NTNU
Norwegian University of
Science and Technology

Ultra-Wideband Radar Simulator for classifying Humans and Animals based on Micro-Doppler Signatures

Helge Langen

Master of Science in Electronics
Submission date: June 2016
Supervisor: Lars Magne Lundheim, IET
Co-supervisor: Jan Roar Pleym, Novelda AS

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Abstract

A system has been developed to allow computers to autonomously classify radar targets based on their micro-doppler signatures. The solution consists of an algorithm for generating a targets micro-doppler signature through frequency analysis of the radar signal, a multi-layer artificial neural network for classifying the target based on the information in the processed signal, and a generator that creates synthetic data for training the artificial neural network.

The neural network is trained using deep learning techniques. Point scatterer models of walking humans, dogs and domestic cats and a mathematical model of an ultra-wideband pulse-doppler radar was used when generating the synthetic training data. Random variations were applied to the model parameters to mimic the real-world diversity experienced between the different specimen of the same species.

In a set of synthetically generated evaluation data, the system was able to correctly classify 444 out of 500 targets (accuracy of 89%). The best results were obtained when the network was fed with a combination of micro-doppler signature data and selected portions of the raw baseband radar frames. It was also shown that the accuracy increased when the synthetic training data was generated with a higher simulated pulse repetition frequency.

Some real recordings were made using a Novelda radar module featuring their X2 radar system-on-chip (SoC). The system was able to correctly classify the targets in the recordings with a very high confidence. While the number of real recordings used to evaluate the system is too low to conclusively qualify the solution as successful in real life conditions, it indicates that it might be possible to avoid costly data gathering by generating the training data synthetically, which is an interesting find in itself.

Sammendrag

En løsning for automatisk gjenkjennelse av radarmål basert på mikro-doppler-signaturer har blitt utvilket. Løsningen består av en algoritme for frekvensanalyse av radarsignalet, en maskinlæringsentitet basert på kunstige nevrale nett, og en generator som lager syntetiske treningsdata for maskinlæringsentiteten.

Det nevrale nettet blir trent ved hjelp av dyp læring. Modeller av gående mennesker, hunder og huskatter basert på punktspredere, samt en matematisk modell av en ultra-bredbåndig pulse-doppler-radar ble brukt ved generering av syntetiske treningsdata. Tilfeldige variasjoner ble lagt på modellparametrene for å gjenskape eksemplarvariasjonene man finner i den virkelige verden.

Av et sett med syntetisk genererte testdata var systemet i stand til å korrekt gjenkjenne 444 av 500 radarmål (89% treffsikkerhet). Den største treffsikkerheten ble oppnådd ved å mate nettverket med en kombinasjon av mikro-doppler-signaturen og utvalgte deler av rådataene fra radaren. Større treffsikkerhet ble også oppnåd ved å bruke en høyere simulert pulsrepetisjonsrate.

Noen opptak av virkelige mål ble gjort ved hjelp av en radarmodul fra Novelda AS, utstyrt med en Novelda X2 radarbrikke. Systemet klarte å gjenkjenne målene fra disse opptakene med høy nøyaktighet. Selv om antallet opptak av virkelige mål er for lavt til å kvantifisere ytelsen til systemet i virkelige situasjoner, viser det at det er mulig å trene opp nettverket til å gjenkjenne virkelige mål ved bruk av syntetiske treningsdata, hvilket er et interessant resultat i seg selv. Bruk av syntetiske treningsdata reduserer behovet for å gjennomføre kostbare innsamlinger av virkelige treningsdata.

Preface

Machine vision is a term most oftenly associated with computers being able to identify objects in a two-dimensional plot of light intensity values, and sometimes also with color information, generated by a CMOS imaging sensor. The signal from an imaging sensor benefit from a very good lateral spatial resolution, allowing the computer to identify features down to a micrometer level, depending on the type of optics used to focus the image on the image sensor plane.

What imaging sensors aren't especially good at, is resolving objects in range. Sure, solutions exist using an angled laser beam that estimates range based on the lateral displacement of the laser dot in the image, but this requires the laser to be aimed precisely at the object for which the range is to be measured. So what if computers could "see" in range, with a wide viewing angle?

With ultra-wideband (UWB) pulse-doppler radar modules, computers can resolve solid objects in range. The ultra-wideband radar pulse and comparatively high carrier frequency facilitates a range resolution down to a sub-centimeter level. UWB radars gives computers depth vision - where the width of the field of view is determined by the antenna used.

Novelda AS is a company specializing in low-power, low-cost system-on-chip (SoC) solutions for ultra-wideband pulse-doppler radars. They proposed in 2014 a project for demonstrating the capabilities of their products with respect to analyzing the motion pattern of a moving radar target. The ultimate goal was to analyze time-varying patterns in the doppler frequency shift caused by a target moving radially with respect to the radar, and use this pattern as a basis for identifying the type of target. Time-harmonic variations in the doppler spectrum of a radar signal is known as the micro-doppler effect, and the pattern of these variations associated with a type of target is known as the target's micro-doppler signature.

We will in this project continue exploring these capabilities, attempting to use the findings from previous work to build a system using micro-doppler processed data as basis for classifying radar targets.

Acknowledgments

Acknowledgments is given to Novelda AS and in particular Jan Roar Pleym for providing this project opportunity, for providing the tools necessary for solving this task, and for sharing their extensive knowledge within the field of ultra-wideband pulse-doppler radar technology.

I further want to thank professor Torbjørn Svendsen and associate professor Magne Hallstein Johnsen, both from the NTNU Department of Electronics and Telecommunications, for pointing this project in the right direction and sharing their knowledge in the field of machine learning.

Finally, big thanks are given to professor Lars Lundheim at the NTNU Department of Electronics and Telecommunications for taking the task of supervising this project, and for providing frequent and valuable guidance, feedback and fruitful discussions, and for motivating me to keeping a high standard in my work.

Helge Langen

Helge Langen
Trondheim, June 10th, 2016

Contents

Abstract **I**

Sammendrag **II**

Preface **III**

Acknowledgments **IV**

Table of Contents **V**

List of Tables **VIII**

List of Figures **IX**

1. Introduction: Radar Target Classification using Micro-Doppler Signatures **1**
1.1. Problem Description 1
1.2. Gait Motion, Gait Frequency and Gait Cycle 1
1.3. The Micro-Doppler Effect in Radar 2

2. Radar Signal Capture and Feature Extraction **4**
2.1. Radar Module Signal Processing 4
2.1.1. Radar Module Overview 4
2.1.2. Generating the Pulse 7
2.1.3. Propagation Loss and Delay 9
2.1.4. Receiver Noise 10
2.1.5. Digitizing the Signal 11
2.1.6. Digital Down-Conversion 12
2.1.7. Matched Filtering 12
2.1.8. Calculating Doppler Frequency Shift 12
2.2. Feature Extraction 14
2.2.1. Feature Extraction Overview 14
2.2.2. Frame Buffer, Clutter Removal and Windowing 18
2.2.3. Range-Frequency Analysis 19
2.2.4. Time-Frequency Analysis 21
2.2.5. Gait-Frequency Analysis 22
2.2.6. Feature Extraction Summary 24

3. Pattern Recognition and Classification with Neural Networks **25**
3.1. Artificial Neural Networks 25
3.1.1. The Neuron 25
3.1.2. Neural Network Structure 26

3.2. Neural Network Evaluation . . . 28
3.2.1. Error Functions . . . 28
3.2.2. Confusion Plots . . . 29
3.3. Neural Network Training . . . 30
3.3.1. Weight Update with Error Gradients and Backpropagation . 31
3.3.2. Bias Neurons . . . 34
3.3.3. Deep Architectures and Deep Learning . . . 35
3.3.4. Training with Autoencoders . . . 35
3.4. Designing a Neural Network for Radar Target Classification . . . 38
3.4.1. Target Classification Process Parameters . . . 38
3.4.2. Preparing the Input Data . . . 38
3.4.3. Type 1 Feature Vector . . . 40
3.4.4. Type 2 Feature Vector . . . 41
3.4.5. Network Layer Structure . . . 44
3.4.6. Activation Functions . . . 45

4. Testing and Results 49
4.1. Generating synthetic Training Data . . . 49
4.1.1. Micro-Doppler Simulator . . . 49
4.1.2. Generating a Dataset . . . 49
4.2. Real Input Data . . . 52
4.3. Testing the Neural Network . . . 54
4.3.1. Neural Network Training . . . 54
4.3.2. Performance Measurements . . . 57
4.3.3. Comments to Performance Measurements . . . 59

5. Discussion 61
5.1. Comments to the Testing Procedure . . . 61
5.1.1. Effect of the Network Weight Initialization . . . 61
5.1.2. Advantages and Disadvantages of using synthetic Training Data . . . 62
5.1.3. PRF Limitations and Aliasing . . . 63
5.1.4. No Clutter in synthetic Data . . . 64
5.2. Further Work . . . 64
5.2.1. Gathering real Training Data . . . 64
5.2.2. Implementation and live Processing . . . 64
5.2.3. Create Decision Limits for separating Classes . . . 65
5.2.4. Optimizing Feature Extraction for Machine Learning . . . 66
5.2.5. Try different Network and Input Data Configurations . . . 67

6. Conclusion 68

Bibliography 69

Appendix A. Target Model Parameters, Position and Range calculation 70

Appendix B. Synthetic Training Data Parameters **72**

Appendix C. Confusion Plots for Network Evaluation **75**
C.1. Confusion Plots for ReLU Network 75
C.2. Confusion Plots for Sigmoid Network 75

List of Tables

2.1. Parameters used in Radar Module 8
2.2. Variables used in Radar Module 8
2.3. Signal Processing Steps in Radar Module 9
2.4. Steps in the Feature Extraction Process 16
2.5. Parameters used in the Feature Extraction Process 16
2.6. Variables used in the Feature Extraction Process 17

3.1. Explanation of Confusion Plot Cells 30
3.2. Parameters and Variables used in the Target Classification Process . 39

4.1. Synthetic Training Data Generation Algorithm 50
4.2. Syntetic Training Datasets 51
4.3. Neural Network Structure and Training Parameters 55
4.4. Description of Variables in Training Algorithm Pseudocode 55
4.5. Target Classification Network Training Algorithm 56
4.6. Performance Metrics - ReLU Network 58
4.7. Performance Metrics - Sigmoid Network 58

B.1. Human Body Model Scatterer Parameters used in synthetic Data Generation 72
B.2. Dog Body Model Scatterer Parameters used in synthetic Data Generation 73
B.3. Cat Body Model Scatterer Parameters used in synthetic Data Generation 73
B.4. Standard Deviations for the random Variations applied to each Body Model Parameter in synthetic Data Generation 73
B.5. Radar Parameters used in synthetic Data Generation 74
B.6. Feature Set, Feature Extraction and Classification Parameters used in synthetic Data Generation 74

List of Figures

2.1. Radar Module Block Diagram and Parameters 6
2.2. Radar Module Signal Processing Steps and Parameters 7
2.3. Feature Extraction Process Block Diagram and Parameters 15
2.4. Slow and fast Time Scales 15
2.5. Example of Range-Doppler Power Spectrum 20
2.6. Example of Time-Frequency Power Spectrum 22
2.7. Example of Gait-Frequency Power Spectrum 23

3.1. Generic Model of a Neuron 26
3.2. Generic Model of a Feedforward Network with two Hidden Layers . . 27
3.3. Example of Confusion Plot with 500 Input Feature Sets 29
3.4. Parameters related to calculating Error Gradient and Weight Deltas 34
3.5. Generic feedforward Network with Bias Neurons 34
3.6. Example of Autoencoder Network 36
3.7. Example of Type 1 Feature Vector 41
3.8. Example of Type 2 Feature Vector, Target Class 1 43
3.9. Example of Type 2 Feature Vector, Target Class 2 44
3.10. Overview of full Neural Network 45
3.11. Logistic Sigmoid Activation Function 46
3.12. ReLU Activation Function 47
3.13. Softmax Layer Structure 48

4.1. Human Body Model 52
4.2. Dog Body Model 52
4.3. Cat Body Model 52
4.4. Type 2 Feature Vector from Radar Recording of Cat 53
4.5. Type 2 Feature Vector from Radar Recording of Human on Treadmill 53

5.1. Development of Confidence with good initial Weights 62
5.2. Development of Confidence with poor initial Weights 62

A.1. Coordinate System 71

C.1. Confusion Plot, ReLU, T1D60 76
C.2. Confusion Plot, ReLU, T1D200 76
C.3. Confusion Plot, ReLU, T2D60 76
C.4. Confusion Plot, ReLU, T2D200 76
C.5. Confusion Plot, ReLU, T1C60 77
C.6. Confusion Plot, ReLU, T1C200 77
C.7. Confusion Plot, ReLU, T2C60 77
C.8. Confusion Plot, ReLU, T2C200 77
C.9. Confusion Plot, Sigmoid, T1D60 78

C.10.Confusion Plot, Sigmoid, T1D200 78
C.11.Confusion Plot, Sigmoid, T2D60 78
C.12.Confusion Plot, Sigmoid, T2D200 78
C.13.Confusion Plot, Sigmoid, T1C60 79
C.14.Confusion Plot, Sigmoid, T1C200 79
C.15.Confusion Plot, Sigmoid, T2C60 79
C.16.Confusion Plot, Sigmoid, T2C200 79

1. Introduction: Radar Target Classification using Micro-Doppler Signatures

1.1. Problem Description

Modern ultra-wideband radar offerings are capable of detecting moving targets with high precision at a few meters range. In alarm and surveillance applications it is desirable to be able to discern humans and animals from each other. Previous work [2] [5] has been conducted focusing on extracting the micro-doppler features present in the recorded radar signal as a basis for target classification (see section 1.3 for an explanation of micro-doppler features, micro-doppler signatures and the micro-doppler effect in radar).

