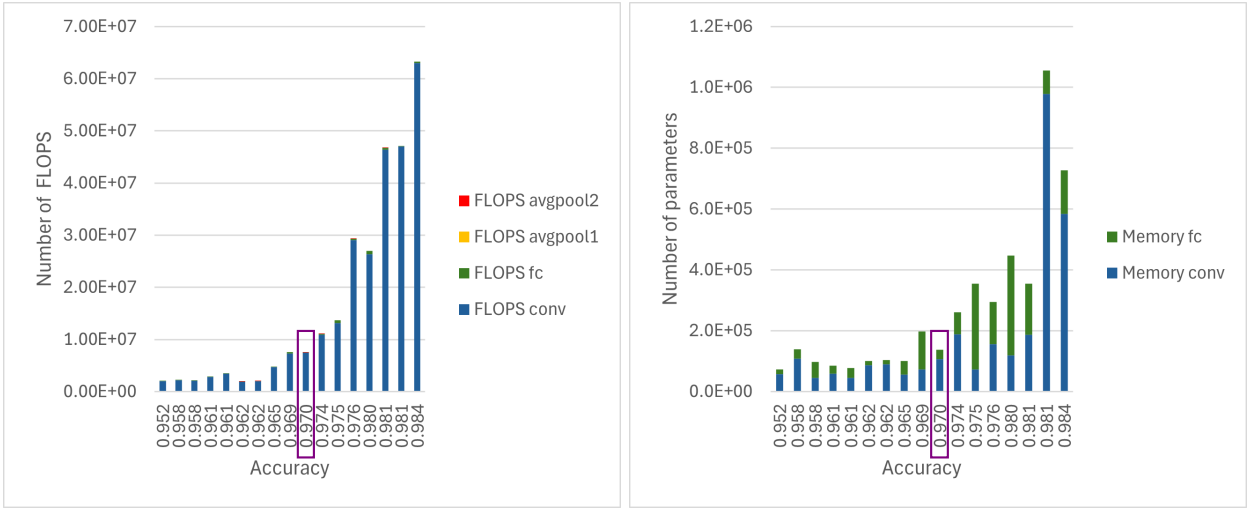
The chosen architecture can be compared with the ResNet18 (Tables 21 and 22 in Appendix 7.3) and the simplification of the classification architecture is significant.

Layer		Conv	Avgpool2	FC
Parameters	Kernel size	26	1	11
	Padding	2	0	
	Stride	4	2	
	Output kernels	158	158	

Table 12: Final architecture parameters per layer.

	Input	Conv	Avgpool2	FC
Height	160	35	18	11
Width	23	1	1	
Kernels	1	158	158	

Table 13: Final architecture sizes of outputs.



(a) Number of operations per layer for the architectures in Figure 30 for validation accuracies above 95%.

(b) Number of parameters per layer for the architectures in Figure 30 for validation accuracies above 95%.

Figure 31: Pareto front operations and memory comparisons (the architectures reaching below 95% accuracy are shown in Figure 50 in Appendix 7.4).

5.3.4 CFAR

The literature review in Section 3.3 provides examples where a CFAR algorithm is used to remove noise by thresholding the RD map.

To test the utility of a CFAR step in this pipeline, the simplified network is re-trained with the input that has been passed through the CA-CFAR tuning through different α_{cfar}, number of neighbouring cells and number of guarding cells. Two sets of tuning have been used, the first binarises the data, and the second applies a mask according to Equation 44.

$$
\begin{aligned}
output_{binary}[x,y] &= \begin{cases} 1 & \text{if } \alpha_{cfar}\ \ input[x,y] > \text{average(neighbouring input cells)} \\ 0 & \text{otherwise} \end{cases} \\
output_{mask}[x,y] &= \begin{cases} input[x,y] & \text{if } \alpha_{cfar}\ \ input[x,y] > \text{average(neighbouring input cells)} \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{44}
$$

In the tuning of the number of neighbouring cells, guarding cells and α_{cfar}, the maximum validation accuracy is 28.9% for the binary case and 54.3% for the masking case, Figures 51a and 51b in Appendix 7.4 display the hyper-parameter search done to obtain those values. Using an OS-CFAR could have reached better results since it is more sensitive, which is because it does not take the average but the k^{th} largest neighbouring cell as the comparison. However, the OS-CFAR is much slower and would have likely not reached validation accuracies as high as 97.0%. For this reason, the CFAR algorithm is no longer taken into consideration in the rest of the work in this thesis.

5.3.5 Normalisation and Regularisation

The final non-spiking classification architecture has been selected in the previous section, the weights need to be re-trained for the spiking equivalent. Since the input RD map needs to be rate encoded, it first needs to be normalised between 0 and 1. While the raw data for the FT was easy to normalise since it was contained in the values $\{-1, 0, 1\}$, the RD map contains a larger range of inputs between 0 and 37 185. The aim is to clamp RD map input can be clamped [6] the RD map to a maximum value to increase the precision of its rate encoding so that fewer encoding steps will be needed to reach the non-spiking accuracy.

In the same hyper-parameter tuning, the optimal learning rate and the batch size are found again since the input's magnitude has been reduced by normalising it. Finally, in an attempt to increase generalisation on the test set, regularisation terms are investigated. This includes a weight decay factor which reduces the magnitude of the weights and prevents the model from reacting strongly to small changes in the input. The second regularisation technique is to add some dropout, this is the removal of a certain percentage of weights during training, the effect is that the model learns to rely on the full network and not only on certain weights, it hence makes it more robust.

From a tuning on the maximal input value before normalisation, the batch size, the learning rate, the percentage of dropped weights, a weight decay factor, and different weight initialisation strategies (uniform and normal distributions), the following analysis is observed:

- a batch size in the range $50 - 200$ is optimal. It is better to use a smaller batch size since the model sees fewer data points between weight updates and the data is hence more varied. The resulting noise creates a form of regularisation which could be beneficial for a better generalisation on the test set,
- the best learning rates are between $6.9 \cdot 10^{-3} - 0.02$ when the validation accuracy is above 97.0%,
- dropout starts to impact the valisation accuracy significantly from 0.4,
- weight decay should not be used since even very small decays impact the validation accuracy considerably. The magnitude of the decay can be compared to the average weight which is 0.24,
- the type of weight initialisation has little impact, but the best is 'normal' initialisation,
- the RD map is clamped to a maximum value of 1000.

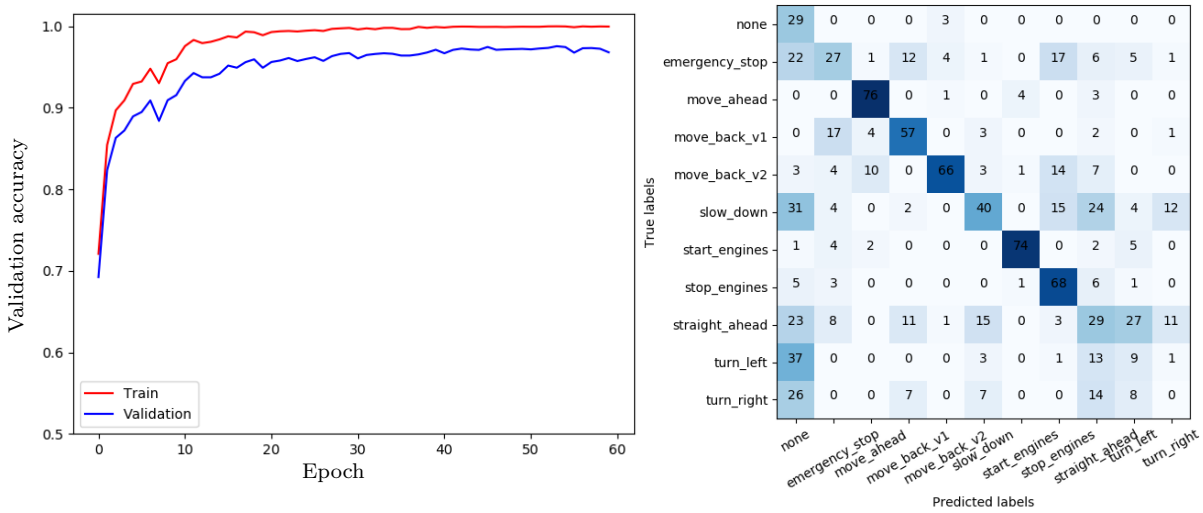
The largest validation accuracy from this tuning is 97.7% with the parameters shown in the first line of Table 14. However, the weights reaching 97.0% with the parameters on the second line of this table are chosen since

[6] Clamping refers to setting all values above/below a certain maximum/minimum number to that value.

they have a higher dropout which should entail a higher generalisation on the test set. It also stops the training at 41 epochs to prevent the model from overfitting too much. The corresponding training curve is given in Figure 32a.

Validation accuracy	Batch size	Learning rate	Dropout	Weight decay	Weight initialisation	Number of epochs	Input clamp
97.7%	68	$0.8 \cdot 10^{-3}$	0.32	$1.89 \cdot 10^{-6}$	normal	58	999
97.0%	50	$0.8 \cdot 10^{-3}$	0.5	0	normal	41	1000

Table 14: Regularisation and training of other hyper-parameters, the parameters reaching 97.0% validation accuracy are kept for the following work.



(a) Training curve

(b) Confusion matrix on the test set

Figure 32: Simplified architecture on normalised RD maps.

Finally, the test accuracy can be obtained on this final architecture. The confusion matrix for the test set is shown in Figure 32b. The 11-gestures accuracy is 50.7% and the 5-best gestures accuracy is 77.6%. This shows that although a smaller architecture can perform as well as the initial ResNet18 on the validation set, it does not generalise as well on the test set since there is about a 10% accuracy reduction of the 11 and 5-best classes compared to ResNet18.

Since the test set comes from a different distribution compared to the train/validation sets, it is possible that the simplified model detects enough features to classify the validation set as well as ResNet18, but that it doesn't detect enough features to differentiate gestures in the test set distribution. It is also possible that the model overfits. However, since the aim of this thesis is to model the spiking pipeline, an in-depth analysis of overfitting is left as possible further work, such as for example, the best early-stopping parameters.

In Figure 18, the current section has found the following:

- the minimum number of chirps per data point required to perform all gestures,
- the acceptable range and velocity cropping,
- a simplified classification network providing weights.

5.4 Spiking Classification

In a similar way to what has been done is Chapter 5.2, the classification network defined needs to be adapted to its spiking equivalent. Sub-section 5.4.1 creates the pseudo-spiking classification and Sub-section 5.4.2 transforms it into the spiking version.

5.4.1 Pseudo-spiking Classification

As given in Table 14, the input has been clamped to 1000 before normalising. In a similar way the outputs of the convolutional and full-connected layers are clamped in a way that does not impact the validation accuracy. By swiping through maximum clamping values, the following are found while maintaining a validation accuracy of 97.0%:

- input: $X_{max} = 1$ (since the network has been trained with the normalised values),
- convolutional layer output: $Y_{max} = 1.2$,
- fully connected layer output: the maximum value is 2 and the minimum is -2.5 since there is no ReLU on the fully-connected layer of the non-spiking network. However, since this output is only proportional to the probability it can be shifted with no accuracy loss by adding a bias of 2.5 so that $Z_{max} = 4.5$. Training the non-spiking with a ReLU following the fully-connected is very inefficient in terms of the validation accuracy reached so this solution is more optimal.

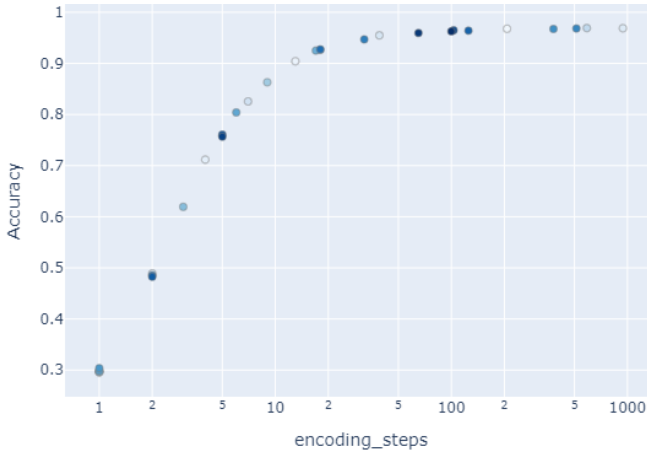


Figure 33: Classification validation accuracy for the pseudo-spiking network for a range of encoding steps.

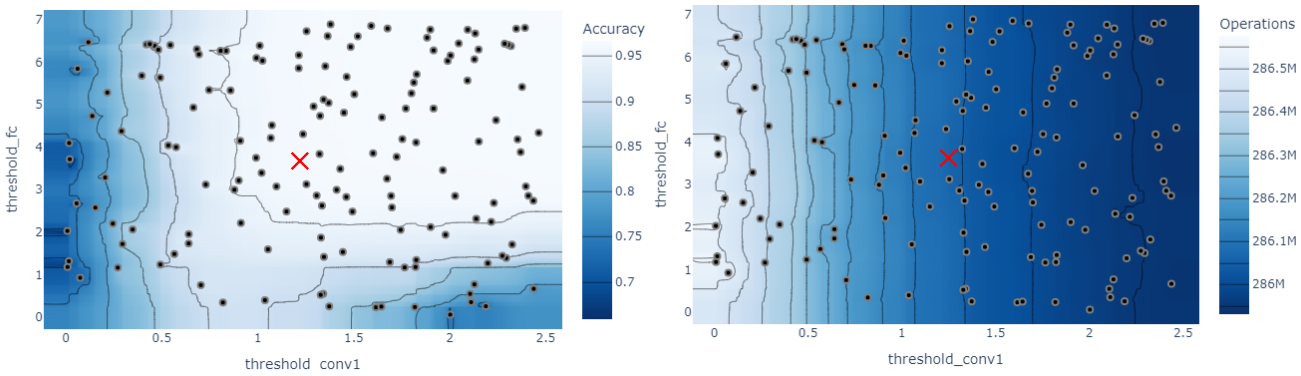
A pseudo-spiking equivalent is built and Figure 33 shows how it predicts that the validation accuracy is impacted by the number of steps.