The focus of this project will be to see if it is possible for a computer to determine if a moving target belongs to one of two classes, one class representing humans, the other representing pet animals. The prime hypothesis will be that the information contained in the target's micro-doppler signature, as extracted from the recorded radar signal, is suitable as input data for such an automated target classifier. Since it was identified in [5] that some information is lost in the micro-doppler extraction process, alternative ways of processing the radar signal prior to being input to the classifier will also be explored. The solution will primarily be evaluated using synthetic data generated using the simulator developed in [5], supported by some limited tests with real data when possible.

1.2. Gait Motion, Gait Frequency and Gait Cycle

The term "gait" will be used frequently throughout this report. "Gait" simply means 'the pattern of movement of the limbs of animals, including humans, during locomotion over a solid substrate' (Wikipedia), and we define the frequency of which the motion pattern repeats itself as the gait frequency, and one cycle of the motion pattern as a gait cycle.

Some typical properties of the gait pattern of a human is e.g. that each arm

and each leg will swing back and forth once during each cycle, the arms will swing in opposite phase to each other, each leg will swing in phase with the opposite arm, and the torso and head will have a minor oscillation at twice the gait frequency. A dog's gait pattern has similar properties, where each leg will swing back and forth once during each cycle, legs on the left side in opposite phase with the right ones, the rear legs in phase with the opposite front leg, and the head/torso with a small oscillation twice per cycle.

1.3. The Micro-Doppler Effect in Radar

Taking advantage of the doppler effect to detect moving target in the presence of stationary clutter has been a well-known technique for decades. With the advent of digital signal processing techniques (and computers with the processing power to support it), continuous analysis of the frequency spectrum of the radar return signal allowed the detection of time-varying patterns in the signal's doppler spectrum by performing joint time-frequency analysis. Moving targets with rigid as well as articulated bodies frequently have oscillating micro-motion components in addition to the bulk motion vector, which again will cause larger or smaller variations to the doppler shift of the return signal already caused by the bulk motion. These time-harmonic doppler frequency variations are known as the micro-doppler effect, and the pattern of the doppler frequency variations produced by a specific type of target over at least one gait cycle is known as the target's micro-doppler signature [1].

As mentioned in the preface, it is suggested that these micro-doppler signature patterns provide information that allows us to identify a radar target. This is possible because distinct features in the micro-doppler signature relates to physical features of the target and its motion pattern. Such features includes the number of distinguishable sinusoids, the amplitude and frequency of each sinusoid, the phase difference between the sinusoids and the power in each sinusoid. One example given in [1] is the use of micro-doppler processing in air surveillance radars to classify helicopters, where it is possible to determine the number of rotor blades (from the number of sinusoids), the angle between each rotor blade (from the phase difference between the sinusoids), the rotational speed of the rotor (from the sinusoid frequency), the tangential speed of the tips of the rotor blades (from the amplitude of the sinusoids) and the length of the rotor blades (by combining the knowledge of tip tangential speed and rotational speed). In combination with knowledge about the target's radar cross section (RCS) and air speed (which we

do not need micro-doppler processing to calculate), we have a list of features that limits the number of possible helicopter types matching these criterions to a very small one.

2. Radar Signal Capture and Feature Extraction

2.1. Radar Module Signal Processing

Most of the derivations in this section have been previously published in [2] and [5]. They are repeated here for reference and to provide consistency when developing the mathematical foundation for the target classification process.

2.1.1. Radar Module Overview

The radar module is shown as a block diagram in figure 2.1, where each block represents a submodule performing one operation. Figure 2.2 represents the same module as a signal processing flowchart showing the relationship between the parameters and process steps and how they affect the received signal. A thorough explanation of how the output signal $r_{i,bb}[n]$ appears as a function of time, the input parameters to the radar and the size, position and motion of the target will be given throughout the rest of this section, but a brief explanation of the blocks in the figures is as follows:

- The frame timer signalizes the start of a new frame every $1/f_p$ seconds
- The discrete sine wave $c[n]$ is mixed with a pulse shaping filter with impulse response $p[n]$ each time the frame timer signalizes start of a new frame to form the transmitted pulse waveform
- The pulse waveform is converted to an analog signal and amplified by the power amplifier (PA) to the transmitter power level P_t before being radiated by the Tx antenna
- The pulse is reflected by the target and received by the Rx antenna with a propagation loss $L(t)$ and propagation delay τ proportional to the target's range $R(t)$ and radar cross section (RCS) σ
- The received pulse is amplified by the receiver low-noise amplifier (LNA)
- The analog to digital converter (ADC) digitizes the signal at a sampling frequency f_s starting at the frame offset time t_s after the new frame signal is received, and continues until N samples have been digitized.
- The digital down-converter creates a complex phasor by summing the carrier

with an imaginary component with the same amplitude 90° out of phase, and mixes it with the received signal $r_i[n]$

- The down-converted signal $r_{i,bbu}[n]$ is mixed with a matched filter with impulse response $p[-n]$, which maximizes the signal to noise ratio, removes the unwanted high frequency parts of the signal, and outputs the filtered baseband signal $r_{i,bb}[n]$

Note that the radar module can be set to output either $r_{i,bb}[n]$ or to output $r_i[n]$ directly, in which the latter two steps are skipped.

Tables 2.1 and 2.2 lists the constant and variable signal processing parameters shown in figures 2.1 and 2.2, while table 2.3 lists the signals and signal processing steps and on which page their relevant formulas are given.

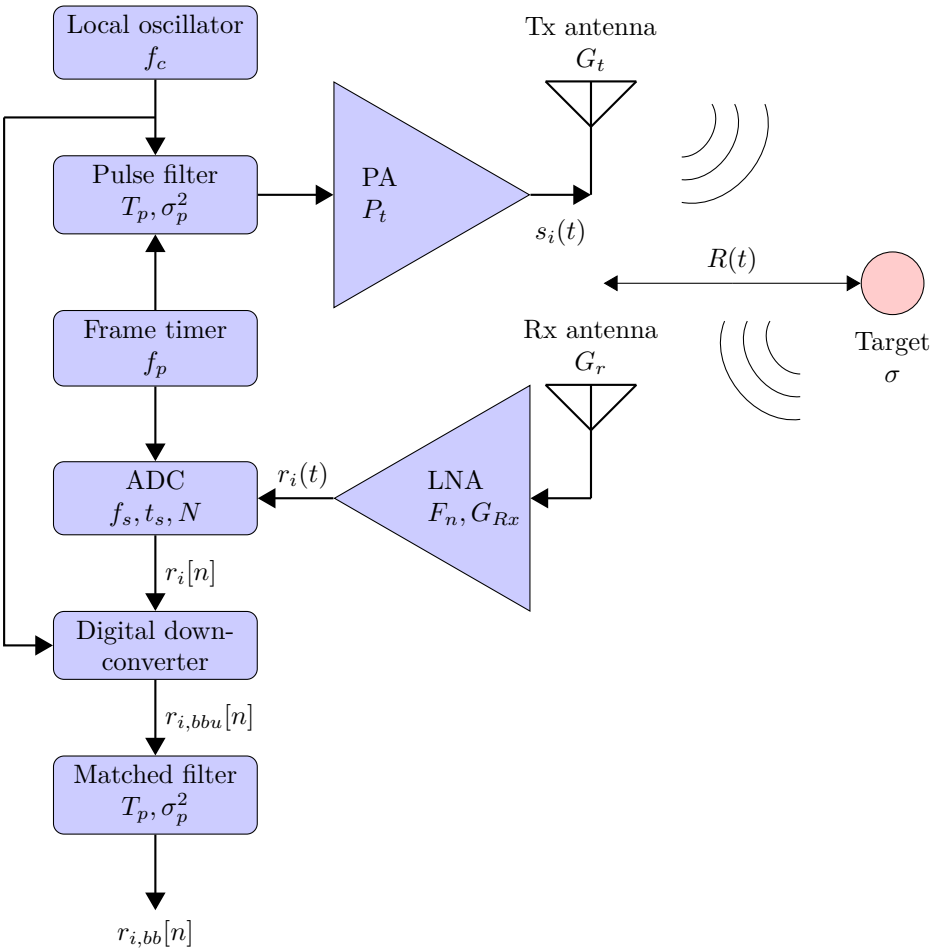

Figure 2.1.: Radar Module Block Diagram and Parameters

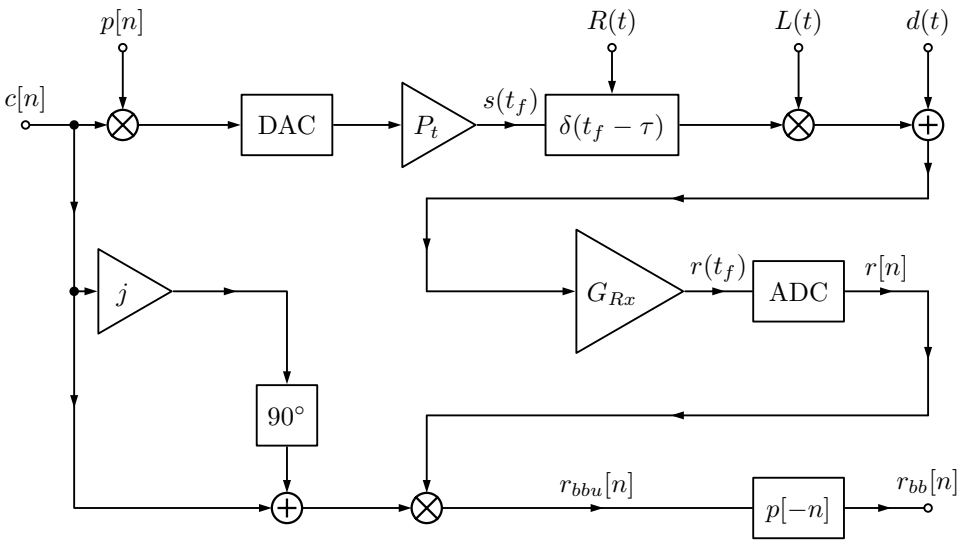

Figure 2.2.: Radar Module Signal Processing Steps and Parameters

2.1.2. Generating the Pulse

In the time domain, the transmitted signal for the ith frame is given by

$$s_i(t) = p(t - iT_f) \cdot \cos(2\pi f_c(t - iT_f) \tag{2.1}$$

which is the ideal output from the power amplifier following the DAC in figure 2.2 (we assume a normalized power level to leave P_t out of the equation). Here, T_f is the duration of the frame given as the inverse of the pulse repetition frequency f_p:

$$T_f = \frac{1}{f_p} \tag{2.2}$$

f_c is the radar carrier frequency, and $p(t)$ is the gaussian pulse shaping filter function defined as

$$p(t) = a \cdot \exp\left(-\frac{(t - \frac{T_P}{2})^2}{2\sigma_p^2}\right) \tag{2.3}$$

where a is the gain of the filter, T_p the duration of the pulse and σ_p^2 the variance defining the steepness of the filter lump.
Following the notation in [2], we define a local frame time variable t_f as

$$t = t_f + iT_f \qquad t_f \in [0, T_f] \tag{2.4}$$

Table 2.1.: Parameters used in Radar Module

Symbol	Parameter	Unit
f_p	PRF/framerate	Hz
T_f	Frame duration (inverse of f_p)	seconds
T_p	Pulse-shaping filter width	seconds
σ_p^2	Pulse-shaping filter steepness	seconds
P_t	Radar output power	Watts
f_c	Carrier frequency	Hz
G_t	Transmitting antenna gain	dBi
G_r	Receiving antenna gain	dBi
$R(t)$	Range to target	meters
σ	Target equivalent radar cross section (RCS)	m^2
F_n	Receiver amplifier noise figure	dB
G_{Rx}	Receiver amplifier gain	dB
f_s	ADC sampling frequency	Hz
t_s	ADC sampling offset	seconds
N	Samples per frame	