5.4.2 Spiking Classification

Using the maximum values obtained from clamping, the theoretical thresholds $U_{th,conv}$ and $U_{th,fc}$ are provided in Equations 45 and 46.

$$U_{th,conv} = \frac{Y_{max}}{X_{max}} = \frac{1.2}{1} = 1.2 \tag{45}$$

$$U_{th,fc} = \frac{Z_{max}}{Y_{max}} = \frac{4.5}{1.2} = 3.8 \tag{46}$$



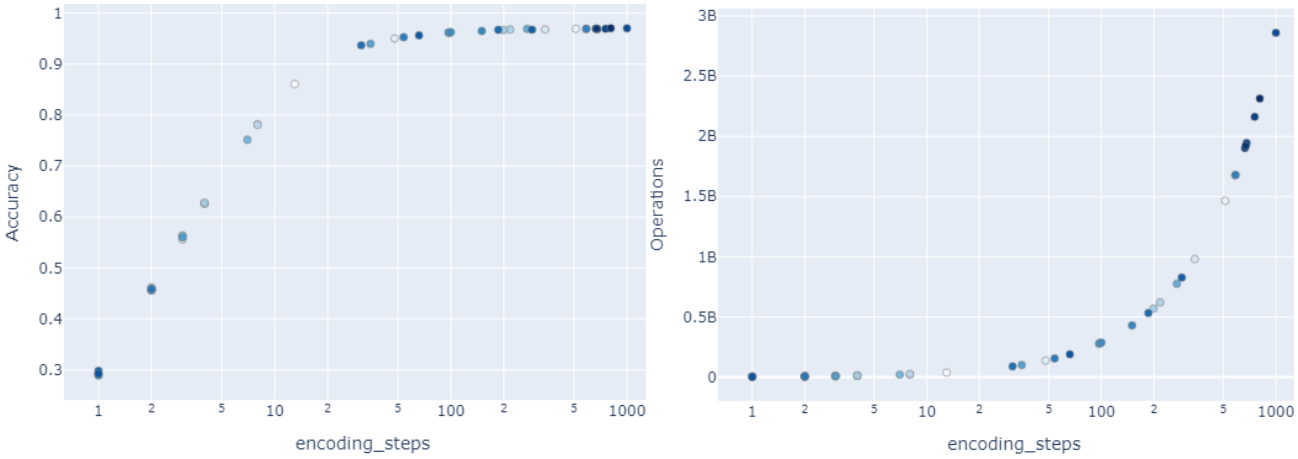
(a) Validation accuracy

(b) Number of add operations per input RD map

Figure 34: Threshold tuning for the spiking classification network using 100 encoding steps (threshold1 = $U_{th,conv}$ and threshold2 = $U_{th,fc}$), a zoomed out version can be found in Figure 52 in Appendix 7.4. The red crosses represent the theoretical thresholds: $U_{th,conv} = 1.2$ and $U_{th,fc} = 3.8$.

From these theoretical thresholds, it can be seen that normalising the input before training has the advantage that both thresholds are in the same order of magnitude. For example if $X_{max} = 1000$ then $U_{th,conv}$ would have been three orders of magnitude smaller. To counter this, the weights would have also needed to be three orders of magnitude smaller. If fixed-point numbers are used to represent the values, then both very large numbers for the input and very small numbers for the weights should be representable. Many more bits would consequently be necessary.

A threshold tuning is performed once more to prove that the theoretical thresholds are meaningful. Figure 34 shows how the validation accuracy and the number of operations is impacted by the thresholds of the convolutional layer and the fully connected layer when 100 steps are used (Figure 49b in Appendix 7.4 provides the corresponding Pareto curve). The pseudo-spiking curve in Figure 33 predicts a validation accuracy of 96.2% and the accuracy obtained for the theoretical thresholds is 96.1%. Similar conclusions to those given in Sub-section 5.2.2 can be made, however, the following can be added.



(a) Validation accuracy

(b) Number of add operations per input RD map

Figure 35: Number of operations - error trade-off of the spiking classification network using using $U_{th,conv} = 1.2$ and $U_{th,fc} = 3.8$

The thresholds can be much larger than the theoretical thresholds. Figure 52 in Appendix 7.4 shows that they can be as large as $U_{th,conv} = 25$ and $U_{th,fc} = 40$. A gesture class is predicted by the neuron spiking the most, however, this does not mean it has to spike a lot, only more than the others. While using the theoretical threshold will create an output representing the output from the non-spiking classification, this is not an end on its own. Nevertheless, Figure 34b shows that the differences in the number of operations that can be obtained by varying the thresholds is not significant. This means that in the following, the theoretical thresholds will be used.

The next step is to characterise how the validation accuracy and the number of operations are impacted by the number of encoding steps. Those are provided in Figure 35. The evolution of the accuracy is nearly the same as in Figure 33 which proves the relevance of the pseudo-spiking network.

In Figure 18, the current section has found the following:

- the number of encoding steps for a certain validation accuracy of the classification network is predicted in Figure 33,
- the classification theoretical thresholds are validated as $U_{th,conv} = 1.2$ and $U_{th,fc} = 3.8$.

5.5 Fourier-Transform and Classification Co-optimisation

In the previous Sections 5.2 and 5.3, the FT and the classification neural networks were optimised individually from each other. This was to reduce the number of parameters to be optimised at once. However, this avoids a significant area of optimisation since the type of RD map transferred to the classification has not been optimised on. This section hence focuses on the trade-off between the computational costs of both algorithms.

Sub-section 5.5.1 analyses how the number of range samples can be sub-sampled before applying the FTs while Sub-section 5.5.2 observes how the pixels fed into the FTs can be pre-cropped so that the RD only contains those that are processed in the classification network.

5.5.1 Sub-sampling

Equation 7 shows that the maximum range that can be observed in the RD map is directly proportional to the sampling frequency. However, [12] states that no gestures are performed beyond 4.5 meters. While the strategy in Section 5.3 is to crop between the FT and the classification to reduce the input size to the classification architecture, the chirp-sample map could be cropped before the FT. By using $d_{max} = 4.5$, the following shows that the necessary sampling frequency is $1.1MHz$ (Equation 47) which is 9.2% of the initial sampling frequency ($12MHz$). As a consequence, it is expected that keeping about 1 in 10 range samples would be equivalent to cropping after the FT.

$$f_s = \frac{4B}{cT_c} \cdot d_{max} = \frac{4 \cdot (750MHz)}{(3 \cdot 10^8 m/s) \cdot (40.96\mu s)} \cdot (4.5m) = 1.1MHz \tag{47}$$

Figure 36 tests whether this hypothesis holds on the architecture defined in Table 12. The chirp-sample maps are first subsampled [7] in the range dimension, then both FT are performed and the resulting RD maps are fed into the classification network. Subsampling by a factor of 10 reaches 94.4% validation accuracy which is in the same order of magnitude compared to when the RD map was cropped after the FT. It is probable that with a further tuning of other parameters such as regularisation terms, this accuracy could have reached 97.0% as well.

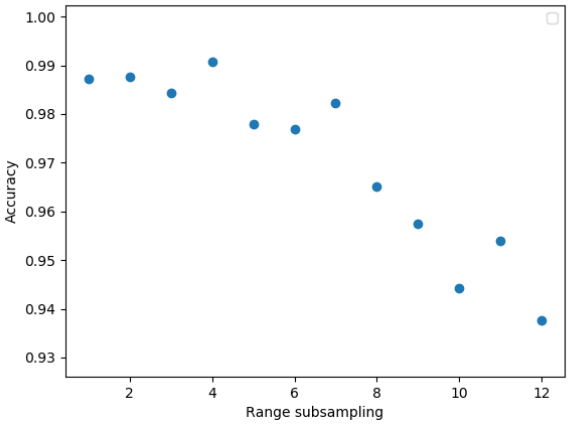


Figure 36: Validation accuracy for different range bins subsampling. Subsampling is defined by selecting 1 in 'subsampling' range bins.

Subsampling fewer than that increases the validation accuracy above that reached with all previously tested networks, which is surprising since the initial ResNet had shown no accuracy drop when the initial RD map had been cropped to the maximal distance of $4.5m$. A hypothesis for this is that the signals appearing beyond $4.5m$ give extra information about the gesture while not being the gesture itself, for example, reflections from the gesture.

In terms of the number of output pixels, it would have been equivalent to crop (hence cropping more range bins) without subsampling first. However, the advantage of subsampling is to reduce the size of the input chirp-sample map which helps reduce the memory needs. Nevertheless, this is not investigated further since it only leads to a small memory gain by reducing the size of the input map, and that it would require the full network to be re-trained.

5.5.2 Cropping

Instead of sub-sampling, or on top of sub-sampling, it is also possible to pre-crop the RD maps. In Figure 26b, the RD maps are cropped before they are injected into the classification. However, all pixels that are cropped have been calculated for no reason, hence wasting energy and memory. Figure 37 shows how to keep only the parts of the maps that are useful. Starting from the second matrix in Figure 37c, the red area represents the

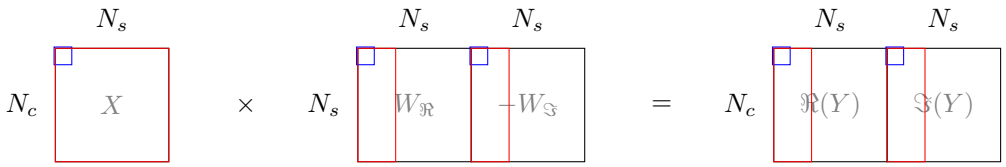
[7] Subsampling refers to selecting in 1 in x frequency bins which is equivalent to reduducing the sampling frequency of the signal by a factor x.

parts of the real and imaginary parts of the RD map that are kept after cropping. The blue rectangle represents which part of the map represents positive range and positive velocities. The first matrix in Figure 37c is the matrix before shifting the $0Hz$ frequencies to the middle of the image, these areas can be followed backwards through the FT to select the parts of the maps and weights that are needed, all of these are surrounded by red rectangles.

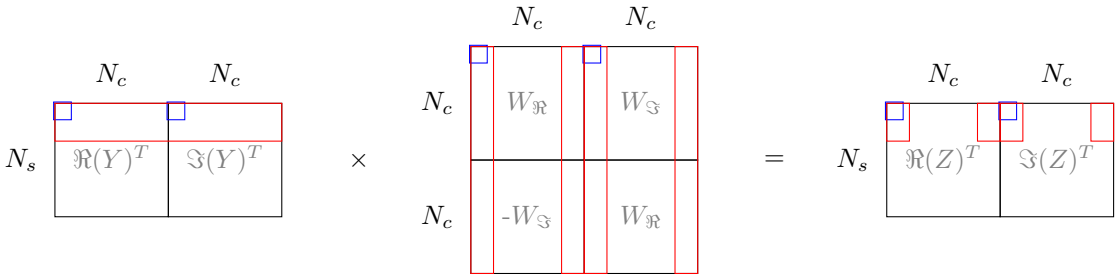
In Sub-section 5.3.2, it has been found that the Doppler bins can be cropped by 400 pixels in the high and low frequencies. Furthermore, only 23 of the Range bins are necessary. Table 15 shows the number of values to be saved in the case where the map is pre-cropped with those values in comparison to when it is not, the result is that only 17.6% of the memory is needed.

	Chirp-sample map	Range weights	Range map	Doppler weights	RD map	Total
No pre-cropping	491 520	524 288	983 040	3 686 400	983 040	6 668 288
With pre-cropping	491 520	23 552	44 160	614 400	3 726	1 177 358

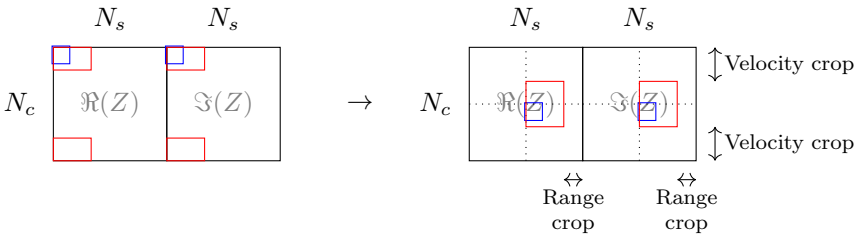
Table 15: Number of values to be saved for the pre-cropped FT in comparison to when the RD map is cropped afterwards.



(a) Matrix Range Fourier transform, representative of Equation 33.



(b) Matrix Doppler Fourier transform representative of Equation 34.



(c) Frequency shift: the $0Hz$ frequencies are shifted towards the centre in both the range and Doppler dimensions.

Figure 37: Visual representation of the Range-Doppler Fourier transforms with cropping: the blue square represents the position of the origin ($0Hz$ frequency for the range and Doppler bins), it is directed towards the positive directions, the red rectangles represent the necessary pixels to consider to only obtain the cropped area in the final RD map.

In Figure 18, the current section has found the following:
- updated range and velocity cropping.

5.6 Quantisation

Section 5.2 reduced the number of elements to save when generating the RD map and Section 5.3 attempted to reduce the complexity of the classification network, all in an attempt to decrease the memory requirements of the gesture-recognition pipeline. However, the previous calculations have been done using 32-bit floating point numbers, the final step is hence to lower this footprint further by using quantisation, the following sections first investigate how to quantise the FTs, then the classification network, and finally verifies the accuracy obtained by the quantised pipeline.

The following sub-sections use power-of-two quantisation since it is the most efficient in hardware. The quantisation schemes are applied with per-tensor granularity since the search space is smaller.

This section analyses how this quantisation can be applied to the FTs in Sub-section 5.6.1 and to the classification network in Sub-section 5.6.2. Sub-section 5.6.3 provides some quantisation schemes for the complete non-spiking pipeline.

5.6.1 Fourier Transforms

A tuning process is applied to the non-spiking FT network by selecting varying bit quantities for different network components. The total bit count is determined by summing the number of bits across the following components:

- the quantised input chirp-sample-map,
- the quantised range weights and biases,
- the quantised range map,
- the quantised Doppler weights,
- the quantised RD map.

Figure 38 shows the validation accuracy obtained when the quantised RD map are passed along in the non-spiking un-quantised classification network. Table 16 shows an example of quantisation without any accuracy loss in its second line, and the third line gives a quantisation where some accuracy is traded against a higher memory saving.

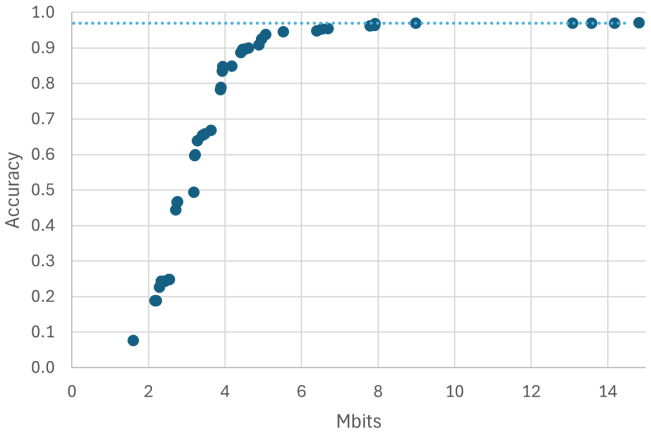


Figure 38: Validation accuracy reached for the non-spiking FT for fixed-point quantisation with different numbers of bits for the weights and memory maps, the dotted line represents the 97.0% validation accuracy reached on the un-quantised network.