Table 2.2.: Variables used in Radar Module

Symbol	Parameter	Unit
i	Frame number	
n	Sample number within frame	
t	Global time since recording started	seconds
t_{f0}	Frame start time relative to global time	seconds
t_f	Local frame time (difference between global time and frame start time)	seconds

Table 2.3.: Signal Processing Steps in Radar Module

Symbol	Description	Inputs	Output	Page
	Local oscillator	f_c, t	$c(t)$	7
$H(f)$	Pulse-shaping filter and power amplifier	$c(t), T_p$, σ_p^2, P_t	$s(t_f)$	7
	Propagation delay	$R(t)$	$\tau(t)$	9
	Signal propagation loss	$R(t)$, G_t, G_r, σ, f_c	$L(t)$	9
	Receiver noise	F_n, f_s	$d(t)$	10
	Propagation	$s(t_f)$, $\tau(t)$, $L(t)$, $d(t)$, G_{Rx}	$r(t_f)$	9
ADC	Digitizer	$r(t_f)$, f_s, t_s	$r[n]$	11
DDC	Digital downconverter	$c(t), r[n]$	$r_{bbu}[n]$	12
$H^{-1}(f)$	Matched filter	$r_{bbu}[n], T_p, \sigma_p^2$	$r_{bb}[n]$	12

which gives us the relationship $t_f = t - iT_f$ when $0 < t_f < T_f$. The transmitted signal for frame i can then be written as

$$s_i(t_f) = p(t_f) \cdot \cos(2\pi f_c t_f) \tag{2.5}$$

2.1.3. Propagation Loss and Delay

The returned signal from a single target for the ith frame is given by

$$r_i(t_f) = G_{Rx} \cdot L(t_f) \cdot s_i(t_f - \tau(t)) = G_{Rx} \cdot L(t_f) \cdot p(t_f - \tau(t)) \cdot \cos(2\pi f_c(t_f - \tau(t))) \tag{2.6}$$

where $\tau(t)$ is the propagation delay equal to the radar pulse round-trip time calculated as

$$\tau(t) = \frac{2 \cdot R(t)}{c} = \frac{2 \cdot R(t_f + iT_f)}{c} \tag{2.7}$$

$L(t)$ is the free space loss given by the square root of the ratio between the transmitted power P_t and the received power P_r:

$$L(t) = \sqrt{\frac{|P_r|}{|P_t|}} \tag{2.8}$$

The received signal power P_r from a target at the LNA input is given by the radar equation [3]:

$$P_r = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 R_t^2 R_r^2} \tag{2.9}$$

where G_t and G_r is the gain of the transmitting and receiving antenna respectively, λ the wavelength of the radar carrier signal, σ the RCS of the target and R_t, R_r the range from the transmitting antenna and receiving antenna to the target respectively.

For the moving targets present in the signals we will be analyzing, R will be a time-dependent function $R(t)$. We assume that the radar module is a monostatic configuration, which means that similar antennas placed at nearly the same location is used for transmitting and receiving, which lets us simplify $G_t G_r$ to G^2 and $R_t^2 R_r^2$ to R^4 (or $R^4(t)$ to highlight its time dependency). We then have

$$P_r(t) = \frac{P_t G^2 \lambda^2 \sigma G_{Rx}}{(4\pi)^3 R(t)^4} \tag{2.10}$$

which gives us the expression for $L(t)$ as

$$L(t) = \sqrt{\frac{G^2 \lambda^2 \sigma}{(4\pi)^3 R(t_f)^4}} \tag{2.11}$$

and consequently, the expression for the received signal becomes

$$\begin{aligned} r_i(t_f) =& G_{Rx} \cdot L(t) \cdot p\left(t_f - \frac{2 \cdot R(t_f + iT_f)}{c}\right) \\ & \cdot \cos\left[2\pi f_c \left(t_f - \frac{2 \cdot R(t_f + iT_f)}{c}\right)\right] \end{aligned} \tag{2.12}$$

2.1.4. Receiver Noise

The amount of noise experienced at the input of the ADC depends on the antenna noise temperature, the receiver noise temperature, receiver bandwidth and receiver gain. We assume that the noise appears as gaussian white noise from which we can estimate an average noise power based on the aforementioned parameters.

We start by calculating an equivalent noise temperature T_e for the cascade formed by the antenna and receiver amplifier:

$$T_e = T_A + \frac{T_{Rx}}{G_{Rx}} \tag{2.13}$$

Where T_A is the antenna noise temperature, T_{Rx} the receiver noise temperature and G_{Rx} the receiver gain. The average noise power $N_{0,in}$ at the input of the receiver is calculated as

$$N_{0,in} = kT_eB \quad [\mathrm{W}] \tag{2.14}$$

where k is Boltzmann's constant, T_e the noise temperature from equation 2.13 and B the receiver bandwidth, which when sampling directly at carrier frequency corresponds to the ADC sampling frequency f_s. When referenced to the ADC input, $N_{0,in}$ is multiplied with the receiver gain G_{Rx}.

We will not include the receiver noise in the calculations in the subsequent sections, since we will assume that the matched filter stage described in section 2.1.7 will improve the signal to noise ratio to a level where the noise can be ignored.

2.1.5. Digitizing the Signal

The signal is digitized by sampling at $(t_s + nTs)$ where t_s is the frame offset time defining the minimum range of the radar, T_s is the sampling period defined by the ADC sampling frequency f_s as $T_s = \frac{1}{f_s}$ and $n = 0, 1, 2, ...N - 1$ where N is the number of samples to be recorded for each frame defining the maximum range or range span. The discrete-time signal is then given by

$$\begin{aligned} r_i[n] = r_i(t_s + nT_s) = G_{Rx} \cdot L(t_s + nT_s) \cdot p\left(t_s + nT_s - \frac{2 \cdot R(t_s + nT_s + iT_f)}{c}\right) \\ \cdot \cos\left[2\pi f_c \left(t_s + nT_s - \frac{2 \cdot R(t_s + nT_s + iT_f)}{c}\right)\right] \end{aligned} \tag{2.15}$$

Since the distance we expect a target to travel throughout the duration of a frame will be several orders of magnitude smaller than the range, we introduce the following simplification

$$R(t_s + nT_s + iT_f) \approx R(iT_f) \tag{2.16}$$

which is akin to saying that the range to a target will be approximately the same throughout the frame as it was at the beginning of the frame. We use this simpli-

fication to write equation (2.15) as

$$r_i[n] \approx G_{Rx} \cdot L(t_s + nT_s) \cdot p\left(t_s + nT_s - \frac{2 \cdot R(iT_f)}{c}\right) \cdot \cos\left(2\pi f_c \left(t_s + nT_s - \frac{2 \cdot R(iT_f)}{c}\right)\right) \tag{2.17}$$

2.1.6. Digital Down-Conversion

$r_i[n]$ will then need to be digitally down-converted to baseband by multiplying with a complex phasor with the same frequency as the radar carrier frequency:

$$\begin{aligned} r_{i,bbu}[n] &= r_i[n] \cdot \exp(-j2\pi f_c(iT_f + nT_s)) \\ &= G_{Rx} \cdot L(t_s + nT_s) \cdot p\left(t_s + nT_s - \frac{2 \cdot R(iT_f)}{c}\right) \\ &\cdot \frac{1}{2}\left[\exp\left(-\frac{j4\pi f_c R(iT_f)}{c}\right) + \exp\left(-j4\pi f_c \left(t_s + nT_s - \frac{R(iT_f)}{c}\right)\right)\right] \end{aligned} \tag{2.18}$$

2.1.7. Matched Filtering

Finally, the signal is filtered using a time-reversed version of the pulse-shaping filter function given in equation (2.3), which then forms a matched filter that maximizes the signal to noise-ratio and removes the high-frequency component present in equation (2.18):

$$r_{i,bb}[n] = \frac{1}{2} \cdot G_{Rx} \cdot L(t_s + nTs) \cdot p\left(t_s + nT_s - \frac{2 \cdot R(iT_f)}{c}\right) \cdot \exp\left(-\frac{j4\pi f_c R(iT_f)}{c}\right) \tag{2.19}$$

The frames that make up one recording are stored in matrix $\boldsymbol{\Upsilon}_i$, where $i = 0, 1, 2, \cdots I - 1$ and I is the number of frames in one recording.

2.1.8. Calculating Doppler Frequency Shift

The doppler frequency shift of the returned signal is given in [3] as:

$$f_d = \frac{-2v_r}{\lambda} \tag{2.20}$$

where the negative sign is used because an approaching target, which will cause a positive doppler frequency shift, will have a negative rate of change of the range and consequently a negative radial velocity. The radial velocity v_r is defined by

$$v_r = \frac{dR_j(t)}{dt} \tag{2.21}$$

and the carrier wavelength λ by

$$\lambda = \frac{c}{f_c} \tag{2.22}$$

By using the relation in equation (2.22) we can rewrite equation (2.19) highlight the relationship between the range and the phase of the baseband signal in terms of number of wavelengths:

$$\begin{aligned} r_{i,bb}[n] = \frac{1}{2} \cdot G_{Rx} \cdot L(iT_f) \cdot p\left(t_s + nT_s - \frac{2 \cdot R(iT_f)}{c}\right) \\ \cdot \exp\left(-\frac{j4\pi R(iT_f)}{\lambda}\right) \end{aligned} \tag{2.23}$$

We define the phase term ϕ_i as

$$\phi_i = \frac{-4\pi R(iT_f)}{\lambda} \tag{2.24}$$

Using the relations described in equations (2.20) and (2.21), the derivative of the phase term gives the doppler frequency of the signal:

$$f_{d,i} = \frac{\omega_{d,i}}{2\pi} = \frac{1}{2\pi}\frac{d\phi_i}{dt} = \frac{-2}{\lambda}\frac{dR(iT)}{dt} = \frac{-2v_{r,i}}{\lambda} \tag{2.25}$$

which is the same as equation (2.20).

2.2. Feature Extraction

The algorithms used for feature extraction was developed and described in [2]. They are presented here for reference and to show how we arrive at the data used as input to the target classification algorithms.

2.2.1. Feature Extraction Overview

An overview of the process steps and variables is shown in figure 2.3. We will work with time scales in two different dimensions throughout this chapter, and for this we use the terms *slow time* and *fast time*, which are linked to the discrete counter variables i and n, respectively.

Figure 2.4 shows how the slow and fast time scales relate to the counter variables in a radar recording consisting of I frames, each with N samples (or range-bins). i represents the number of the frame, and is conversely incremented at a rate equal to the pulse repetition frequency f_p. n represents the number of a sample within one frame, and is incremented at a rate equal to the ADC sampling frequency f_s.

Table 2.5 lists the radar parameters that we need to know to perform the feature extraction process, while table 2.6 lists the variables that will be used. The different process steps, along with on which page each step is described are listed in table 2.4.

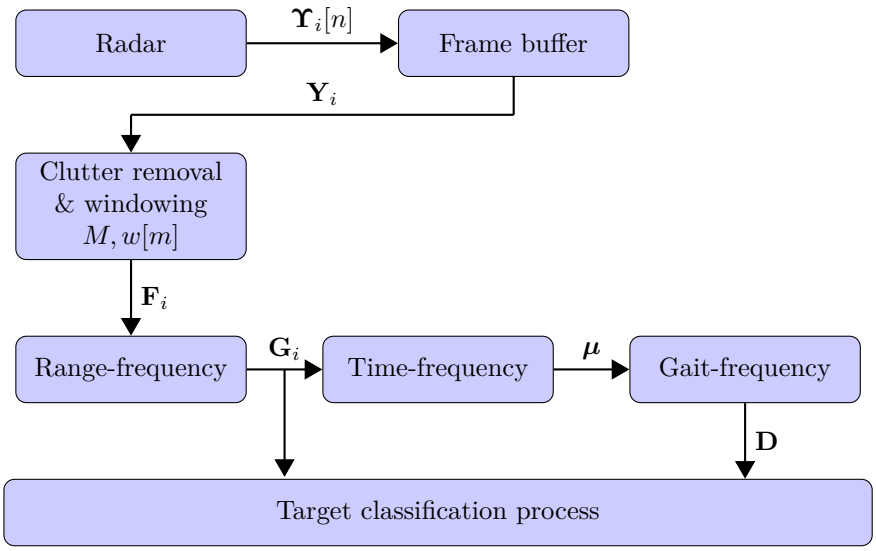

Figure 2.3.: Feature Extraction Process Block Diagram and Parameters

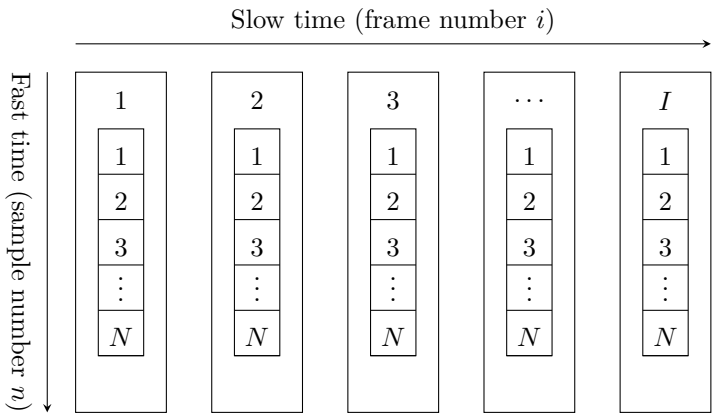

Figure 2.4.: Slow and fast Time Scales

Table 2.4.: Steps in the Feature Extraction Process

Description	Inputs	Output	Page
Frame buffer	$\boldsymbol{\Upsilon}_i[n], M$	$\mathbf{Y}_i$	18
Clutter removal & windowing	$\mathbf{Y}_i, M, w[m]$	$\mathbf{F}_i$	18
Range-frequency analysis	$\mathbf{F}_i, M, f_p,\ f_s, t_s$	$\mathbf{G}_i$	19
Time-frequency analysis	$\mathbf{G}_i, I, N$	$\boldsymbol{\mu}$	21
Gait-doppler analysis	$\boldsymbol{\mu}$	$\mathbf{D}_i$	22

Table 2.5.: Parameters used in the Feature Extraction Process

Symbol	Parameter	Unit
f_p	PRF/frame rate	Hz
T_f	Frame duration (inverse of f_p)	seconds
f_c	Carrier frequency	Hz
$R(t)$	Range to target	meters
f_s	ADC sampling frequency	Hz
t_s	ADC sampling offset	seconds
$w[m]$	Window function	
N	Samples per frame	
M	Window length	frames
σ_w	Window function variance	seconds

Table 2.6.: Variables used in the Feature Extraction Process

Symbol	Parameter	Unit
i	Frame number (slow time)	
n	Sample number within frame (fast time)	
t	Global time since recording started	seconds
t_{f0}	Frame start time relative to global time	seconds
t_f	Local frame time (difference between global time and frame start time)	seconds
I	Total number of frames in recording	
T	Global time when recording stopped	seconds
f_d	Doppler frequency	Hz

2.2.2. Frame Buffer, Clutter Removal and Windowing

The input to the analysis process is the digitally down-converted and low-pass filtered frames $\boldsymbol{\Upsilon}_i[n]$ where $i = 0, 1, 2, \cdots, I-1$ of which I is the number of frames recorded given by $I = f_p \cdot T$, T is the time over which the frames were recorded and $n = 0, 1, 2, \cdots, N-1$ where N is the number of samples per frame.

We start by creating the framebuffer matrix $\mathbf{Y}_i$, given by

$$\mathbf{Y}_i = \begin{bmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,M-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,M-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N-1,0} & y_{N-1,1} & \cdots & y_{N-1,M-1} \end{bmatrix} \tag{2.26}$$

where M is the window length and

$$y_{n,m} = \boldsymbol{\Upsilon}_{i+m-\frac{M}{2}}[n] \qquad m = 0, 1, 2, \cdots, M-1 \tag{2.27}$$

which tells us that each column contains one frame with the center column containing frame $\boldsymbol{\Upsilon}_i[n]$, and each row representing the activity within one range bin over the window time period, which will be M/f_p seconds long.

From this we create matrix $\mathbf{F}_i$ where stationary clutter has been removed and a gaussian window function has been applied to avoid smearing of the doppler frequency spectrum in the range-frequency analysis step. $\mathbf{F}_i$ i given by

$$\mathbf{F}_i = \begin{bmatrix} z_{0,0} & z_{0,1} & \cdots & z_{0,M-1} \\ z_{1,0} & z_{1,1} & \cdots & z_{1,M-1} \\ \vdots & \vdots & \ddots & \vdots \\ z_{N-1,0} & z_{N-1,1} & \cdots & z_{N-1,M-1} \end{bmatrix} \tag{2.28}$$

where

$$z_{n,m} = w[m] \cdot (y_{n,m} - \frac{1}{M} \sum_{m=0}^{M-1} y_{n,m}) \tag{2.29}$$

$$w[m] = \exp\left(-\frac{1}{2}\left(\frac{m-(M-1)/2}{\sigma_w(M-1)/2}\right)^2\right) \tag{2.30}$$

where the step of subtracting the complex average value calculated across the

window length in equation (2.29) serves to remove stationary clutter. N and M are the same as in equation (2.26).

The duration of the gaussian window function $w[m]$, as given in equation (2.30), is in theory infinitely long. However, it decays quite quickly, and can safely be truncated once it has decayed below a certain level.

How quickly it rises and decays is governed by the window steepness σ_w^2. The exact value of this parameter is not critical to the functionality of the window, but should be set wide enough to attenuate the data as little as possible, while still decaying sufficiently towards the end of the window. If we want it to decay to e.g. one tenth of the maximum value towards the ends, the *Full width at a tenth of a maximum (FWTM)* of the gaussian window function is found by treating m as a continuous variable and solving equation (2.30) to find the distance between the two points where $w(m) = 0.1$ (since the maximum value is 1). This distance turns out to be

$$\text{FWTM} = 2\sqrt{2\ln 10}\sigma_w \approx 4.29193\sigma_w \tag{2.31}$$

For the FWTM to be equal to the window duration we simply set $\text{FWTM} = \frac{M-1}{f_p}$, which lets us express σ_w in terms of M:

$$\sigma_w = \frac{M-1}{4.29193 \cdot f_p} \tag{2.32}$$

Both the clutter removal and windowing steps are performed in the slow-time dimension, which follows the row direction in matrix $\mathbf{F}_i$.

2.2.3. Range-Frequency Analysis

The range-doppler matrix $\mathbf{G}_i$ is calculated for each point in time using $\mathbf{F}_i$ as input. It is defined as:

$$\mathbf{G}_i = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,N-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{M-1,0} & g_{M-1,1} & \cdots & g_{M-1,N-1} \end{bmatrix} \tag{2.33}$$

where

$$g_{k,n} = \left| \sum_{m=0}^{M-1} z_{n,m} \cdot \exp\left(-\frac{j2\pi km}{M}\right) \right|^2 \quad k = 0, 1, 2, \ldots, M-1 \tag{2.34}$$

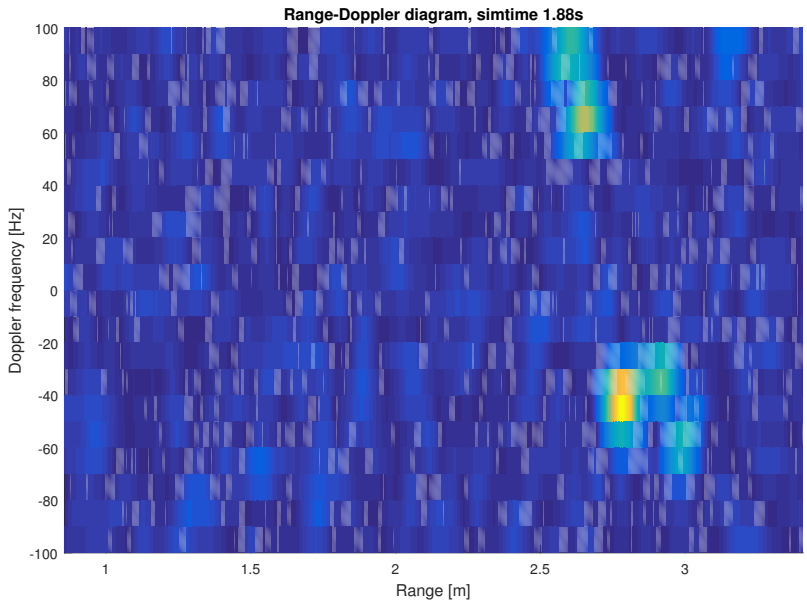

Figure 2.5.: Example of Range-Doppler Power Spectrum

An example plot of the range-doppler matrix $\mathbf{G}_i$ for one frame is shown in figure 2.5. The color represents the power at a certain range and doppler frequency, where a brighter color corresponds to more power. The doppler frequency is plotted on the y-axis with a scale corresponding to the pulse repetition frequency f_p and a resolution equal to the window length M:

$$f_d[m] = f_p \left(\frac{m}{M-1} - \frac{1}{2} \right) \text{[Hz]} \quad m = 0, 1, 2, \ldots, M-1 \tag{2.35}$$

and the range along the x-axis with a scale corresponding to the ADC sampling frequency f_s:

$$R[n] = \frac{c}{2} \left(t_s + \frac{n}{f_s} \right) \quad n = 0, 1, 2, \ldots, N-1 \tag{2.36}$$

In this plot, we can visually distinguish five scatterers, all in the range between 2.5 and 3 meters. Three of the scatterers appear at around $f_d = -40$ Hz, and the remaining two scatterers around $f_d = 80$ Hz. This plot was generated using synthetic radar data where $f_p = 200$ Hz, $M = 20$ frames, $f_s = 41$ GHz, $t_s = 6$ ns, and $f_c = 6.8$ GHz. Using equation (2.25) and finding the wavelength $\lambda = \frac{c}{f_c}$ this gives a radial velocity $v_r = 0.9$ m/s and $v_r = -1.5$ m/s, respectively (a negative

velocity means the target is approaching the radar).

As can be seen in figure 2.5, and also by examining equation 2.33, matrix $\mathbf{G}_i$ is transposed compared to the $\mathbf{Y}_i$ and $\mathbf{F}_i$ matrices (equations (2.26) and (2.28), respectively). This means that the slow-time variable m now follows the vertical axis and that the fast-time variable n follows the horizontal axis, since this is more intuitive by giving us the range horizontally in figure 2.5.

2.2.4. Time-Frequency Analysis

The micro-doppler signature matrix $\boldsymbol{\mu}$ is given by

$$\boldsymbol{\mu} = \begin{bmatrix} u_{0,0} & u_{0,1} & \cdots & u_{0,I-1} \\ u_{1,0} & u_{1,1} & \cdots & u_{1,I-1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{M-1,0} & u_{M-1,1} & \cdots & u_{M-1,I-1} \end{bmatrix} \tag{2.37}$$

where each column in u contains the row-wise sum of all range bins in each range-doppler matrix g_i:

$$u_{m,i} = \sum_{n=0}^{N-1} g_{n,m,i} \quad i = 0, 1, 2, \ldots, I-1 \tag{2.38}$$

where the added i index indicates that this is the $g_{n,m}$ matrix associated with frame i, which was defined in equation (2.33).

Figure 2.6 shows an example plot of a time-frequency matrix $\boldsymbol{\mu}$. As in figure 2.5, brighter colors indicate higher power. The parameters used here are recording time T of 2 seconds, pulse repetition frequency f_p of 200 Hz (giving I of 400 frames) and window length M of 27 frames. The data was synthetically generated. It can be visually interpreted that the target is approaching the radar, given the positive average doppler frequency shift, and that the gait frequency is approximately 2 Hz, given the repetition rate of the pattern in the plot.