Precision	Range layer		Doppler layer	Input	Range FT output	Doppler FT output	Total	Relative total [8]	Validation accuracy
	weights	biases	weights						
Reference	32	32	32	32	32	32	37.8M	100.0%	97.0%
Higher	12	32	11	12	12	14	13.6M	36.0%	97.0 %
Lower	7	32	6	7	12	14	7.9M	21.0%	96.8%

Table 16: FT quantisation, the first line represents the 32-bit floating point baseline, and the 'higher' and 'lower' precision represent fixed-point quantised representations.

Quantising the spiking FT is similar with the exception that the outputs of the spiking neurons only hold 2 bits (since it should be able to represent ± 1). The membrane potential is also kept un-quantised since the quantisation library Brevitas [27] is not well defined for spiking neurons, however, this could be considered in future work. The same quantisation is kept for the spiking network as those given in Table 16. It is probable that those are not optimal, but it would require a quantisation tuning for each number of encoding steps. An example of a spiking quantisation tuning is given in Section 5.7.

[8] Relative to the non-spiking equivalent 32-bit floating-point architecture

5.6.2 Classification

In a similar manner to what has been done for the FT, the following can be quantised:

- the quantised input RD map,
- the quantised convolutional weights and biases,
- the quantised convolutional layer output,
- the quantised FT weights and biases,
- the quantised FT output.

Figure 39 shows how the validation accuracy increases with increasing memory footprint. The blue curve represents the non-spiking network and the blue dotted line is the 97.0% validation accuracy for full-bit precision. Table 17 shows in its second line that using 32.6% of the memory footprint maintains the same validation accuracy as the non-quantised network, however the third line shows that with a loss of 1% accuracy this can be reduced to up to 16.0%. The same comment about spiking quantisation that has been made on the FT can be made for the classification network.

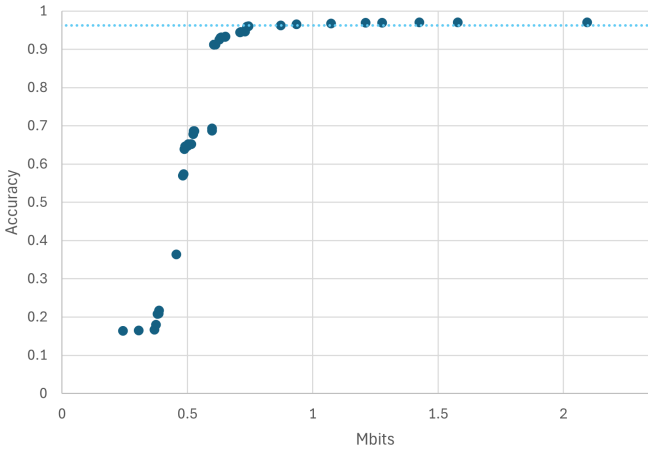


Figure 39: Validation accuracy reached for the non-spiking classification for fixed-point quantisation with different number of bits for the weights and memory maps, the dotted line represents the 97.0% validation accuracy reached on the un-quantised network

Precision	Conv. layer		FC layer		Input	Conv. output	FC output	Total	Relative total [9]	Validation accuracy
	weights	biases	weights	biases						
Reference	32	32	32	32	32	32	32	4.6M	100%	97.0%
Higher	11	32	9	16	7	9	10	1.5M	32.6%	97.0%
Lower	5	32	5	16	6	9	10	744k	16.0%	96.0%

Table 17: Classification quantisation, the first line represents the 32-bit floating-point baseline, and teh 'higher' and 'lower' precision represent fixed-point quantised representations (selected from Figure 39).

5.6.3 Full Non-spiking Pipeline Quantisation

Table 18 summarises the memory gains achieved in the previous sections. The second line shows the non-spiking pipeline with the simplified architecture (Section 5.3) and the discrete FT (Section 5.2.1). The third line provides the memory requirements when the pre-cropping is applied on the FT (Section 5.5.2), and the fourth line gives the requirements when both FT and classification are quantised in a way that does not impact the accuracy. This is feasible since the number of bits at the output of the FT is higher than that needed at the input of the classification network. The fifth line gives an example of sacrificing some accuracy and saving memory by quantising more than in the previous case.

[9] Relative to the non-spiking equivalent 32-bit floating-point architecture

Network types	FT bits		Classification bits		Total	Relative	Validation accuracy
	Absolute	Relative	Absolute	Relative			
Non-spiking replication from [12] [10]	Built-in function		360M-1.1G	-	-		98.5%
Non-spiking with the simplified classification	213.3M	100%	4.6M	100%	217.9M	100%	97.0%
Non-spiking with pre-cropped map	37.8M	17.7%	4.6M	100%	42.4M	19.5%	97.0%
Non-spiking quantised higher precision	13.6M	6.4%	1.5M	32.6%	15.1M	6.9%	96.9%
Non-spiking quantised lower precision	7.9M	3.7%	744k	16.0%	8.6M	4.0%	95.8%

Table 18: Memory requirement and validation accuracy for the full non-spiking pipeline using different memory-reducing strategies.

In Figure 18, the current section has found the following:
- quantisation schemes for the non-spiking FTs,
- quanitsation schemes for the non-spiking classification network.

[10] The line is given to obtain the scale of the size of the classification architecture. The lower bound is calculated by multiplying the number of weights from Table 10 by 32 (for 32-bit floating points). However, there is no straight-forward way to quantify the exact memory required since it depends on how the hardware saves the values between the layers. The upper bound is calculated by simply adding all values to be saved by using the sizes of the inputs from Table 22 which results in an extra 800Mbits. Due to this large range, and that the FT are only implemented as a built-in function, all following relative sizes are calculated relative to the second line of the table

5.7 Full Gesture Recognition Pipeline

The previous sections analysed how to design each step of a gesture-classification pipeline. To recap, they explained how to decouple the different hyper-parameters of the pipeline to investigate each one independently of the others by making use of the following:

- the non-spiking pipeline to define different quantisation parameters in Sub-sections 5.6.1 and 5.6.2,
- the pseudo-spiking pipeline to approximate the achieved validation accuracy/error that can be obtained for a certain number of encoding steps in Sub-sections 5.2.1 and 5.4.1,
- the theoretical thresholds using the theory explained in Sub-section 4.1.2. The analysis done in Sub-sections 5.2.2 and 5.4.2 has also shown that the number of operations is about as low as possible for those thresholds too.

This chapter focuses on merging the results to create the full spiking pipeline, illustrated in red in Figure 18. The following procedure is used to join the parameters listed above:

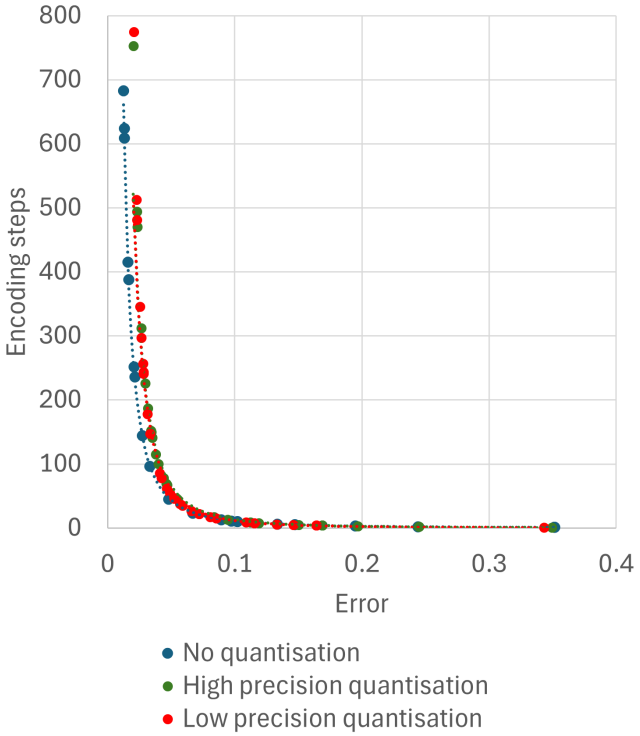


Figure 40: Function finding the number of encoding steps from the error of the pseudo-spiking RD map

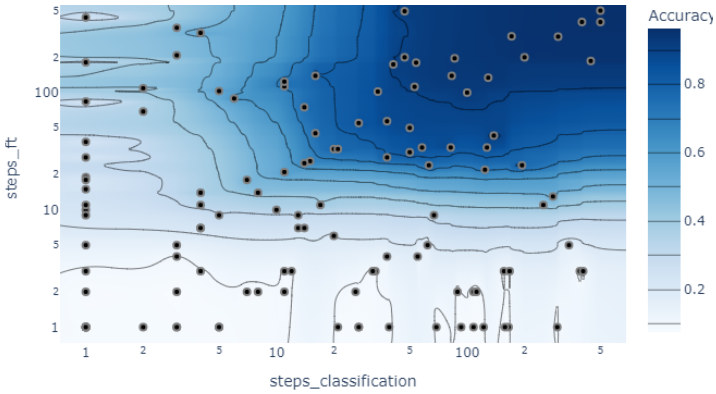
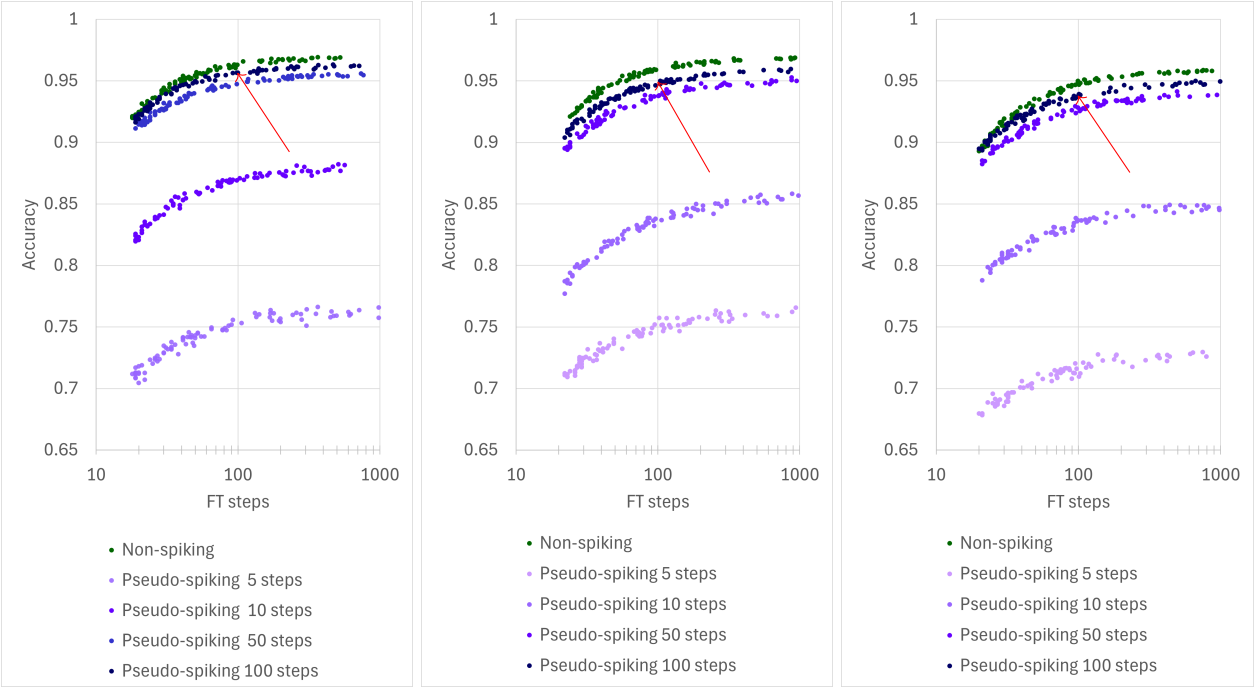


Figure 41: Spiking pipeline validation accuracy for the high-precision quantised network for a range of FT and classification encoding steps

1. a relationship between the number of steps and the error of the FT network is extracted from Figure 40. This is a similar graph to Figure 20, with the difference that it is updated by the maximum clamping of the input (1000) and different quantisation schemes, the relationship for these quantisation levels are modelled by the fitted exponential functions as follows:
   - no quantisation: time steps $= 0.1311 \cdot \text{error}^{-1.924}$,
   - high-precision quantisation: time steps $= 0.0920 \cdot \text{error}^{-2.137}$,
   - lower-precision quantisation: time steps $= 0.0854 \cdot \text{error}^{-2.128}$.
2. independently of step 1 above, random noise of increasing intensity is added to the input of the non-spiking classification networks, this noise is representative of the error caused by the spiking FT. The result is a validation accuracy-noise curve which can be transformed into validation accuracy-FT error using the relationships from step 1. Figure 42 shows this accuracy-error curve in green,
3. this noise addition is repeated for a certain number of encoding steps on the pseudo-spiking classification algorithm. Figure 42 shows examples with 5, 10, 50 and 100 steps. The green non-spiking curve is an upper bound to the validation accuracy that can be reached with the pseudo-spiking classification for a given FT error (x-axis),
4. the number of encoding steps and the quantisation type for the each of the FT and classification networks are chosen relative to the desired validation accuracy, the available memory as well as the acceptable latency. The aim here is not to assume that the spiking pipeline will perform as given by the pseudo-spiking pipeline, but to observe for which number of encoding steps it starts to saturate. When that happens, then the pseudo-spiking approximation is relevant,

5. the theoretical thresholds for both spiking networks are found, which enable the full spiking pipeline to be built.



(a) No quantisation (b) High-precision quantisation (c) Lower-precision quantisation

Figure 42: Graphs to help decide which number of FT encoding steps, classification encoding steps, and quantisation to choose for a given validation accuracy. The red arrows provide the predicted accuracy mentioned in Table 19

The above stated how to select the thresholds, number of encoding steps and the quantisation type in order to obtain the desired validation accuracy. Figure 41 shows how the validation accuracy evolves for the high-precision quantisation pipeline with varying number of encoding steps. As explained above, the expected pseudo-spiking validation accuracies can be trusted for encoding steps when the accuracy stagnates. For example, the beginning of stagnation can be taken when both FT and classification use 100 encoding steps: Figure 42b predicts a validation accuracy worth $\approx 94.7\%$ and the spiking network reaches 94.1%. This is given in the second line of Table 19, the other two lines provide the cases when the values are un-quantised and quantised to a lower precision.