The brighter background color on the left side of the plot is caused by a combination of the poorer signal to noise ratio experienced when the target is far from the radar, and the normalization of the power level within the frame (without normalization, both the background color (noise) and the signal pattern would be darker on that end of the plot). We can also see from the dark horizontal line in the middle of the plot that the clutter removal step also removes much of the

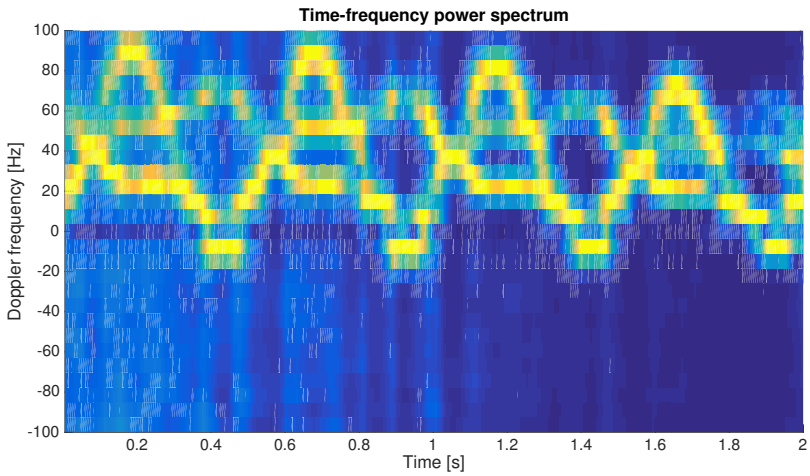

Figure 2.6.: Example of Time-Frequency Power Spectrum

uncorrelated noise around the 0 Hz doppler frequency.

2.2.5. Gait-Frequency Analysis

The final analysis of the recorded data is the creation of the gait-doppler matrix $\mathbf{D}$, defined by

$$\mathbf{D} = \begin{bmatrix} d_{0,0} & d_{0,1} & \cdots & d_{0,I-1} \\ d_{1,0} & d_{1,1} & \cdots & d_{1,I-1} \\ \vdots & \vdots & \ddots & \vdots \\ d_{M-1,0} & d_{M-1,1} & \cdots & d_{M-1,I-1} \end{bmatrix} \tag{2.39}$$

where each row $d_{m,k}$ is the DFT of the corresponding row in the micro-doppler signature matrix $\boldsymbol{\mu}$:

$$d_{m,k} = \left| \sum_{i=0}^{I-1} u_{m,i} \cdot \exp\left(-\frac{j2\pi ki}{I}\right) \right| \quad k = 0, 1, 2, \ldots, I-1 \tag{2.40}$$

The result of this operation gives us the frequencies at which events are occurring in the time-frequency plot, which again will tell us something about the frequency components of the motion pattern of the target(s).

Figure 2.7 shows an example plot of a gait-doppler matrix. We can see that the target is moving away from the target given the negative average doppler frequency,

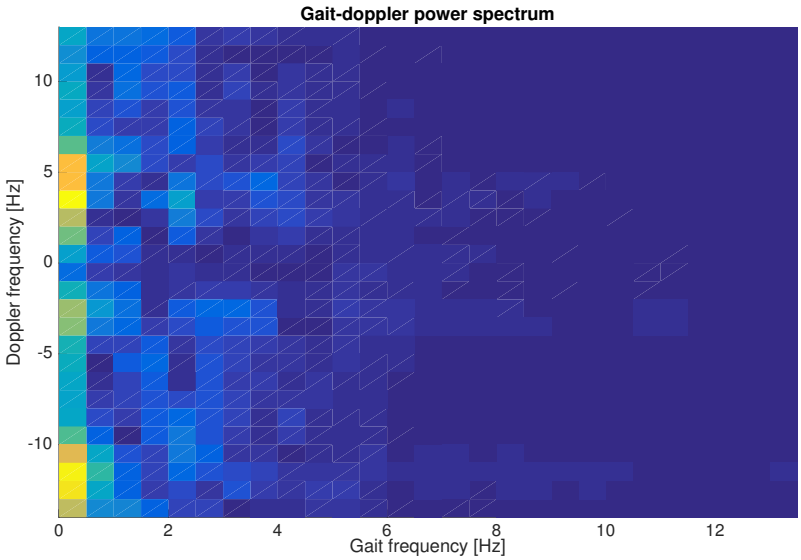

Figure 2.7.: Example of Gait-Frequency Power Spectrum

and that we have some bright spots around the 2 Hz gait frequency. Little or no information is found beyond approximately 5 Hz. The large concentration of power around the 0 Hz gait frequency is most probably caused by the noise, or more precisely the poorer signal to noise ratio experienced when the target is far away (seen as the brighter background on the left half of figure 2.6).

2.2.6. Feature Extraction Summary

The feature extraction process seeks to manipulate the recorded radar data in order to extract information that has a physical meaning, such as the gait frequency (found from the $\mathbf{D}$ matrix) and the stride length of a moving target (found by combining the information about the gait frequency and the average velocity). While this might not prove all that useful for an automated target classification system, which does not have any understanding of these physical phenomena in the first place, it also serves the task of reducing the size of the input data, which will both make the classification process less computationally intensive, and simplify the task of shaping the data into a standardized format.

Nevertheless, we will keep using the feature extraction processes presented in this chapter for preparing the input data for the classification system, and leave any optimization for machine input friendliness as a future task discussed in 5.

3. Pattern Recognition and Classification with Neural Networks

3.1. Artificial Neural Networks

Artificial neural networks (ANNs) are mathematical models that attempt to resemble the way an animal brain learns how to predict the outcome of an event based on the knowledge about the outcome of previous, similar events. Their primary usage is pattern recognition problems (in fact, in its purest sense, pattern recognition is the only task an ANN is able to perform), of which classification problems are a subcategory. We therefore identify ANNs as a possible solution to our target classification problem, and will delve deeper into this subject throughout this chapter.

We adopt the notation used by Heaton [8] and others, where the term 'neural network' always refers to ANNs, and is not to be confused with biological neural networks (BNN). From this point on, whenever the term 'neural network' is used, it always refers to an ANN, unless specifically stated otherwise.

3.1.1. The Neuron

Heaton [8] describes the neuron as the basic building block of the neural network. It can take on one or both of the roles as hidden or output neuron.

Let $\mathbf{x}$ be an input vector of size R presented to the neuron's input nodes, and $\mathbf{w}$ a vector of the same size containing the weight values associated with the respective input nodes. The neuron's output value $f(\mathbf{x},\mathbf{w})$ is then given by

$$f(\mathbf{x},\mathbf{w}) = \phi\,(y) \tag{3.1}$$

$$y = \sum_{r=0}^{R-1} (w_r \cdot x_r) \tag{3.2}$$

where $\phi(y)$ is the neuron's activation function, and y is the sum of all inputs to the neuron, multiplied with the weights associated with each of the neuron's inputs [8].

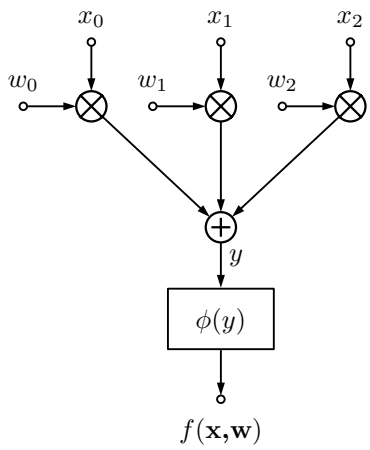

Figure 3.1.: Generic Model of a Neuron

The activation function is the neuron's transfer function. We are at liberty to choose any function as activation function, however, as one might suspect, there are certain types of functions that are more suit for the task than others. Section 3.4.6 will introduce some common activation functions that will be used in this project.

The weight values $\mathbf{w}$ are essential to the functionality of the neural network. Most oftenly they are initialized to a random value, and then adjusted to suit the patterns the network is designed to recognize through the training process which is described in section 3.3. The number of input nodes depends on the network structure and the role of the neuron within the network, and will be discussed in the next section.

For equation 3.1 to provide meaningful output, we see that neurons, and also the neural networks, take real numerical values as their only valid input data type.

Figure 3.1 shows a schematic model of a neuron with three input nodes and weight values.

3.1.2. Neural Network Structure

A neural network consists of neurons organized in layers. Figure 3.2 shows an example of a neural network with an input layer, two hidden layers and an output layer. A neural network does not need to have all three types of layers, and as mentioned in the previous section, one neuron can take the role of both hidden and output neuron (in which the network would only have one neuron layer).

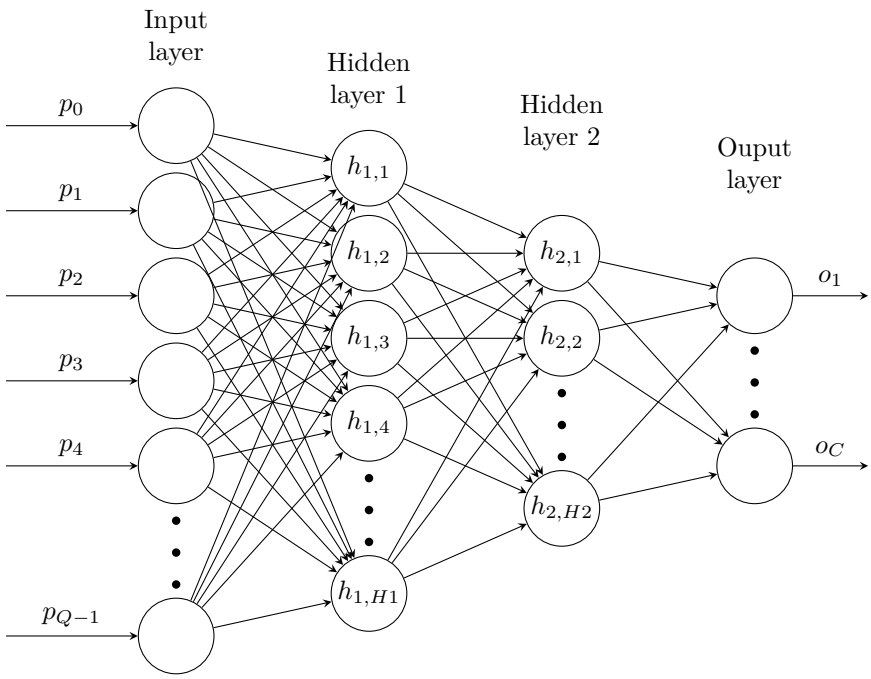

Figure 3.2.: Generic Model of a Feedforward Network with two Hidden Layers

The network in figure 3.2 is a feedforward network, where the neurons only feed their output values forward to the next layer, and never backwards. Feedforward networks are one of the most commonly used neural network architectures due to their versatility and is especially popular in classification applications [8], which is why we will primarily be investigating this type of neural network for our problem.

Neurons in a feedforward network connect to each other in the following ways:

- The first layer, the input layer, does not have neurons, just input nodes (in some texts, the input layer is not considered a layer at all - just inputs to the first hidden layer)
- Neurons in hidden layers and output layers are connected to all the neurons or nodes in the preceding layer, which means that each neuron has the same number of input nodes and associated weight values as the number of neurons or input nodes in the preceding layer
- Neurons only have one output node, ie. they only output one value. This same value is fed to all neurons connected this neuron's output. In the final layer, the output layer, the number of neurons correspond to the number of values the network is designed to output

The example network shown in figure 3.2 is drawn so to indicate that each layer is consecutively smaller (in terms of number of neurons) than the previous layer. This is common in applications such as classification problem, where a large input vector is to be categorized into a limited number of target groups. It should be noted, however, that this is not always the case, and that there are applications where a layer will have more neurons than the preceding layer.

All neurons in one layer have the same activation function. The different layers can, however, have different activation functions.

3.2. Neural Network Evaluation

Before advancing to the neural network training process, we will need some means of evaluating the performance of a neural network.

3.2.1. Error Functions

Error functions quantify the performance of a neural network in a single score value based on the difference between the ideal and actual output for all output neurons. Error functions are also known as performance functions or objective functions, and in addition to assessing the performance of a neural network, it also serves an important task in the training process, which will be explained in section 3.3.

The input to the error function are the vectors $\mathbf{o}$ and $\hat{\mathbf{o}}$, where $\mathbf{o}$ represents the desired, ideal output from all output neurons for a given set of input data, and $\hat{\mathbf{o}}$ holds the actual output from the network in its current training state when presented with the same set of input data.

Several different error functions exist. For classification problems, a common error function is the cross-entropy error function. It is defined as

$$E(\mathbf{o},\hat{\mathbf{o}}) = -\frac{1}{U}\sum_{u=1}^{U}\sum_{c=1}^{C}\left[o_{u,c}\ln(\hat{o}_{u,c}) + ((1-o_{u,c})\cdot\ln(1-\hat{o}_{u,c})\right] \tag{3.3}$$

where U is the number of training data feature sets, C the number of target classes, $o_{u,c}$ is the desired output from output neuron c for input feature set u, and $\hat{o}_{u,c}$ is the actual output from the network for the same input feature set.

A neural network is regarded to perform better when the score calculated from the error function is as low as possible, which means that the degree of error is lower. The second term in the cross-entropy error function requires that the

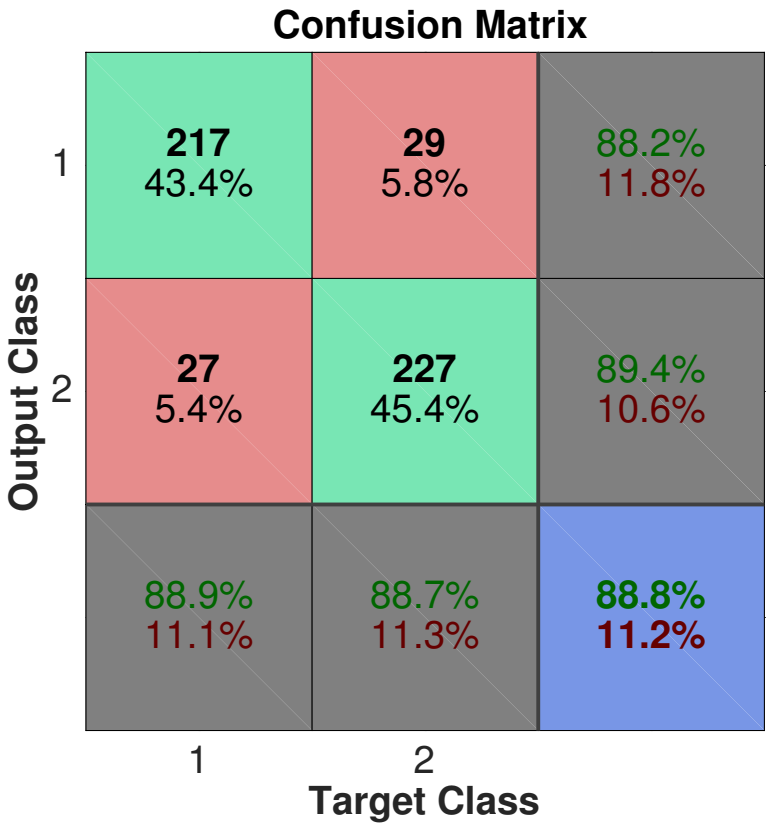

Figure 3.3.: Example of Confusion Plot with 500 Input Feature Sets

output values are in the range $\langle 0, 1 \rangle$ for this to hold true. For networks that are to be evaluated using the cross-entropy function, the network must be designed to give output in this range.

3.2.2. Confusion Plots

A confusion plot is a visual representation of a neural network's degree of correct and erroneous classifications. An example confusion plot for a network with two possible outputs is shown in figure 3.3. The horizontal axis, 'Target Class' shows which class the input feature sets belongs to, while the vertical axis, 'Output Class' shows which class the network predicts that the feature set belongs to. A full explanation of the numbers in each cell is given in table 3.1.

Table 3.1.: Explanation of Confusion Plot Cells

Number of elements correctly predicted as class 1	Number of elements belonging to class 2 but predicted as class 1	Percentage of elements predicted as class 1 which was correct Percentage of elements predicted as class 1 which actually belonged to class 2
Number of elements belonging to class 1 but predicted as class 2	Number of elements correctly predicted as class 2	Percentage of elements predicted as class 2 which was correct Percentage of elements predicted as class 2 which actually belonged to class 1
Percentage of elements belonging to class 1 which was predicted as class 1 Percentage of elements belonging to class 1 which was predicted as class 2	Percentage of elements belonging to class 2 which was predicted as class 2 Percentage of elements belonging to class 2 which was predicted as class 1	Percentage of elements predicted correctly Percentage of elements predicted wrong

3.3. Neural Network Training

Training is the process of adjusting a neural network to fulfill the tasks it was designed for. Prior to training, the output from a neural network is garbage that serves no purpose. The training process achieves this adaption by adjusting the neuron input weights to reduce the error produced when presenting a given input feature set to the network.

Successful training of a neural network requires access to a large set of training data. Gathering this data is usually the most cumbersome task when attempting to solve a problem using neural network.

Many approaches to neural network training exists. Training can be performed either supervised, in which the network is presented with a desired outcome for a given input feature set, or unsupervised, in which the network tries to figure out patterns in the input data without knowing the desired outcome.

We will in this text mainly focus on the training methods we will actually be using in our application. We will start by describing the process of training a network with one hidden layer only, in a supervised fashion. Training a network with multiple hidden layers requires an expansion of this process known as *deep learning*, which will be introduced in section 3.3.3.

Prior to training, the weights in the network are initialized to a random value. How the initial weight values are found is actually a factor affecting to which degree the network can be successfully trained, which has led to weight initialization being a subject of research in itself. According to Heaton, the most promising algorithm for weight initialization is the Xavier algorithm, which generates random numbers with a normal (gaussian) distribution [8].

3.3.1. Weight Update with Error Gradients and Backpropagation

Training is an iterative process, where for each iteration, the network process some training data, calculates new values for the weights in the network, and evaluate the result. Different approaches to how many elements of the training data that shall be processed for each iteration exists, the most common ones being *online training*, where only one training data element is processed for each iteration, and *batch training*, where a number of training data elements are processed for each iteration.

Recall that vector $\mathbf{w}$ contains the weight values for the entire network. Let $\Delta\mathbf{w}$ contain the *weight deltas* which is the amount of change to be applied to each weight after an iteration has completed. We use the counter variable z to keep track of the number of iterations, and since both $\mathbf{w}$ and $\Delta\mathbf{w}$ will change for each iteration, we define them to be functions of z as $\mathbf{w}(z)$ and $\Delta\mathbf{w}(z)$. We then have

$$\mathbf{w}(z+1) = \mathbf{w}(z) + \Delta\mathbf{w}(z) \qquad z = 1, 2, 3, \ldots, Z \tag{3.4}$$

where Z is the total number of iterations to perform during one *epoch*, where one epoch is completed when the entire set of available training data has been evaluated one time.

$\Delta\mathbf{w}(z)$ contains the weight deltas Δw_r associated with neuron r, where $r = 1, 2, 3, \ldots, R$ and R is the total number of input nodes in all hidden and output neurons in the network. The weight deltas are found using

$$\Delta\mathbf{w}(z) = -\eta\nabla E + \alpha\Delta\mathbf{w}(z-1) \tag{3.5}$$

where ∇E is the error gradient, which is the partial derivative of the error function $E(\mathbf{o},\hat{\mathbf{o}})$ with respect to the weight vector:

$$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \frac{\partial E}{\partial w_3} \\ \vdots \\ \frac{\partial E}{\partial w_R} \end{bmatrix} \tag{3.6}$$

where as we remember from section 3.2.1 we are at liberty to choose the error function $E(\mathbf{o},\hat{\mathbf{o}})$ as we desire.

Equation 3.6 reminds us that the output from the error function $E(\mathbf{o},\hat{\mathbf{o}})$ depends on the weight vector $\mathbf{w}$, since it is used in calculating the output vector $\mathbf{o}$, which we can see from equations (3.1) and (3.2). If we imagine a neural network with only two weight values, the output from the error function with respect to the weight values would form a continuous surface, for which the gradient can be calculated to tell us in which direction the weight values must change in order to converge towards the lowest point on the surface - which will be the weight values producing the lowest error score.

We call this surface the error surface, and although it cannot be interpreted as a surface in three-space in networks with more than two weight values, the concept remains the same, we want to find the lowest point on the error surface, which is found through the gradient descent method of equation 3.5, which in an iterative manner moves along the error surface in the direction of the steepest negative gradient.

Equation (3.5) also contains parameters η and α. η is the network's learning rate. It is usually chosen to a value less than 1, to prevent the training process from overstepping the optimal point on the error surface. α is the network's momentum, a factor added to prevent the network from getting stuck at a local minimum on the error surface which is not the global minimum by adding a part (usually less than 1) of the previous weight delta into the calculation of the current weight delta.

The error gradient is calculated individually for each weight w_r. If w_r is the weight associated with input node r to neuron i, where i can be any of the hidden or output neurons in the network, the error gradient for that input node is calculated

as

$$\frac{\partial E}{\partial w_r} = x_r \cdot \delta_i \tag{3.7}$$

where x_r is the value presented to that specific input node when processing the current training data element, and δ_i is the *node delta* associated with the neuron the node is input to. The node delta is a helper value which lets us calculate the error gradient without performing the actual partial differentiation of the error function. It is calculated differently based on which error function that is in use, and whether the neuron is in the output layer or in an internal layer.

Figure 3.4 shows a simple feedforward network with two hidden neurons h_1 and h_2, and output neuron o_c. When using the cross-entropy error function shown in equation (3.3), the node deltas for the output layer is given by

$$\delta_c = \hat{o}_c - o_c \tag{3.8}$$

which is simply the difference between the desired and actual output for the current output neuron. For neurons in an internal layer, such as the hidden neurons, the node delta formula does not depend on the chosen error function on the output, but rather on the activation function for the neurons in that layer. Node deltas for hidden neuron h is defined as

$$\delta_h = \phi'_h \left(\sum_{c=1}^{C} w_{h,c} \delta_c \right) \tag{3.9}$$

where ϕ'_h is the derivative of the hidden neuron's activation function, $w_{h,c}$ the weight value associated with the connection between the current hidden neuron and output neuron c, and δ_c the node delta of output neuron c, where $c = 1, 2, 3, \ldots, C$ are all output neurons the hidden neuron is connected to. This backward propagation of errors, or *backpropagation*, where the error at the output is used for calculating the gradient at a previous layer, is one of the most common ways of training a feedforward neural network.

Training continues over multiple epochs until a predefined termination condition is met. Examples of such conditions are:

- The error gradient falls below a certain limit, ie. we are at or sufficiently close to the lowest point on the error surface
- The score from a chosen performance function falls below a certain limit
- A predefined number of training epochs are completed

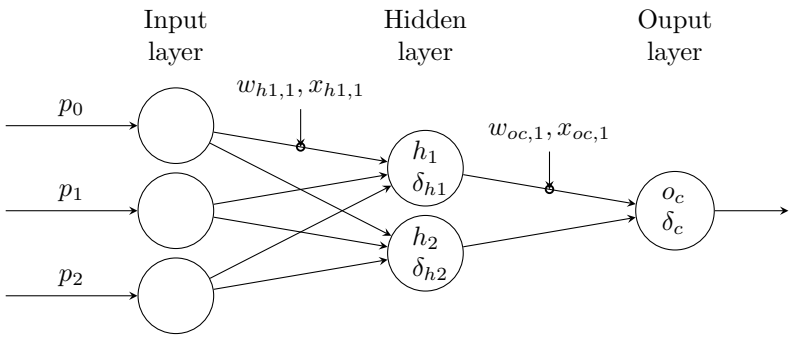

Figure 3.4.: Parameters related to calculating Error Gradient and Weight Deltas

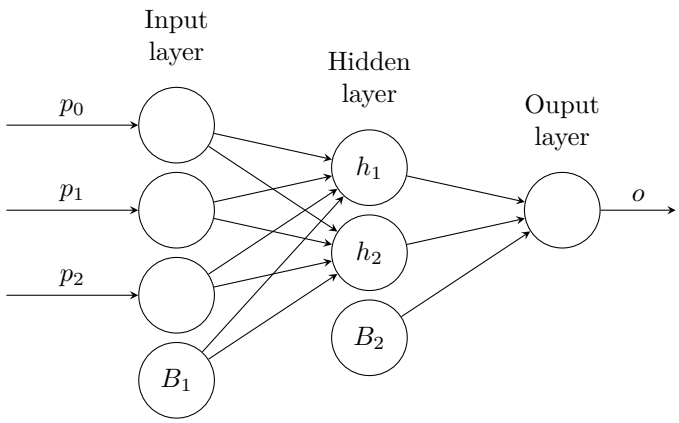

Figure 3.5.: Generic feedforward Network with Bias Neurons

3.3.2. Bias Neurons

Bias neurons are neurons with no input nodes that output a constant value. Bias neurons are placed at the end of one or more of the layers in the network, and allows the network to shift the point where the activation function of the neurons in the succeeding layer intercept the y-axis. The output value is normally set to 1, but any sensible value will work since the training process will adjust the weight value it is multiplied with to work as it is supposed to. This will add an additional degree of freedom to the network's learning process [8].