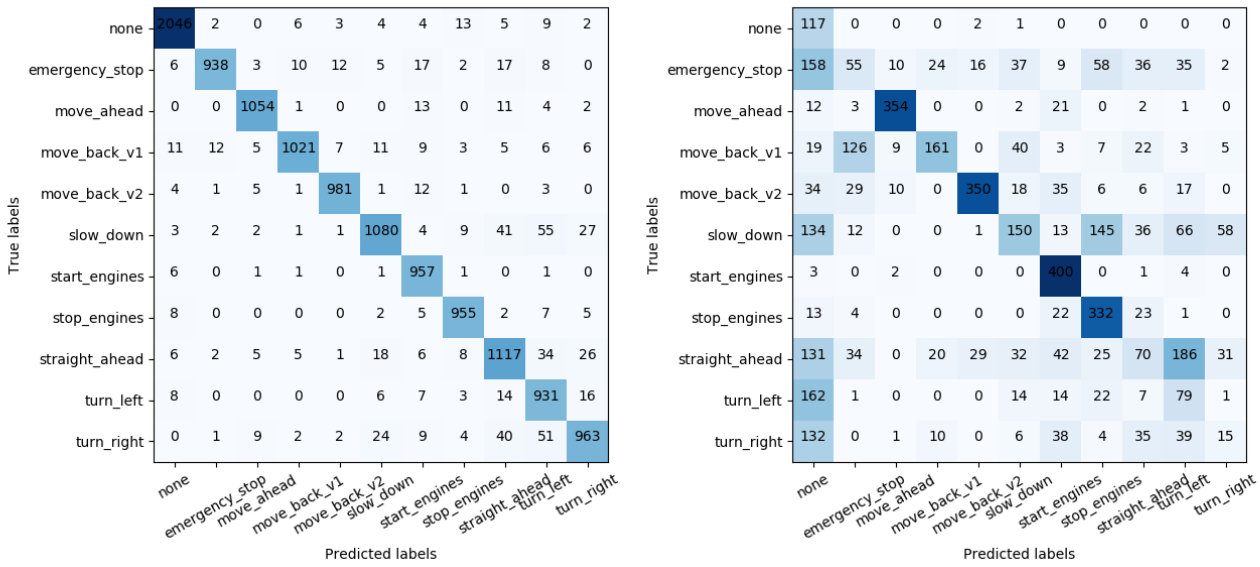
This table shows that there is a small discrepancy between the predicted validation accuracy and the obtained spiking validation accuracy. This is probably caused by the quantisation scheme that is based on the non-spiking network which does not model the behaviour of the quantised number interacting with the spiking neurons. To test this hypothesis, a hyper-parameter on the spiking quantisation parameters is achieved using 100 steps for both FT and classification. This reaches 94.5% for $12.4Mb$ (the precise parameters are in Tables 23 and 24 in Appendix 7.4). This is about the same validation accuracy compared to the expected accuracy for the high-precision pipeline but for even smaller memory requirements. This can be compared to the blue and green dots that represent the expected values from Table 19.

Figure 43a shows the confusion matrix for this specific case, and Figure 43b provides the test set's matrix. The 11-gestures test accuracy is 46.8% and the 5-best-gestures test accuracy is 72.8%. While the validation accuracy has been allowed to be reduced by 2.5% in comparison to the non-spiking un-quantised pipeline, the test accuracy reduced by 4% and the 5-best-gestures by 5%.

Overall, it is concluded that the method decoupling all hyper-parameters is efficient in reaching accuracies close to that of the non-spiking pipeline, where the accuracy gap could be decreased if the latency-memory-accuracy trade-off was shifted towards accuracy. While the test accuracy is quite lower to that from [12], it must be noted that this is a result of the simplification from a ResNet18 architecture to the simpler classification: although the validation accuracy barely dropped by 1.5%, the 11-gestures test accuracy decreased to 50.7%, most probably due to the large difference in distribution between the validation and test sets.

Network types	FT bits		Classification bits		Total	Relative [11]	Validation accuracy	Predicted validation accuracy
	Absolute	Relative [1]	Absolute	Relative [1]				
Spiking with pre-cropped map un-quantised	37.8M	17.7%	4.6M	100%	42.4M	19.5%	94.8%	95.6%
Spiking quantised high-precision	15.2M	7.1%	1.6M	34.8%	16.8M	7.7%	94.1%	94.7%
Spiking quantised lower-precision	9.6M	4.5%	812k	17.6%	10.4M	4.7%	91.6%	93.7%

Table 19: Memory requirement and validation accuracy for the full spiking pipeline using different memory-reducing strategies



(a) Validation set

(b) Test set

Figure 43: Confusion matrices for 100 encoding steps for the FT and classification networks.

[11] Relative to the non-spiking FT, classification and full-pipeline for the un-quantised non-spiking algorithms given in the first line of Table 18.

5.8 Hardware

The following section presents an example of a hardware implementation until the post-synthesis stage for a spiking and a non-spiking network. The aim is to prove the energy gains of the former relative to the latter. It presents a smaller case of the Range and Doppler FT algorithm and highlights different ways this HDL code should be modified for the hardware to truly represent the full pipeline. This smaller example contains an input chirp-sample map where $N_s = 3$ (number of samples) and $N_c = 3$ (number of chirps), with $T = 3$ (number of encoding steps) for the spiking case, the values are quantised with 8 fractional and 8 integer bits.

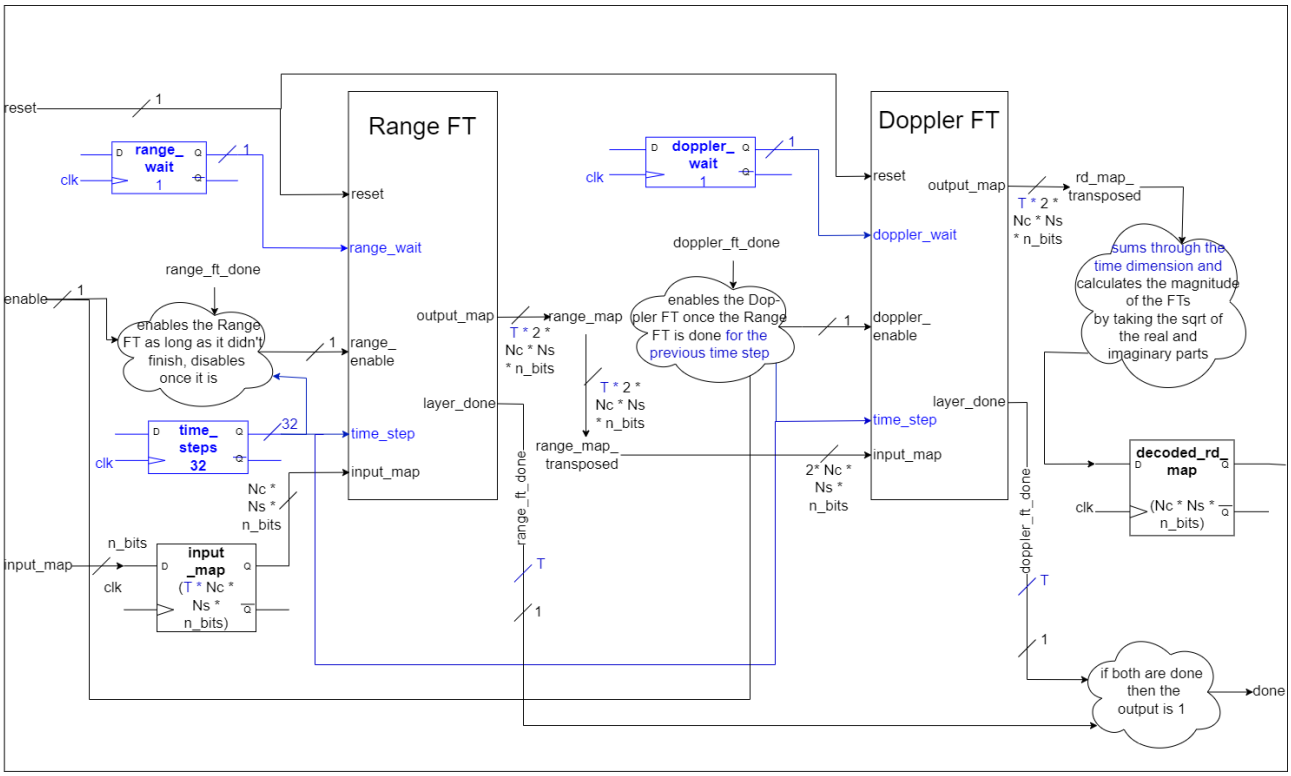


Figure 44: High-level hardware diagram for the spiking and non-spiking FTs, the blue parts are only present in the spiking hardware. The value `n_bits` represents the number of bits used to save the values in the input `input_map`, and all resulting maps, which is in the example case `n_bits` = 16. The example case also contains the following variables: $N_s = 3$, $N_c = 3$, and $T = 3$.

Figure 44 shows a high level hardware diagram of the FTs. This diagram only shows the signals required to calculate the RD map and removes all flags and signals used to write the weights and input map into registers, as well as those used to read the output RD map. Once the `reset` has been activated to initialise all neurons, the `enable` signal is activated to start calculating the FTs. The non-spiking case is quite straight-forward:

- the `Range FT` sub-block calculates the range map of the input map `input_map` and stores it in a local register,
- when the range FT is done, the flag `range_ft_done` is activated and the wire `range_map` contains the range map,
- the `Doppler FT` sub-block calculates the Doppler map of the transposed range map called `transposed_range_map` and stores it in a local register,
- when the Doppler FT is done, the flag `doppler_ft_done` is activated and the wire `rd_map_transposed` contains the transposed RD map,
- when both `range_ft_done` and `doppler_ft_done` are activated which activates the output flag `done`,
- the output RD map is calculated by taking the magnitude of the real and imaginary parts of the `rd_map_transposed`, it is then stored in the register `decoded_rd_map`.

For the spiking case, the following happens:

- `time_steps` has been initialised to 0,
- the `Range FT` sub-block calculates the range map of the first step of the map `input_map`,

- when the first step of the range FT is done, the first bit of the flag `range_ft_done` is activated and the wire `range_map` contains the range map of the first step,
- the `Doppler FT` sub-block calculates the Doppler map of the first step of `transposed_range_map`. Simultaneously, the `Range FT` sub-block calculates the range map of the second step of the input map `input_map`,
- when the first step of the Doppler FT is done, the first bit of the flag `doppler_ft_done` is activated, and when the first step of the range FT is done then the second bit of the flag `range_ft_done` is activated,
- the first step has been fully processed so `time_step` is incremented,
- this processed is repeated where the `Range FT` sub-block calculates the previous step and `Doppler FT` calculates the following one while incrementing `time_step`.
- both `range_ft_done` and `doppler_ft_done` are fully activated then all steps have been processed for both FTs which activates the output flag `done`,
- the output RD map is calculated by summing the spikes across the steps and taking the magnitude of the real and imaginary parts of the `rd_map_transposed`, it is then stored in the register `decoded_rd_map`.

Figure 45 shows more details about the hardware design for the FTs. There are two registers with the saved weights and biases. Furthermore, there are as many neurons as there are pixels in the output of the FT which is $2 \cdot N_c * N_s$ for both the range map and the RD map (as a reminder, this is illustrated in Figure 17). The address is incremented at each clock cycle to fetch a new element from the input `input_map` and a weight from the `weights` register.

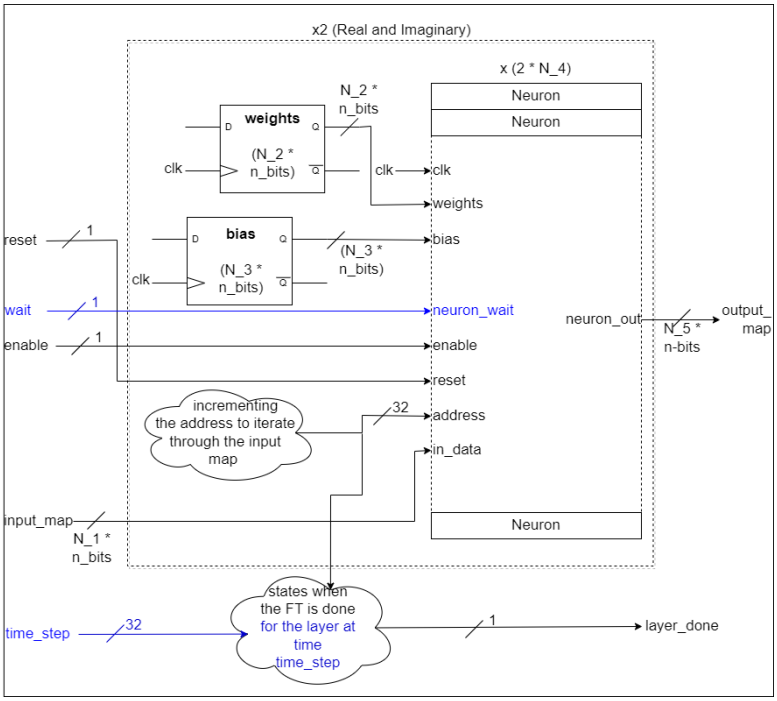


Figure 45: Hardware diagram for the FTs, the blue parts are only present in the spiking hardware. If this is the range FT: $N\_1 = N_c \cdot N_s$, $N\_2 = N\_3 = N\_4 = N\_5 = \cdot N_c \cdot N_s$. If it is the Doppler FT: $N\_1 = N\_2 = N\_3 = N\_4 = N\_5 = 2 \cdot N_c \cdot N_s$.

For the non-spiking case of the range/-Doppler FT, it requires one cycle to fetch the first weight, $N_s/2 \cdot N_c$ cycles to fetch the next weight and simultaneously perform the MAC operation, and a remaining two cycles to write the element in a register and to activate the `done` flag. This is illustrated in the computation of the total number of clock cycles given in Equation 48. This equation also shows that there needs to be one initial cycle for initialisation in the sense that it notices that the `enable` flag of Figure 44 is activated.

The spiking case is very similar with the difference that the `time_step` variable selects the input map at the right step to perform the FT. Furthermore, the `wait` variable is a flag used to halt the calculation. This is useful when one of the range or Doppler FT takes fewer time than the other and it must pause so that both FTs only differ by one time step. The number of clock cycles for T time steps is provided in Equation 49. It shows that the first time step requires $1 + N_s + 3$ cycles. The first cycle is required to extract the first weight, the N_s next ones computes the current by adding the weights when there is an incoming spike, and the following three cycles are used to add the current to the membrane potential, verify if it is greater than the threshold and if it is write the spike to the output map. Performing the Doppler FT for one step follows the same logic and requires $1+2N_c+3$ cycles. For the following $T-1$ steps, the range and Doppler FT overlap which means that it will take as long as the longest of the two. When over, there is still one final time step to calculate for the Doppler FT of the last time step.

$$N_{non-spiking} = \underbrace{1}_{\text{Initialise}} + \underbrace{(1 + N_s + 2)}_{\text{Range FT }^{12}} + \underbrace{(1 + 2N_c + 2)}_{\text{Doppler FT }^{12}} + \underbrace{1}_{\text{Update}} \tag{48}$$

$$N_{spiking} = \underbrace{1}_{\text{Initialise}} + \underbrace{(1 + N_s + 3)}_{\substack{\text{Range FT }^{13} \\ \text{first step}}} + \underbrace{(T-1) \cdot \max(N_s + 4, 2N_c + 4)}_{\substack{\text{The faster between the Range} \\ \text{and Doppler FT }^{13}\text{ waits} \\ \text{for the slower one}}} + \underbrace{(1 + 2N_c + 3)}_{\substack{\text{Doppler FT }^{13} \\ \text{last step}}} + \underbrace{1}_{\text{Update}} \tag{49}$$

For more details about the hardware description of the spiking and non-spiking neurons, their schematics are in Figure 53 in Appendix 7.5. Furthermore, the final state machines are available in Figures 54, 56 and 55.