Figure 3.5 shows how bias neurons are connected to the succeeding layers in a feedforward network.

3.3.3. Deep Architectures and Deep Learning

The network in figure 3.2 is considered a *deep architecture* network, a term describing networks with more than one hidden layer. Using multiple hidden layers allows the network to learn even more complex patterns with less complex network structures and also requires less training data than what would be the case with a single hidden layer. This stems from the fact that a network with e.g. two hidden layers with 100 and 50 neurons each, respectively, forms 5000 neural interconnections while just requiring to train 150 neurons. A network with a single hidden layer would need 5000 neurons to achieve the same number of connections.

Deep architectures have become increasingly popular in the last decade due to the advent of *deep learning* methods. While the advantages of multiple hidden layers were identified quite early in the history of neural network research, it was difficult to obtain good results with conventional training methods [6]. Although it should seem trivial to expand the methods described in section 3.3 to calculate the node deltas and gradients of multiple hidden layers, doing so rarely resulted in a properly trained network.

With deep learning methods, the hidden layers are trained one at a time in a semi-unsupervised fashion, allowing the network to learn features in the training data at different level of abstraction for each hidden layer, from the raw input data (number sequences such as $[0.45, 1.2, 0.99, 0.01]$) at the inputs to an interpretation understandable by humans at the output (Bengio [6] gives an example of a computer being able to output 'sitting man' when presented with a picture showing a sitting man).

After training the individual hidden layers by themselves, the full network can then be trained further using conventional backpropagation methods.

3.3.4. Training with Autoencoders

Autoencoders are an essential topic within deep learning. An autoencoder is a neural network with the same number of input and output nodes, and a single hidden layer with a lower number of neurons than the input and output layers. The autoencoder is then trained with the same vector used as both input and desired output, which causes the autoencoder to attempt to recreate its input at its output. This allows the autoencoder to learn a compressed representation of the input data.

Figure 3.6 shows an example of an autoencoder designed to learn to represent an input feature set with 784 elements using a hidden layer with 100 neurons. In

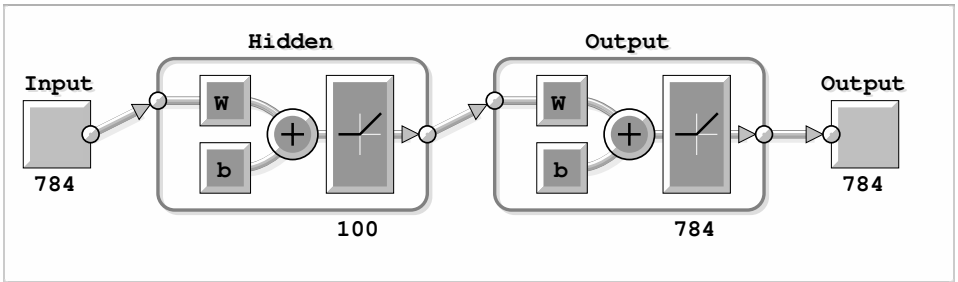

Figure 3.6.: Example of Autoencoder Network

this figure, the rounded rectangles represent a neuron layer, with the number of neurons in the layer displayed below it. The size of the input and output is shown at each end.

When training a multi-layer feedforward network using deep learning techniques, one creates an autoencoder network for each hidden layer, where the autoencoder's hidden layer has the same size and the same activation function as the corresponding hidden layer in the main network. The autoencoders are then trained one at a time in the following way:

- The first autoencoder, corresponding to the first hidden layer in the main network, has the same input size as the main network, and is trained using the training data meant to be input to the main network.
- After training the first autoencoder, the weight values from the hidden layer in the first autoencoder are copied to the first hidden layer in the main multi-layer network
- One then creates a new network with just an input and an output layer, where the output layer has the same size as the hidden layer in the first autoencoder, and the weights from the hidden layer in the first autoencoder are copied to the output layer of the new network.
- The new network is then used to create a new, compressed representation of the training data by using it to process the original training data. The new training data will have a dimensionality equal to the size of the first hidden layer of the main network.
- The second autoencoder, corresponding to the second hidden layer in the main network, has an input and output layer, both with the same size as the size of the first hidden layer of the main network. It also has a hidden layer with a smaller size than the input and output layer, corresponding to the size

of the second hidden layer in the main multi-layer network.

- The second autoencoder is trained in the same manner as the first, but using the compressed training data as input and desired output. The weights of the hidden layer of the second autoencoder is copied to the second hidden layer of the main multi-layer network.
- The process is repeated for all hidden layers of the main network. New training data with even higher level of compression is created by processing the compressed training data generated in the previous step, for as many iterations as necessary.

3.4. Designing a Neural Network for Radar Target Classification

3.4.1. Target Classification Process Parameters

Table 3.2 shows the parameters and variables that will be encountered when describing the target classification process throughout this chapter (except for those that were already defined in chapter 2), as well as on what page you can find their definitions.

3.4.2. Preparing the Input Data

Remember from section 3.1 that the feature set input to a neural network should be organized in a one-dimensional vector of floating point numbers, since the network input layer only has one dimension. This will be vector $\mathbf{p}$ in figure 3.2. The vector contains the observation data (recorded from the radar, or generated synthetically), and has a size sufficiently large to contain the amount of information required to identify the target. Each element in the vector corresponds to an input node of the neural network.

In addition, when dealing with labeled training data, we also need a vector with a size equal to the number of target classes, which contains the value 1 in the vector element corresponding to the class the current target belongs to, and zero in all other locations. This vector is called the label vector, and is only used for training purposes.

In order to minimize the effects of variations in power level in a signal (we do not count the average signal power level, which is proportional to the target's range and radar cross section, to be a feature useful for classification, since large pets and small animals can assume a radar cross section (RCS) within the same order of magnitude). To achieve this, we perform a normalization of the input data (see equation (3.11)).

We will create two variants of the input feature vector for classification:

- Type 1: a vector based on the gait-doppler matrix $\mathbf{D}$ defined in equation (2.39)
- Type 2: a vector based on the center frame (column) in framebuffer matrix $\mathbf{F}_i$ (defined in equation (2.33)) where the target has the largest observed distribution in range, and the gait-doppler matrix $\mathbf{D}$.

Table 3.2.: Parameters and Variables used in the Target Classification Process

Symbol	Parameter	Page
$\mathbf{p}_1$	type 1 input vector	40
$\mathbf{p}_2$	type 2 input vector	43
$\mathbf{s}$	label vector	
q	sample number in input vectors	
Q_1	type 1 input vector size	40
Q_2	type 2 input vector size	43
$P_{\text{avg},i}$	Average power in a frame	42
SAR	Signal to average power ratio in frame	42
$P_{\text{th},i}$	Threshold level between signal and noise	42
$n_{f,i}$	First sample in frame to exceed $P_{th,i}$	42
$n_{l,i}$	Last sample in frame to exceed $P_{th,i}$	42
$N_{e,i}$	Target extent within frame	43
I_α	Number of columns in $\mathbf{D}$ matrix used in classification	40
$\mathbf{f}_L$	frame with largest target extent	43
U	Number of feature sets (and labels) in training data	
C	Number of target classes/size of label vector	
H_1	Size of neural network hidden layer 1	
H_2	Size of neural network hidden layer 2	
$\phi_l(y)$	Logistic sigmoid activation function	45
$\phi_r(y)$	ReLU activation function	46
$\phi_s(y)$	Softmax activation function	47

3.4.3. Type 1 Feature Vector

Given the gait-doppler matrix $\mathbf{D}$ as described in equation (2.39):

$$\mathbf{D} = \begin{bmatrix} d_{0,0} & d_{0,1} & \cdots & d_{0,I-1} \\ d_{1,0} & d_{1,1} & \cdots & d_{1,I-1} \\ \vdots & \vdots & \ddots & \vdots \\ d_{M-1,0} & d_{M-1,1} & \cdots & d_{M-1,I-1} \end{bmatrix} \tag{3.10}$$

We transform this into the normalized one-dimensional vector $\mathbf{p}_1 = [p_0, p_1, \ldots, p_{Q1-1}]$ given by

$$p_{1q} = \frac{d_{\alpha,\beta}}{\sqrt{\sum_{i=0}^{I-1} \sum_{m=0}^{M-1} d_{m,i}^2}} \qquad q = 0, 1, 2, \ldots, Q_1 - 1 \tag{3.11}$$

where

$$\alpha = (q \mod M) \tag{3.12}$$

$$\beta = \frac{q - (q \mod M)}{M} \tag{3.13}$$

$$Q_1 = M \cdot I \tag{3.14}$$

where $q \mod M$ means the modulo operation, or the remainder of $\frac{q}{M}$.

The DFT used for creating $\mathbf{D}$ from $\boldsymbol{\mu}$ in equation (2.40), which gives us the frequency content of the target's gait pattern, might cover a frequency range that far exceeds what is necessary to record both the gait frequency and several orders of harmonics. In figure 2.7 we see that we cover a range up to 12 Hz of gait frequency, while there is nearly no signal power beyond 4-5 Hz. In addition, the range of the spectra exceeding half the sampling frequency is useless due to exceeding the Nyquist frequency where aliasing starts to occur. Further data size reduction can then be achieved by at least limiting q to e.g. $(M \cdot \frac{I}{2}) - 1$ (to remove the aliased area). For this purpose, we introduce the size I_α which denotes how many columns of the $\mathbf{D}$ matrix (starting from column 0) we want to include when creating the $\mathbf{p}_1$ vector, and replaces I in equation (3.14) which is the rewritten to:

$$Q_1 = M \cdot I_\alpha \tag{3.15}$$

A suitable value for I_α must be found empirically, but it should at least be lower than $\frac{I}{2}$, in order to remove the aliased half of the spectrum. Choosing a too

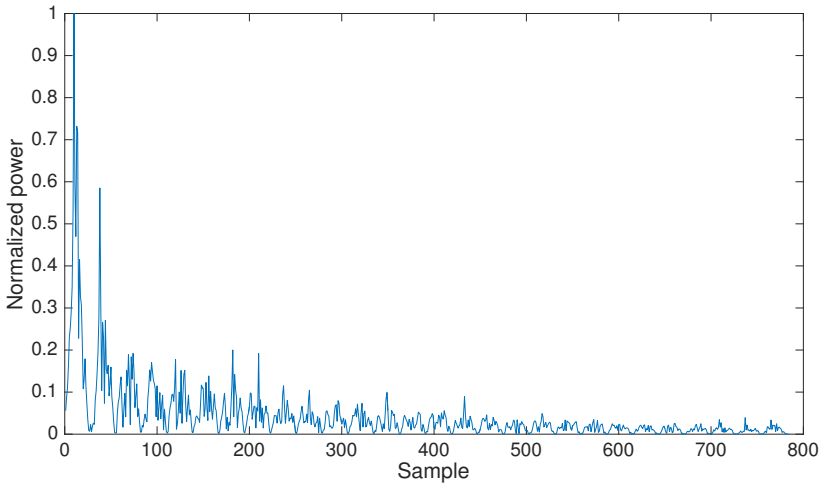

Figure 3.7.: Example of Type 1 Feature Vector

low value should also be avoided, since this will cause a risk of discarding valuable data from the radar recording.

Limiting the size Q_1 in this manner is mainly done for practical reasons, as it reduces the size and complexity of the neural network, which in turn simplifies training, making it easier to attain good results.

Figure 3.7 shows a plot of an example of a type 1 feature vector. The size $Q_1 = M \cdot I$ of the vector is governed by the window length M used for creating the range-doppler matrices $\mathbf{G}_i$ and the number of frames I used to calculate the gait-doppler matrix $\mathbf{D}$ (actually it's the length of the DFTs in both cases that govern the size of the resulting $\mathbf{p}_1$ feature vector, however in most cases this is chosen to be the same as the length of the input data for which the DFT is calculated over). In this plot, a window length M of 28 frames and a recording length I of 400 frames was used. I_α was then limited to 28 frames due to little or no useful information found beyond this point in the $\mathbf{D}$ matrix, giving a feature vector size Q_1 of 784 elements.