The two hardware designs are synthesised to extract their power and area requirements, which are summarised in Table 20, more details about the power distribution is summarised in Table 25 in Appendix 7.5. The constraints used are highlighted in Appendix 7.5, and the designs are assumed to be clean since all slacks are positive.

		Non-spiking	Spiking	Spiking/Non-spiking
Total power (μW)	No clock gating	1063.5	834.4	0.784
	With clock gating (94.1%)	495.9	181.2	0.365
Total area (μm^2)	Total	40 734	26 340	0.647
Number of clock cycles		17	39	2.17
Energy estimation if 100 clock cycles are used (nJ)		0.8	18.3	22.9

Table 20: Non-spiking vs spiking hardware specifications

Table 20 shows that the spiking hardware's power requirements are a third of those of the non-spiking one when clock gating is used. Clock gating is much more efficient on the spiking network since the neurons' outputs only change when they spike. The total area is also smaller for the spiking case since both hardware have the same memory requirements for the weights and biases but lower requirements for the input map, the range map and the RD map since their values are binary. When scaling to more encoding steps, the power should not vary much since the same operation is just repeated for longer. The area will increase since the input map, the range map and the RD map sizes are proportional to the number of steps.

Table 20 shows an energy estimation for the non-spiking case $E_{non-spiking}$ which is calculated in Equation 50 where P is the power, t is the total time and T_{clk} is the time per clock period. As proven in Section 5.7, using about 100 steps results in a validation accuracy of 94.5%. An energy estimation for 100 steps is provided in Equation 51: the spiking network uses 22.9× the energy of the non-spiking one.

$$E_{non-spiking} = P \cdot t = P \cdot T_{clk} \cdot N_{non-spiking} = (495.9\mu W) \cdot (100ns) \cdot (17) = 0.8nJ \tag{50}$$

$$E_{spiking} = P \cdot t = P \cdot T_{clk} \cdot N_{spiking} = (181.2\mu W) \cdot (100ns) \cdot (1009) = 18.3nJ \tag{51}$$

This large energy requirement can be caused by multiple factors. First, the example input map contains a high percentage of 1s relative to 0s. This is because it was initialised to contain random 1s and 0s while a real input map would contain a higher percentage of 0s. This is because the distribution of values is skewed towards smaller numbers as shown in Figure 21, as a result the rate encoded values will contain fewer spikes. In consequence, the switching power should decrease.

Another cause is that the combinational logic calculates some values continuously such as the output RD map from the real and imaginary part. If more conditions were used in the design to only compute useful values, the internal power could be reduced.

For ease of implementation and to access all weights in parallel, registers have been used for this first hardware exploration. Toward practical deployment and scalability, future work will investigate the use of SRAM instead. Furthermore, to implement the classification, the fully-connected layer can be designed in the same way with a modified version to represent the convolutional layer.

[12] fetch first weight + (compute current and fetch next weight) +(write map and set flag to done))

[13] fetch first weight + (compute current and fetch next weight) + (add to membrane + spike + write map))

6. Conclusion and Future Work

6.1 Conclusion

This work describes how to implement a gesture classification algorithm based on radar data, and how to transform it into its spiking neural network form. It highlights the different trade-offs to be optimised and exemplifies them on an air-marshalling gesture dataset.

The pipeline contains two parts to be designed: a double FT implementing first a range and then a Fourier transform that transforms the pre-processed map into a RD map, which is followed by a classification algorithm that extracts the gesture from the RD map (Figure 2). The first step of the procedure is to optimise the non-spiking pipeline:

1. the work done in [12] is taken as a reference, the replicated pipeline reaches 98.5% validation accuracy and 59.8% 11-gestures test accuracy, this is similar to the 59.1% validation accuracy obtained by [12],
2. the FTs are initially built-in Python Fourier transforms, they are first modelled into an ANN and then simplified to only process the distance and velocity frequency bins that are required by the classification,
3. the classification network is initially a ResNet18, it is simplified to only contain a convolutional and a fully-connected layer while trading a 1.5% validation accuracy loss. There is a much higher cost on the test accuracy which drops by 11.7% as shown in the 4^{th} line of Table 10,
4. the FTs and classification networks are joined to generate the full pipeline with accuracies provided in the 5^{th} line of Table 10: the validation accuracy is 97.0% and the 11-gestures test accuracy is 50.7%,
5. superfluous areas of the RD map are removed and the networks are quantised to fixed point numbers to reduce the memory requirements as highlighted in Table 18.

The second step is to adapt the non-spiking pipeline to its spiking equivalent. A spiking implementation has more parameters to tune that are interdependent to some extend: the neurons' thresholds, the number of encoding steps, the quantisation schemes, the network weights. This work has decoupled the previous parameters so that they could be tuned independently of one another:

1. the non-spiking weights are used,
2. the thresholds are mathematically defined so that the number of spikes emitted corresponds to the non-spiking equivalent output of these neurons in the optimal rate-encoded manner, given the specified number of time steps.
3. the pseudo-spiking graphs are used to estimate the number of encoding steps required, and the non-spiking quantisation scheme is used to estimate the number of bits required, the estimations obtained are illustred in Figure 42,
4. the quantisation scheme is fine-tuned for the number of encoding steps chosen, the spiking results are provided in Table 19.

An example of the trade-off achieved for the spiking pipeline reaches a validation accuracy of 94.5% and an 11-gestures test accuracy of 46.8% when it is implementated with 100 steps for both FTs and classification while using only 5.7% of the memory of the initial non-spiking un-quantised and un-simplified pipeline. Comparing with the second line of Table 10 which is the replicated pipeline from [12], the validation accuracy drops by 4%. The 11-gestures test accuracy drops by 12.3% and the 5-best-gestures test accuracy drops by 14.1%. The large gap between the validation and test accuracy losses is a consequence of the difference in distribution between the train/validation sets and the test set. This causes the simplified architecture to perform significantly worse on the test set although it performs nearly as well on the validation set compared to the baseline ResNet18.

However, these accuracies, latency and memory needs can be shifted towards lower latency, lower memory or higher accuracy according to the desired requirements. In theory, the elimination of MAC operations should save on energy by considerably reducing the power so that the added latency is compensated, an example of such results have been obtained by [11] which achieves more than 30× power reduction as has been highlighted

in the introduction. This work presents a hardware exploration which reaches only a power reduction of 3×. When the latency is factored in, the energy is therefore still higher for the spiking version.

6.2 Future work

From the spiking implementation, several aspects could be improved or modified. The following lists some ideas:

- there is a poor generalisation on the test set when simplifying the ResNet18 architecture into its simpler form. This was not visible when choosing hyper-parameters based on the validation set since the validation set and test set come from different distributions as shown in Figure 48 in Appendix 7.4. Different generalisation strategies could have been used. For example the scale of the dataset could have been increased by applying transformations to attempt to increase the accuracy (in the same way as [26]),
- the simplified classification network uses square kernels since this reduces the number of parameters to be tuned. However, since the chirp-sample map input is very rectangular ($N_c = 160$ and $N_s = 23$) it might be more efficient to use rectangular kernels. It is probable that those kernels would be smaller, hence reducing the number of weights to save,
- the implemented pipeline requires to decode and encode between the FT and the classification. This is because the absolute value of the RD cannot be calculated from the coded real and coded imaginary parts. It would be interesting to investigate an approximation to this absolute value equation to avoid decoding and re-encoding to save on energy,
- the current work used rate-encoded data since it leads to higher accuracies, but it would be interesting to implement the pipeline using latency encoding. For example, [28] implements a latency-encoded FT, which demonstrates how to only use 1 spike per value while obtaining low error compared to the non-spiking version,
- the weights in the FT are pre-defined which means that only an ANN-SNN conversion can be used. However, the weights of the classification could be trained. It is possible that using BPTT to train the spiking network would require fewer encoding steps for the same accuracy,
- the hardware exploration should be developed to use a full size input map and about 100 time steps to prove the energy gains of a SNN in comparison to its equivalent ANN.

7. Appendix

7.1 Intermediate Frequency Calculations

The mixing of the transmitted signal $x_T(t)$ from Equation 1 with the received signal $x_R(t)$ from Equation 3, can be expanded as follows.

$$\begin{aligned}
x_T(t)x_R(t) &= \cos\left(2\pi f_c t + \pi\frac{B}{T_c}t^2\right)\cdot\alpha_{att}\cos\left(2\pi f_c(t-t_d)+\pi\frac{B}{T_c}(t-t_d)^2\right)\\
&= \frac{\alpha_{att}}{2}\Bigg[\cos\left(2\pi f_c t + \pi\frac{B}{T_c}t^2 + 2\pi f_c(t-t_d)+\pi\frac{B}{T_c}(t-t_d)^2\right)\\
&+\cos\left(2\pi f_c t + \pi\frac{B}{T_c}t^2 - 2\pi f_c(t-t_d)-\pi\frac{B}{T_c}(t-t_d)^2\right)\Bigg]\\
&= \frac{\alpha_{att}}{2}\Bigg[\cos\Bigg(2\pi\Big(2\underbrace{f_c}_{10^9} - \underbrace{\frac{B}{T_c}t_d}_{\substack{\frac{10^6}{10^{-6}}\cdot 10^{-6}\\=10^6}}\Big)t + 2\pi\underbrace{\frac{B}{T_c}}_{\substack{\frac{10^6}{10^{-6}}\\=10^{12}}}t^2 + \pi t_d\Big(\frac{B}{T_c}t_d - 2f_c\Big)\Bigg)\\
&+\cos\left(2\pi\underbrace{\frac{B}{T_c}t_d}_{\substack{\frac{10^6}{10^{-6}}\cdot 10^{-6}\\=10^6}}t + \left(2\pi f_c t_d - \pi\frac{B}{T_c}t_d^2\right)\right)\Bigg]
\end{aligned}$$

The orders of magnitudes under the values are obtained from Section 5.1.1: t and t_d are the same magnitude as T_c since $t, t_d \in [0, T_c]$. This shows that the first cosine contains frequencies much higher than those of the second cosine, and when a low-pass filter is applied, it removes this first cosine. This outputs the IF signal $r(t)$.

$$r(t) = LPF\big[x_T(t)x_R(t)\big] = \frac{\alpha_{att}}{2}\cos\left(2\pi\frac{B}{T_c}t_d t + \left(2\pi f_c t_d - \pi\frac{B}{T_c}t_d^2\right)\right)$$

Range

The following shows that, with the same orders of magnitude, the equation is further simplified by observing that the last parameter is much smaller than the two others.

$$\begin{aligned}
r(t) &= \frac{\alpha_{att}}{2}\cos\left(2\pi\underbrace{\frac{B}{T_c}t_d}_{\substack{\frac{10^6}{10^{-6}}\cdot 10^{-6}\\=10^6}}t + \Big(2\pi\underbrace{f_c t_d}_{\substack{10^9\cdot 10^{-6}\\=10^3}} - \pi\underbrace{\frac{B}{T_c}t_d^2}_{\substack{\frac{10^6}{10^{-6}}\cdot(10^{-6})^2\\=10^0}}\Big)\right)\\
&\approx \frac{\alpha_{att}}{2}\cos\left(2\pi\underbrace{\frac{B}{T_c}t_d}_{f_R}t + \underbrace{2\pi f_c t_d}_{\Phi_R}\right)
\end{aligned}$$

The previous equation demonstrates that the frequency f_R is proportional to the delay t_d, and hence proportional to the distance (according to Equation 5), which means that the spectrum of one chirp shows peaks at the frequencies corresponding to the distances between the radar and objects.

Velocity

In the situation where several chirps are collected, each chirp can be denoted by its subscript i. In this case, orders of magnitude are again used to simplify the equation.

$$\begin{aligned} r_i(t) &= \frac{\alpha_{att}}{2}\cos\left(2\pi\frac{B}{T_c}t_{d,i}t + 2\pi f_c t_{d,i}\right) \\ &= \frac{\alpha_{att}}{2}\cos\left(2\pi t_{d,i}\Big(\underbrace{\frac{B}{T_c}t}_{\substack{\frac{10^6}{10^{-6}}\cdot 10^{-6} \\ =10^6}} + \underbrace{f_c}_{10^9}\Big)\right) \\ r_i(t_{d,i}) &\approx \frac{\alpha_{att}}{2}\cos\left(2\pi t_{d,i} f_c\right) \end{aligned}$$

The delay is expressed relative to a reference delay $t_{d,0}$ which is the delay for the first chirp. If the object moves between two chirps, then the delay will change from one chirp to the next by time $\Delta t_{d,i}$. If the object accelerates, then $\Delta t_{d,i}$ is variable, however, if the velocity is assumed constant between two chirps, then $\Delta t_i = \Delta t \ \ \forall i$ and $t_{d,i} = \Delta t_d \cdot i$. This provides a new frequency f_D that is observed between the chirps.

$$\begin{aligned} r_i(t_{d,i}) &= \frac{\alpha_{att}}{2}\cos\left(2\pi t_{d,i} f_c\right) \\ &= \frac{\alpha_{att}}{2}\cos\left(2\pi (t_{d,0} + \Delta t_{d,i}) f_c\right) \\ r_i &= \frac{\alpha_{att}}{2}\cos\left(2\pi \underbrace{f_c \Delta t_d}_{f_D}\cdot i + \underbrace{2\pi t_{d,0} f_c}_{\Phi_D}\right) \end{aligned}$$

This frequency f_D can be related back to the velocity of the object using $v = \frac{\Delta d}{T_{c,diff}}$ since the velocity is equivalent to the extra distance of the signal between two chirps by the time that occurred between those chirps.

$$\begin{aligned} f_D = f_c \Delta t_d = \underbrace{f_c}_{\approx \frac{c}{\lambda}} \frac{2\Delta d}{c} &\approx \frac{2\Delta d}{\lambda} \\ v = \frac{\Delta d}{T_{c,diff}} &= \frac{\lambda}{2T_{c,diff}} f_D \end{aligned}$$

7.2 Spiking Neural Network Thresholds - 2nd Layer and Above

Figure 46 provides an adapted version of Figure 14 for the second layer of the neurons in Figure 13. The logic developped holds for all layers that follows the second one since the input (Y_j in the example of the second layer) would have also passed first into the previous layer's ReLU, hence setting its minimum value to 0.

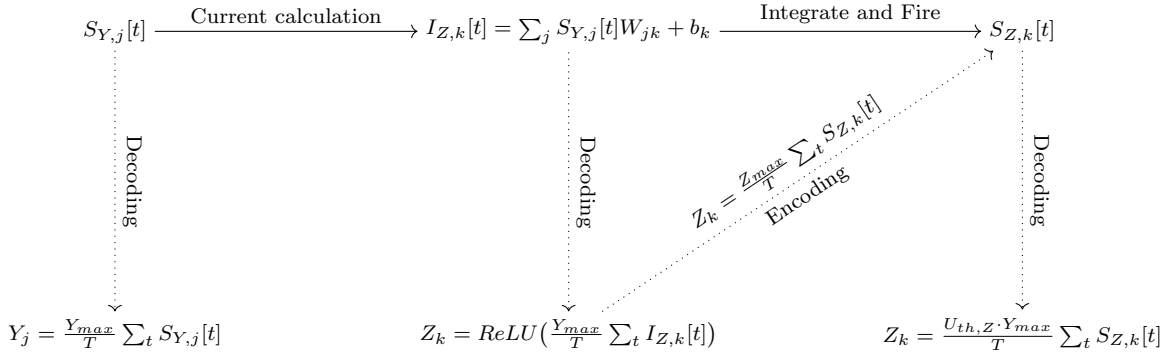


Figure 46: Diagram showing the relations between the different steps of an I&F spiking neuron and its non-spiking equivalent for the second layer. The top part represents the spiking operations, and the bottom part represents the non-spiking equivalents that can be obtained throught the vertical dotted "Decoding" lines. The "Encoding" line shows how the non-spiking Z can be rate encoded to the spikes $S_Z[t]$

7.3 ResNet18 Parameters

Tables 21 and 22 provide the number of parameters per layer and the sizes of the outputs of each layer for a ResNet18 architecture given in Figure 47 when it is fed with the un-cropped RD map. These tables are given to provide a comparison with the simplified architecture.

Layer		Conv	Maxpool	Block 1				Block 2				Block 3				Block 4				Avgpool	FC
				1.1		1.2		2.1		2.2		3.1		3.2		4.1		4.2			
				Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Czonv	Conv	Conv	Conv	Conv	Conv		
Parameters	Kernel size	7	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	Adaptive [8]	512 to 11
	Padding	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
	Stride	2	2	1	1	1	1	2	1	1	1	2	1	1	1	2	1	1	1		
	Output Kernels	64	64	64	64	64	64	128	128	128	128	256	256	256	256	512	512	512	512		

Table 21: ResNet18 parameters per layer [29]

	Input	Conv	Maxpool	Block 1				Block 2				Block 3				Block 4				Avgpool	FC
				1.1		1.2		2.1		2.2		3.1		3.2		4.1		4.2			
				Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv	Conv		
Height	960	480	240	240	240	240	240	120	120	120	120	60	60	60	60	30	30	30	30	1	11
Width	512	256	128	128	128	128	128	64	64	64	64	32	32	32	32	16	16	16	16	1	
Kernels	1	64	64	64	64	64	64	128	128	128	128	256	256	256	256	512	512	512	512	512	

Table 22: ResNet18 sizes of outputs

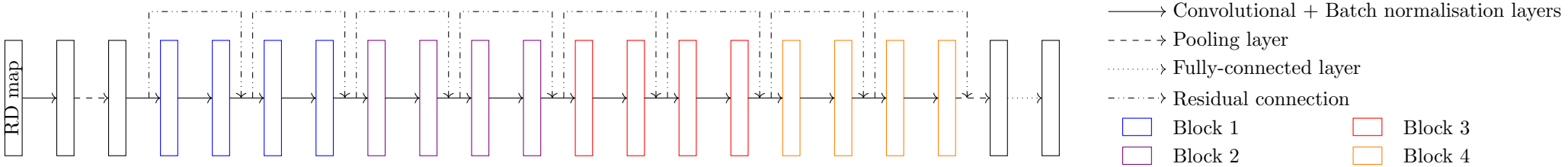


Figure 47: ResNet18 architecture diagram

7.4 Extra Figures - Results

This section provides figures that further illustrate the points made in the main text. Figure 48 shows the distribution of the initial raw train set (that has not yet been divided into the train / validation sets used in this thesis) and the test set in terms of the person who performs the gesture, where the gesture is performed and at which distance from the radar. Figure 49 provides a Pareto curve representation of the trade-off between the number of operations and the error/accuracy obtained for the FTs and classification networks, they contain the same data as that provided in Figures 22 and 34. Figure 50 shows all architectures on the Pareto curve between the number of operations/parameters in terms of the accuracy reached by the simplified architecture from Figure 29. All architectures reaching above 95.0% validation accuracy are those provided in the main text in Figure 30. Figure 51 exemplify the tuning that has been done in Section 5.3.4. Figure 52 provides a threshold tuning for the classification network for a larger range of thresholds compared to Figure 34.

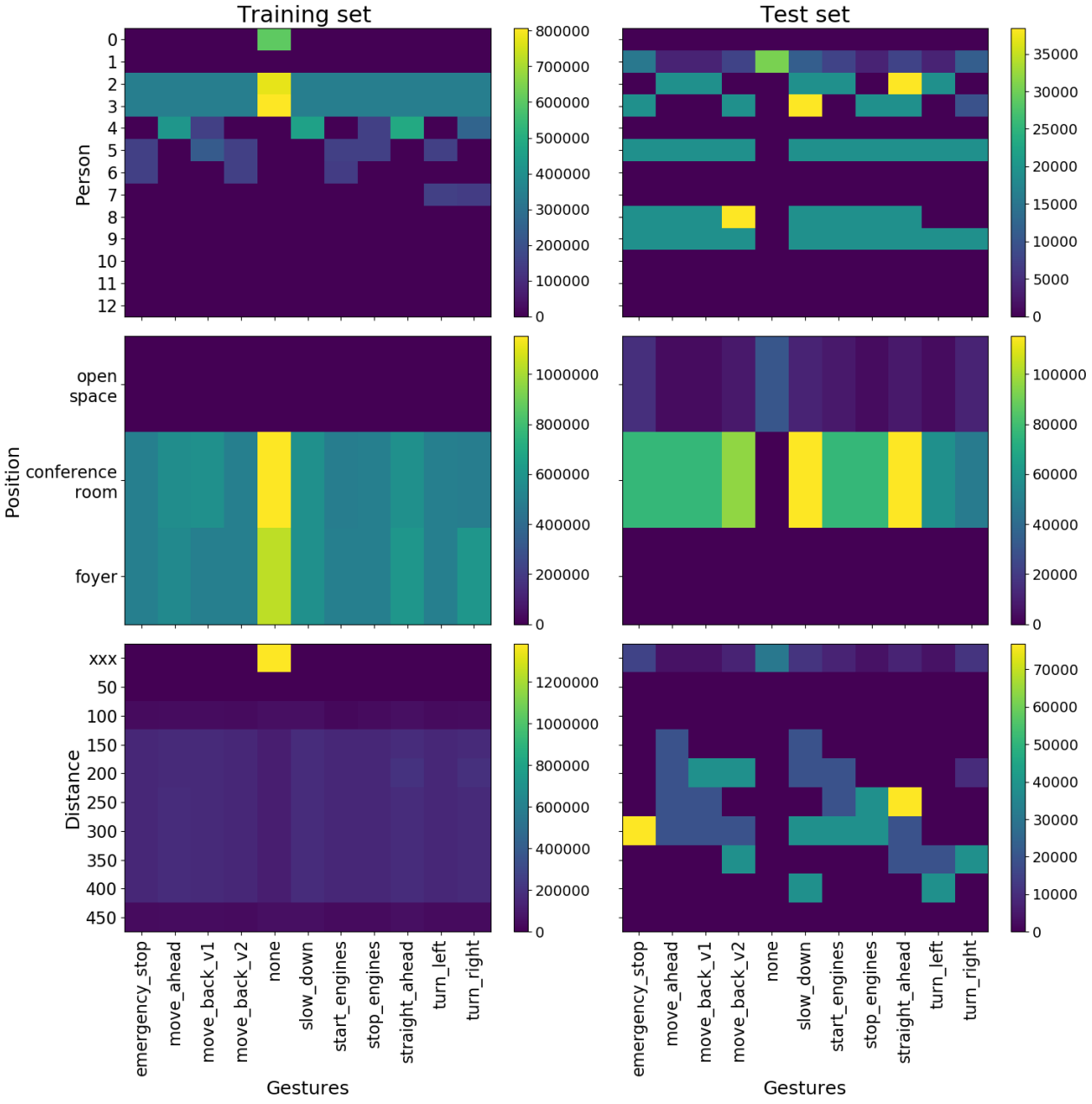
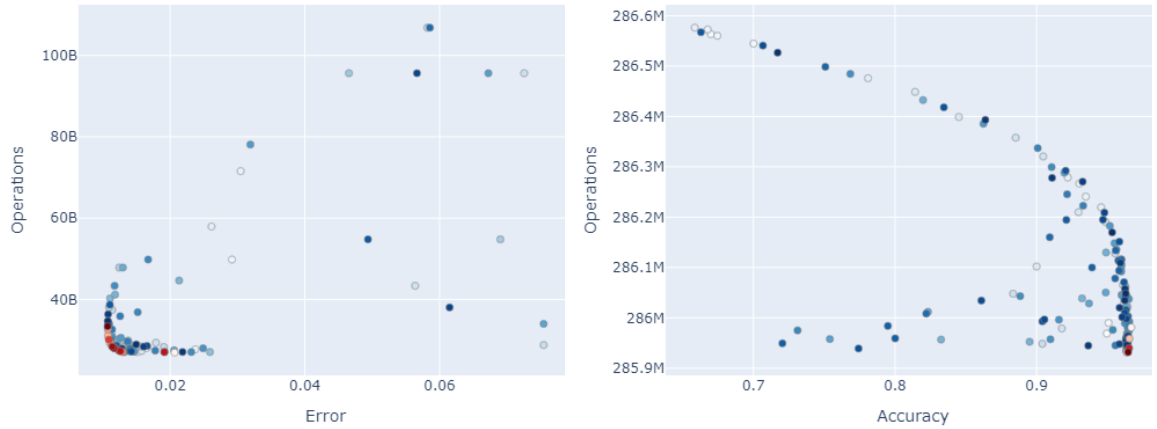


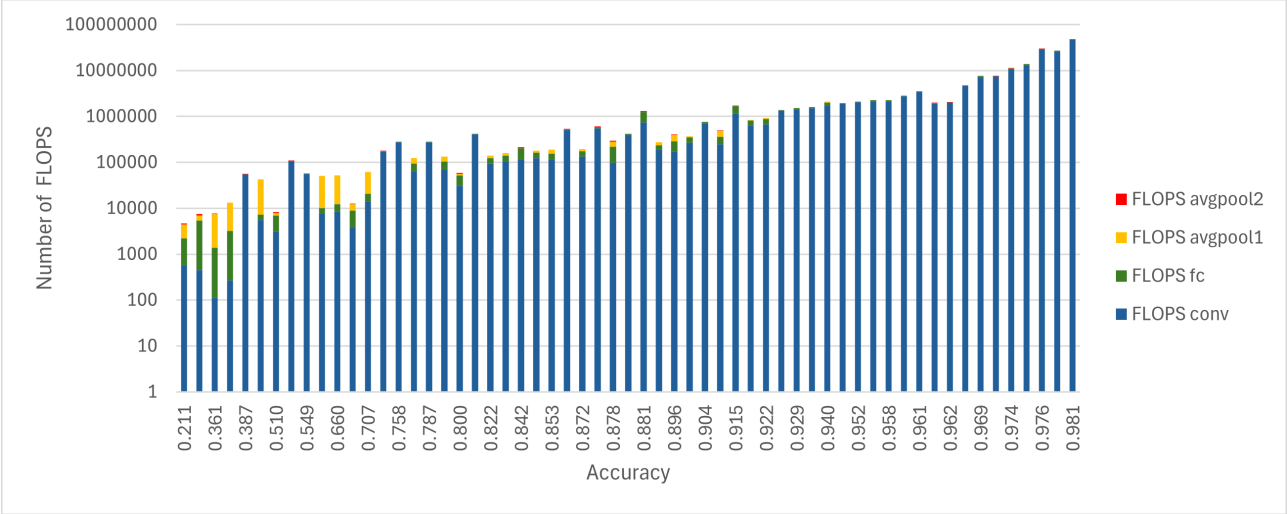
Figure 48: Distribution of the number of chirps per person, distance and location between the training set (the raw training set from [12] that contains both the train and validation sets used in this thesis) and the test set.



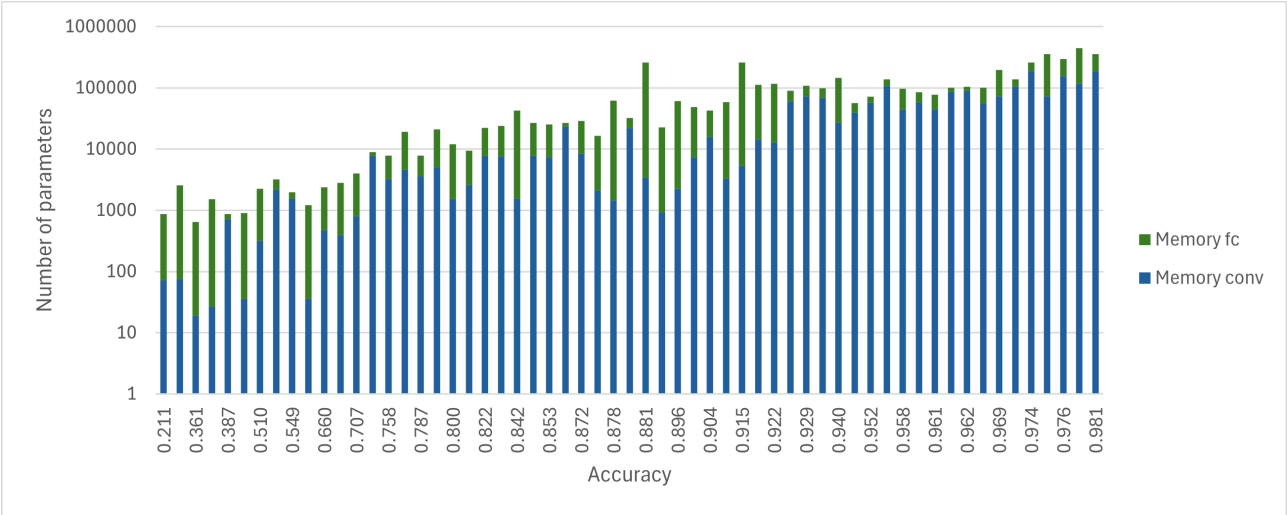
(a) Error vs number of operations for a spiking FT with 100 encoding steps using different thresholds, this is a Pareto front of the data in Figure 22

(b) Error vs number of operations for a spiking classification with 100 encoding steps using different thresholds, this is a Pareto front of the data in Figure 34

Figure 49: Operations vs error/accuracy curves for the spiking FT and classification networks

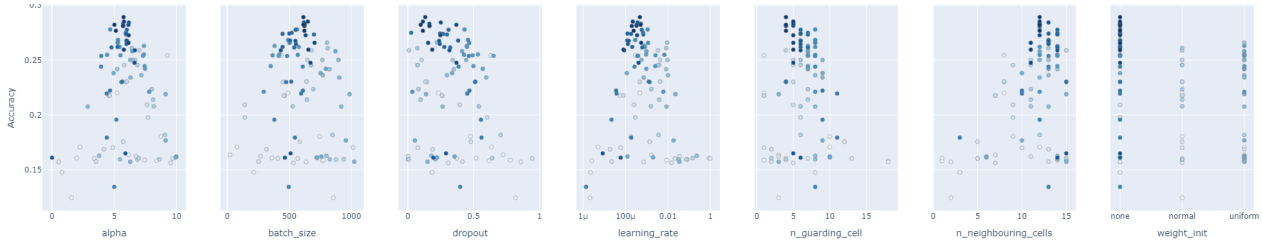


(a) Number of operations per layer for the architectures in Figure 30 in order of increasing accuracy

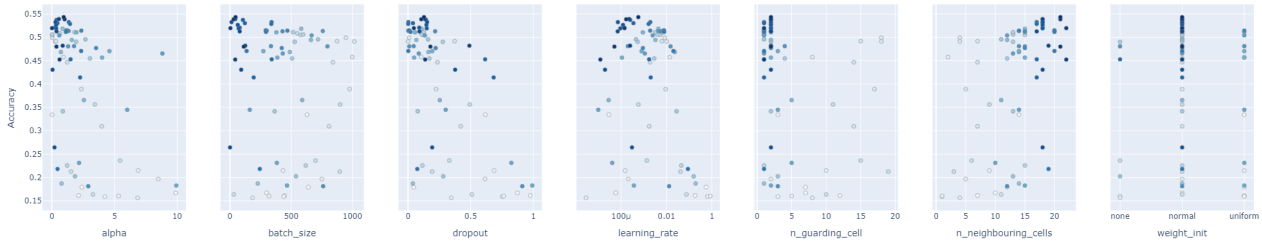


(b) Number of parameters per layer for the architectures in Figure 30 in order of increasing accuracy

Figure 50: Pareto front operations and memory comparisons

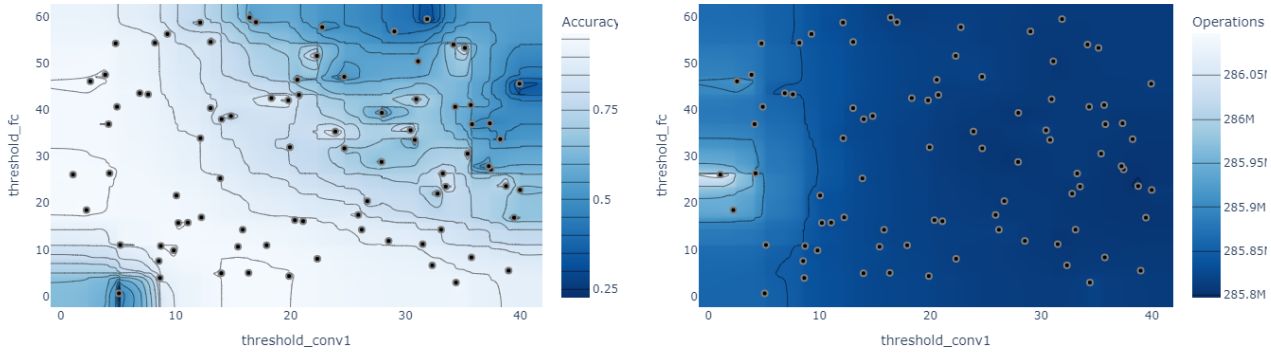


(a) Binary CA-CFAR tuning



(b) Masked CA-CFAR tuning

Figure 51: Validation accuracy for the tuning of the parameters of binary/masked CA-CFAR



(a) Validation accuracy

(b) Number of add operations per input data

Figure 52: Threshold tuning for the classification network using 100 encoding steps (threshold1 = $U_{th,conv}$ and threshold2 = $U_{th,fc}$)

Conv. layer		FC layer		Input	Conv. output	FC output
weights	biases	weights	biases			
12	32	9	16	7	6	16

Table 23: Final quantisation parameters for the spiking classification

Range layer		Doppler layer	Input	Range FT output	Doppler FT output
weights	biases	weights			
8	8	11	9	7	14

Table 24: Final quantisation parameters for the spiking FT

7.5 Extra Figures - Hardware

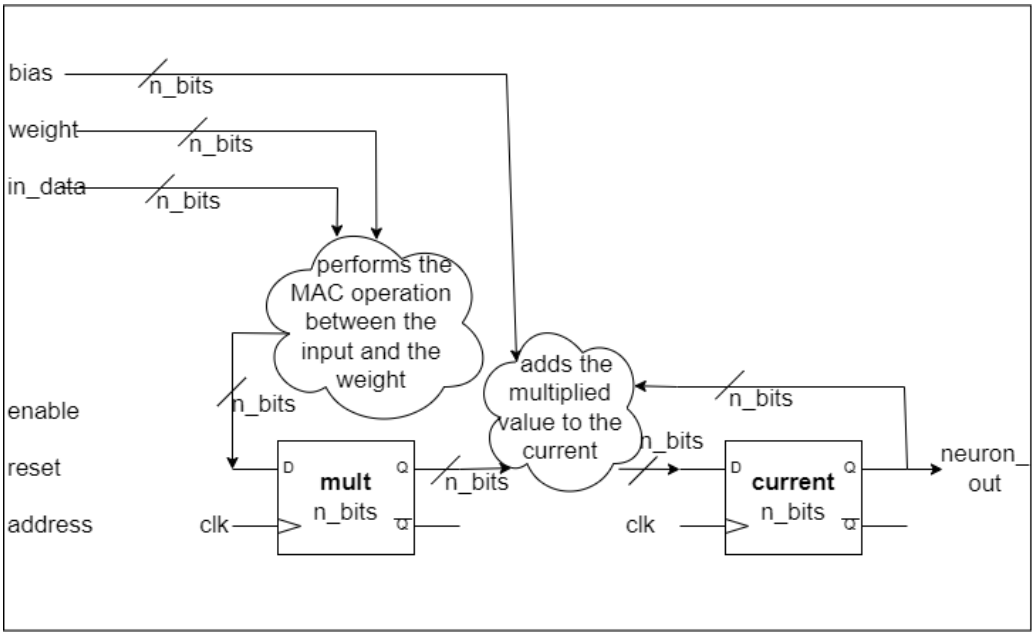
This section provides figures that further illustrate the points made in the main text. Figure 53 show how the memory is organised for a non-spiking and a spiking neuron. Figures 54, 55 and 56 provide the corresponding final state machines. Table 25 provides the power distribution between the leakage, internal and switching power for both spiking and non-spiking cases.

The constraints used in the hardware synthesis are the following:

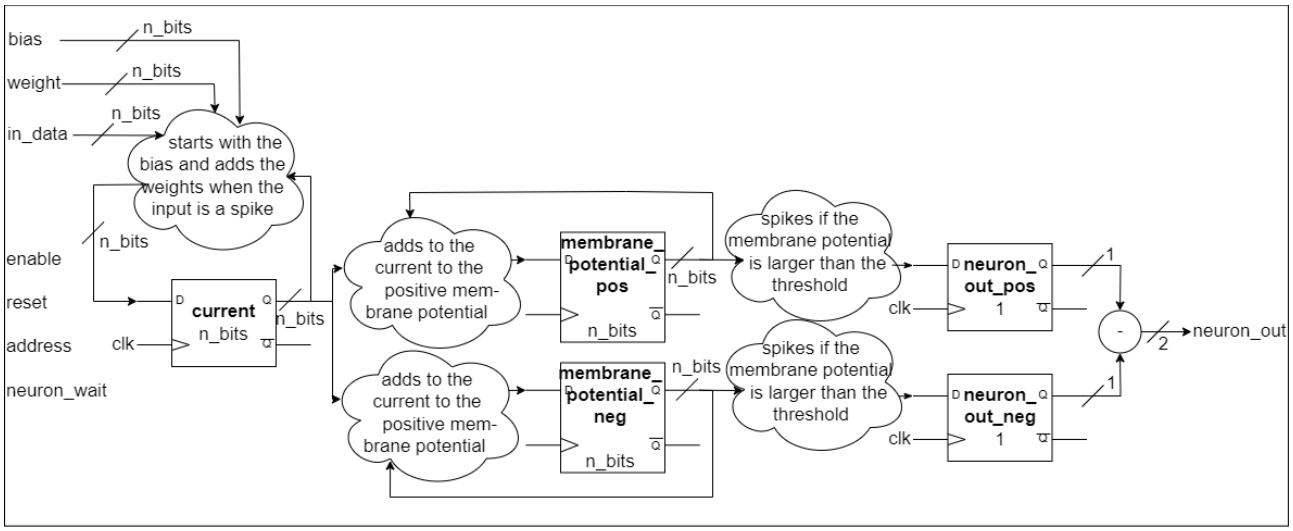
- clock period: $100ns$,
- clock uncertainty: $0.25ns$,
- maximum input/output delay: $5ns$,
- minimum input/output delay: $0ns$,
- all inputs that are not linked to the direct calculation of the RD map, so those used to write the weights/inputs and read the output, are set as false paths since their timings are not critical,
- clock gating is enabled since it provides additional energy gains,
- activity annotation is enabled since it allows the power reports to be more reliable, since the hardware is very small, it is possible to calculate activity annotation for the full calculation of the smaller RD map.

Category	Non-spiking			Spiking		
	Leakage (μW)	Internal (μW)	Switching (μW)	Leakage (μW)	Internal (μW)	Switching (μW)
Register	3.9	182.3	12.5	3.8	61.9	1.5
Logic	19.0	110.7	114.1	10.0	28.4	19.2
Clock	0.1	32.0	21.3	0.2	42.1	14.2
Sub-total	23.0	325.0	147.9	13.9	132.4	34.9
Total	495.9			181.2		

Table 25: Power consumption distribution for the non-spiking and spiking hardware

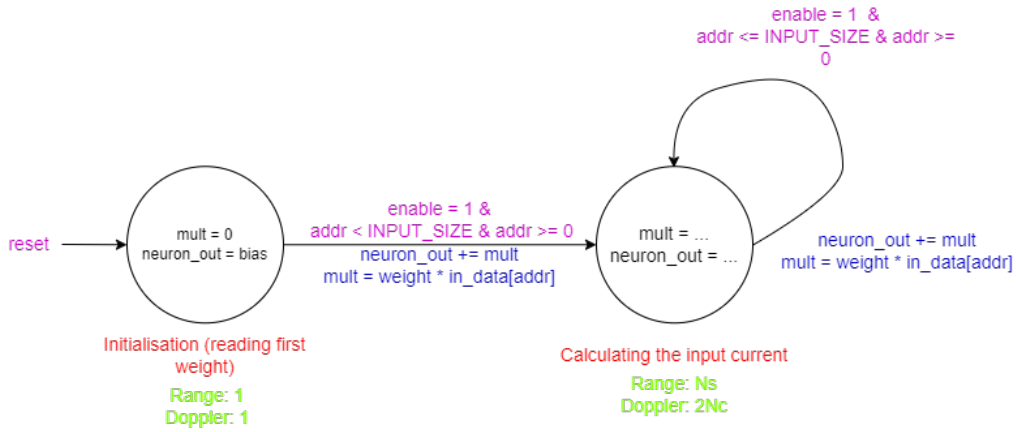


(a) Non-spiking neuron hardware diagram

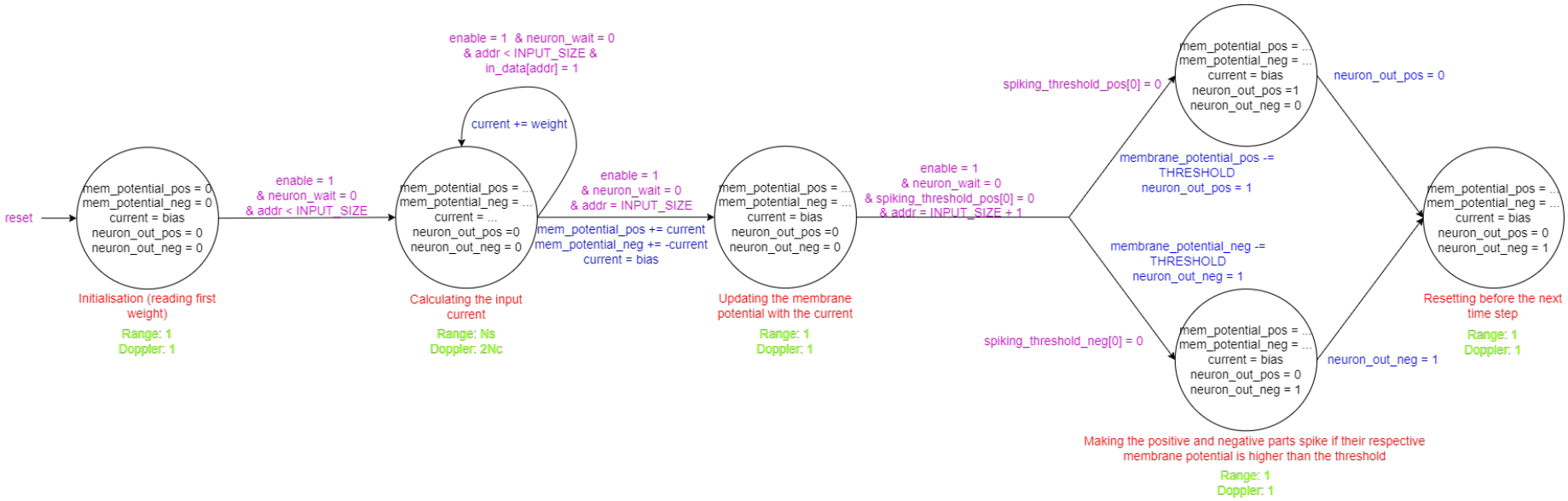


(b) Spiking neuron hardware diagram

Figure 53: Hardware diagrams for the neurons: the inputs that are not linked to anything are used in combinational logic, those links are removed for the diagrams to remain readable, the numbers of bits needed per register are provided in the register schematics (`n_bits` represents the number of bits kept after quantisation)



(a) Non-spiking neuron FSM



(b) Spiking neuron FSM

Figure 54: FSMs for the spiking and non-spiking neurons, the colour code is as follows: red → utility of each FSM state, green → number of clock cycles at each step, purple → conditions to move to the next state, blue → states what happens when the condition is satisfied (those are only highlighted when it cannot be deducted from the following state)

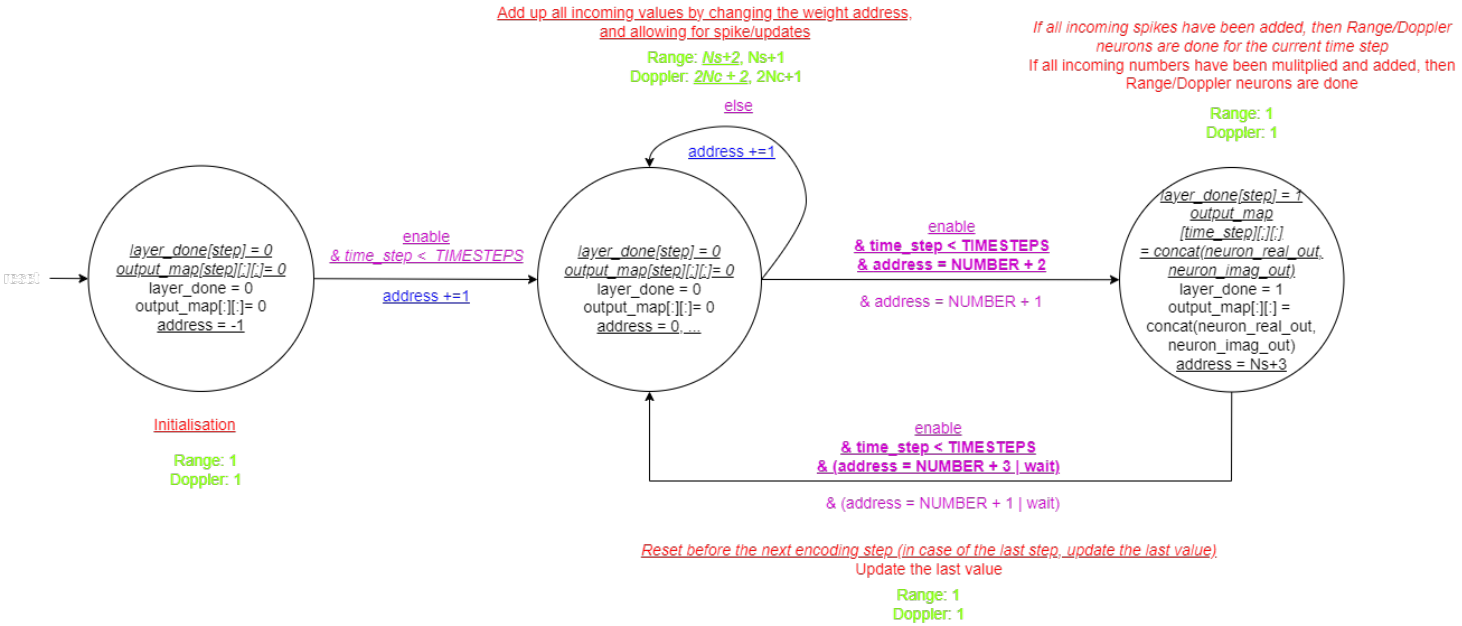
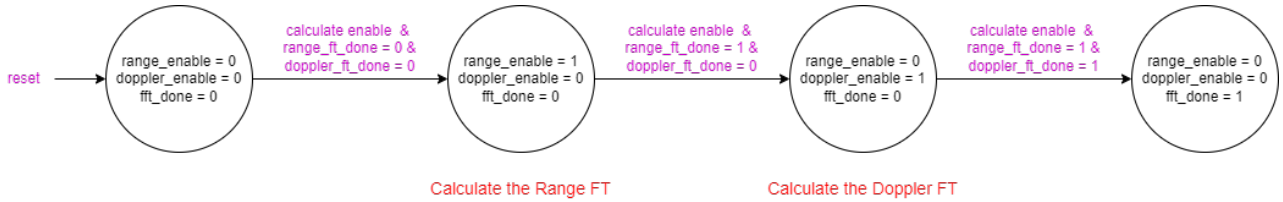
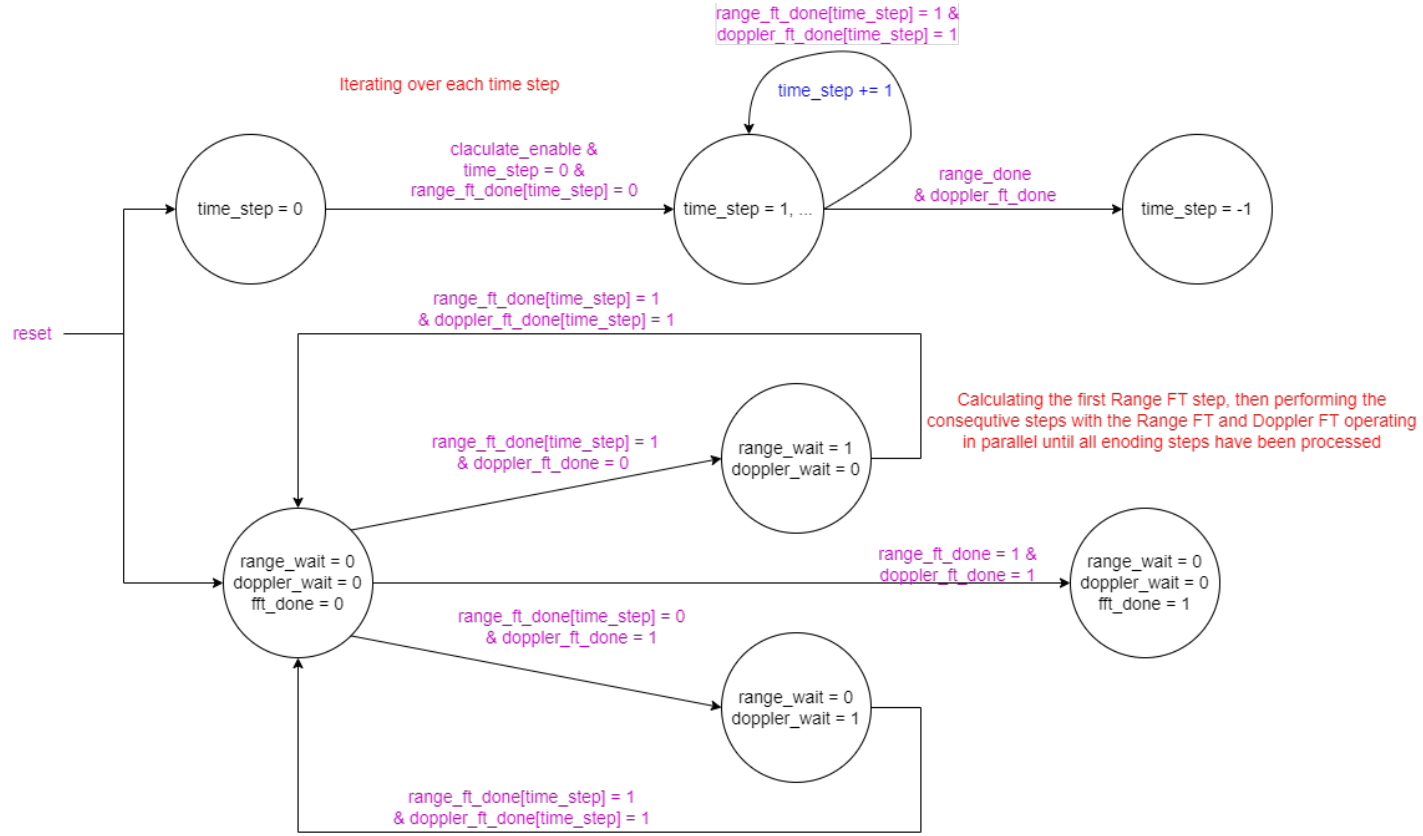


Figure 55: FSM for the Range (then $NUMBER = N_s$) and Doppler FT (then $NUMBER = 2 \cdot N_c$), the colour code is as follows: red $\rightarrow$ utility of each FSM state, green $\rightarrow$ number of clock cycles at each step, purple $\rightarrow$ conditions to move to the next state, blue $\rightarrow$ states what happens when the condition is satisfied (those are only highlighted when it cannot be deducted from the following state), the underlined text applied to both spiking and non-spiking, the underline italic text applied only to the spiking case and the non-underlined non-italic text applied only to the non-spiking case



(a) Non-spiking FT FSM



(b) Spiking FT FSM

Figure 56: FSM for the full algorithm (Range and Doppler FTs), the colour code is as follows: red → utility of each FSM state, purple → conditions to move to the next state, blue → states what happens when the condition is satisfied (those are only highlighted when it cannot be deducted from the following state)

References

[1] Muhammad Arsalan et al. "Radar-Based Gesture Recognition System using Spiking Neural Network". In: *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2021, pp. 1–5. DOI: 10.1109/ETFA45728.2021.9613183.

[2] Dighanchal Banerjee et al. "Application of Spiking Neural Networks for Action Recognition from Radar Data". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pp. 1–10. DOI: 10.1109/IJCNN48605.2020.9206853.

[3] Bernhard Vogginger et al. "Automotive Radar Processing With Spiking Neural Networks: Concepts and Challenges". In: *Frontiers in Neuroscience* 16 (Apr. 2022). ISSN: 1662453X. DOI: 10.3389/fnins.2022.851774.

[4] Ali Safa et al. "Improving the Accuracy of Spiking Neural Networks for Radar Gesture Recognition Through Preprocessing". In: *IEEE Transactions on Neural Networks and Learning Systems* 34 (6 June 2023), pp. 2869–2881. ISSN: 21622388. DOI: 10.1109/TNNLS.2021.3109958.

[5] I J Tsang et al. "Radar-Based Hand Gesture Recognition Using Spiking Neural Networks". In: 10 (2021), p. 1405. DOI: 10.3390/electronics. URL: https://doi.org/10.3390/electronics.

[6] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. 2023. DOI: 10.1109/JPROC.2023.3308088.

[7] Amar Shrestha et al. "A Survey on Neuromorphic Computing: Models and Hardware". In: *IEEE Circuits and Systems Magazine* 22.2 (2022), pp. 6–35. DOI: 10.1109/MCAS.2022.3166331.

[8] Vasiliki Giagka. *Bioelectricity course (ET4130)*. TUDelft. 2024.

[9] Seijoon Kim et al. "Spiking-yolo: spiking neural network for energy-efficient object detection". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 07. 2020, pp. 11270–11277.

[10] Fei Tang and Wanling Gao. "SNNBench: End-to-end AI-oriented spiking neural network benchmarking". In: *BenchCouncil Transactions on Benchmarks, Standards and Evaluations* 3.1 (2023), p. 100108. ISSN: 2772-4859. DOI: https://doi.org/10.1016/j.tbench.2023.100108. URL: https://www.sciencedirect.com/science/article/pii/S277248592300025X.

[11] Bing Han, Abhronil Sengupta, and Kaushik Roy. "On the energy benefits of spiking deep neural networks: A case study". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. 2016, pp. 971–976. DOI: 10.1109/IJCNN.2016.7727303.

[12] Leon Muller et al. "Aircraft Marshaling Signals Dataset of FMCW Radar and Event-Based Camera for Sensor Fusion". In: *Proceedings of the IEEE Radar Conference* 2023-May (2023). ISSN: 23755318. DOI: 10.1109/RadarConf2351548.2023.10149465.

[13] Aston Zhang et al. *Dive into Deep Learning*. 2023. arXiv: 2106.11342 [cs.LG]. URL: https://arxiv.org/abs/2106.11342.

[14] Qiaoyi Su et al. *Deep Directly-Trained Spiking Neural Networks for Object Detection*. 2023. arXiv: 2307.11411 [cs.CV].

[15] Ana Stanojevic et al. "Time-encoded multiplication-free spiking neural networks: application to data classification tasks". In: *Neural Computing and Applications* 35.9 (2023), pp. 7017–7033.

[16] Sandro Widmer et al. "Design of Time-Encoded Spiking Neural Networks in 7-nm CMOS Technology". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 70.9 (2023), pp. 3639–3643.

[17] Markus Nagel et al. "A white paper on neural network quantization". In: *arXiv preprint arXiv:2106.08295* (2021).

[18] Muhammad Arsalan, Avik Santra, and Vadim Issakov. "RadarSNN: A Resource Efficient Gesture Sensing System Based on mm-Wave Radar". In: *IEEE Transactions on Microwave Theory and Techniques* 70 (4 Apr. 2022), pp. 2451–2461. ISSN: 15579670. DOI: 10.1109/TMTT.2022.3148403.

[19] Javier López-Randulfe et al. "Spiking Neural Network for Fourier Transform and Object Detection for Automotive Radar". In: *Frontiers in Neurorobotics* 15 (June 2021). ISSN: 16625218. DOI: 10.3389/fnbot.2021.688344.

[20] Javier López-Randulfe et al. "Time-coded spiking fourier transform in neuromorphic hardware". In: *IEEE Transactions on Computers* 71.11 (2022), pp. 2792–2802.

[21] Zheng Dong et al. *Location-aware Single Image Reflection Removal*. 2021. arXiv: 2012.07131 [cs.CV]. URL: https://arxiv.org/abs/2012.07131.

[22] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV]. URL: https://arxiv.org/abs/1405.0312.

[23] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[24] Seongbin Oh et al. "Hardware implementation of spiking neural networks using time-to-first-spike encoding". In: *arXiv preprint arXiv:2006.05033* (2020).

[25] Jianhao Ding et al. *Optimal ANN-SNN Conversion for Fast and Accurate Inference in Deep Spiking Neural Networks*. 2021. arXiv: 2105.11654 [cs.NE]. URL: https://arxiv.org/abs/2105.11654.

[26] Jim Johan Vermunt and Federico Corradi. *FMCW Radar Signal Processing Pipeline for Human Gesture Classification*. 2023.

[27] Alessandro Pappalardo. *Xilinx/brevitas*. DOI: 10.5281/zenodo.3333552. URL: https://doi.org/10.5281/zenodo.3333552.

[28] Javier López-Randulfe et al. "Time-Coded Spiking Fourier Transform in Neuromorphic Hardware". In: *IEEE Transactions on Computers* 71.11 (2022), pp. 2792–2802. DOI: 10.1109/TC.2022.3162708.

[29] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.