3.4.4. Type 2 Feature Vector

Information about a target's distribution in range is discarded when the range-doppler matrices $\mathbf{G}_i$ are summed to form the columns in the time-doppler matrix $\boldsymbol{\mu}$. In an attempt to retain at least some of this information, and use it as part

of the feature set presented to the neural network, we establish the type 2 feature vector, where some of this information is included.

In order to keep the size of the vector as compact as possible, we just want to use information from the input frame showing the greatest expansion in range that the target assumes while moving. This will require the use of some means of target tracking, or at least some means of identifying where within the frame a target exists.

We evaluate the range distribution of a target for each frame in the recorded signal. To determine if the signal to noise ratio in a frame is good enough to try identifying a target, we first calculate the average power in the frame (after removing stationary clutter as described in section 2.2.2):

$$P_{\text{avg},i} = \frac{\sum_{n=0}^{N-1} |z_{n,\frac{M}{2}}|^2}{N} \tag{3.16}$$

where $z_{n,\frac{M}{2}}$ is the center column of the framebuffer matrix with clutter removed $\mathbf{F}_i$ generated for the current frame as described in equation (2.28). Although this average power level in many cases will be higher than the actual noise level in the frame (at least with a positive signal to noise ratio) it will be helpful in determining if it is possible to detect a target at all. We determine this by simply taking the ratio between the max and average power in the frame:

$$\text{SAR} \approx \frac{\max(|z_{n,\frac{M}{2}}|^2)}{P_{\text{avg},i}} \tag{3.17}$$

where SAR means signal to average power ratio. The minimum ratio required will need to be found empirically.

We will also use this number to establish a threshold value for determining if a target is encountered:

$$P_{\text{th},i} = P_{\text{avg},i}(1 + \alpha \cdot (\text{SAR - 1})) \tag{3.18}$$

Where α is a value between 0 and 1. A suitable value for α must be found empirically. We then define n_f to be the first sample in a frame with a value that exceeds the threshold value, and n_l as the last sample to exceed the threshold value. We then only copy the part of the frame from n_f to n_l to the input vector, to remove the absolute range information, which does not provide any insight about the type of target, and also lets us remove the parts of the frame where no target

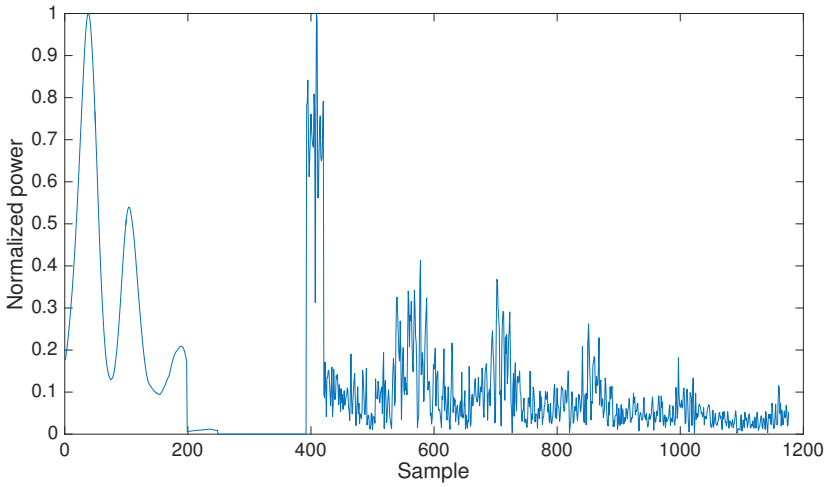

Figure 3.8.: Example of Type 2 Feature Vector, Target Class 1

is present.

If we define a type 2 feature vector to be Q_2 samples long, it is composed as follows:

$$\mathbf{p}_2[s] = \begin{cases} \mathbf{f}_L[s] & s < \frac{Q_2}{3} \\ \mathbf{p}_1[s - \frac{Q_2}{3}] & \frac{Q_2}{3} \leq s \leq Q_2 \end{cases} \tag{3.19}$$

where

$$\mathbf{f}_{L,n} = \begin{cases} z_{n-nf,\frac{M}{2},\text{large}} & n < N_e \\ 0 & N_e \leq n \end{cases} \tag{3.20}$$

$$N_e = n_{l,i} - n_{f,i} \tag{3.21}$$

Equation (3.19) shows us that 2/3 of the $\mathbf{p}_2$ feature vector is made up of vector $\mathbf{p}_1$. By using the relationship in equation (3.15), we see that the type 2 vector length Q_2 must be

$$Q_2 = \frac{3}{2} M \cdot I_\alpha \tag{3.22}$$

Figures 3.8 and 3.9 shows example plots of a type 2 feature vector, generated synthetically using the dog target model (class 1, figure 3.8) and the human target model (class 2, figure 3.9), respectively. It can be seen that the range distribution information provides useful insight; the human target model has the major scatterer

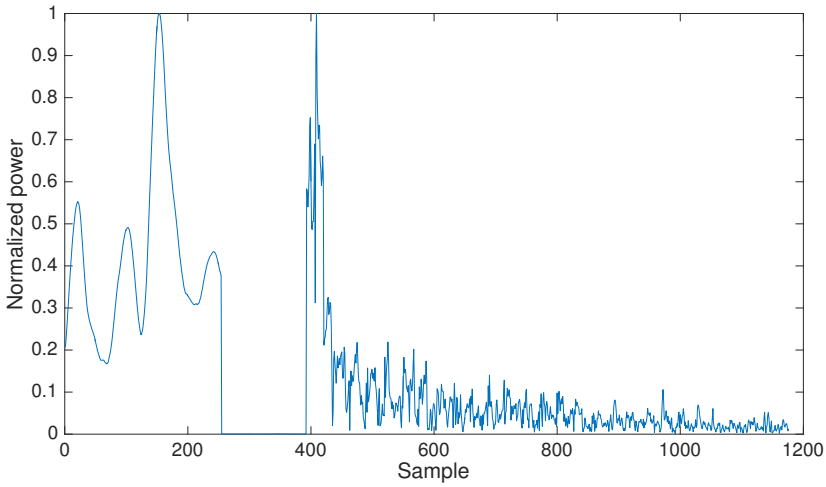

Figure 3.9.: Example of Type 2 Feature Vector, Target Class 2

(the torso) centered among all scatterers present, while the dog model has the major scatterer placed at the end. This holds true as a discerning feature between bipedal and quadrupedal animals regardless of which direction they are moving with respect to the radar. For this specific use-case, where the primary objective is to separate humans from pet animals, one could improve the generalization within one target class even further by attempting to rectify the range distribution information, ie. by ensuring that the major scatterer is always to the left of the center (or always to the right, for that matter), and reversing it if it isn't.

3.4.5. Network Layer Structure

Our network will feature two hidden layers, and as such be considered a deep architecture network. The size of the input to the network is governed by the size of the input feature vector, and will hence be equal to either Q_1 or Q_2 depending on which vector type we are using. The size of the output layer is equal to the number of target classes, which for our application is 2 (Heaton [8] points out that a classification network with only two possible outcomes could be implemented with a single output, where a '1' indicates one class and a '0' indicated the other. We will, however, use two outputs in order to make it easy to expand the code with more than two classes if desired in the future).

The two hidden layers will have a consecutively smaller size, in order to reduce

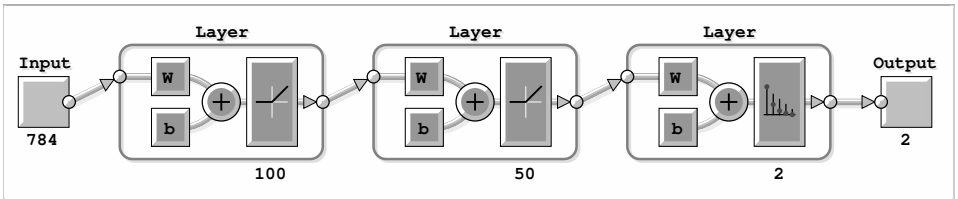

Figure 3.10.: Overview of full Neural Network

the size of the input data down to the two output nodes.

The third layer, the output layer, will be a softmax layer with the same number of nodes as the number of target classes. The softmax activation function (defined in equation (3.27)) is commonly used in the output layer of classification networks since it transforms the output to a value that gives the probability of the input data belonging to the target class associated with each node.

Figure 3.10 shows the superficial layout of the network. As in figure 3.6, the rounded corner rectangles represent a neuron layer, with their respective sizes shown below each layer (the layer sizes are subject to change).

3.4.6. Activation Functions

Activation functions were first mentioned in section 3.1.1. In this section, we will introduce some common types of activation functions used in feedforward neural networks, which we will be using in our radar target classification network.

The value y in all equations in this section refers to the weighted sum of the inputs to a neuron as seen in figure 3.1, and defined in equation (3.2). The derivatives of the activation functions are given because they in some scenarios are needed for calculating the node deltas.

Logistic Sigmoid Function

For hidden layers 1 and 2, we will be trying two different activation functions, and then evaluate which function gives the best performance in our application. The first of these is the logistic sigmoid function, which is defined as

$$\phi_l(y) = \frac{1}{1 + \exp(-y)} \tag{3.23}$$

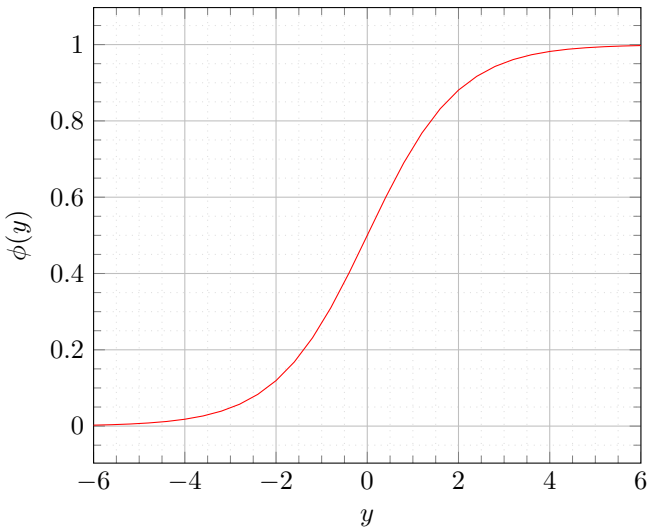

Figure 3.11.: Logistic Sigmoid Activation Function

We will also need its derivative to calculate node deltas. The derivative of the sigmoid function is given as

$$\phi_l'(y) = \phi_l(y)(1 - \phi_l(y)) \tag{3.24}$$

A plot of the logistic sigmoid function for y values between -6 and 6 is shown in figure 3.11. We see that the function tends to zero for low values of y, and saturates to 1 for high values of y.

ReLU Function

The other function we will be investigating as hidden layer activation function is the rectified linear unit, or ReLU activation function. It is defined as

$$\phi_r(y) = \max(0, y) \tag{3.25}$$

and its derivative is defined as

$$\phi_r'(y) = \begin{cases} 1 & y > o \\ 0 & y \leq 0 \end{cases} \tag{3.26}$$

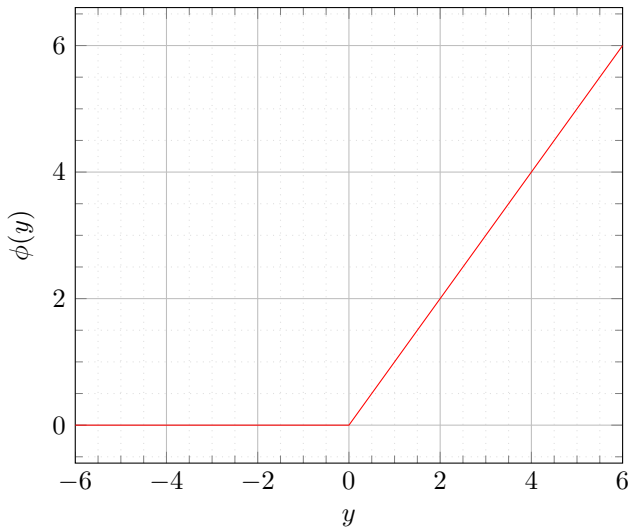

Figure 3.12.: ReLU Activation Function

(from a mathematical point of view, this isn't entirely correct, as the derivative of the ReLU function is not defined for $y = 0$. When training a neural network using the ReLU function, however, one might experience situations where the derivative for $y = 0$ is needed, for which a zero value is used).

The ReLU activation function is plotted in figure 3.12. The prime difference between the ReLU activation function and the logistic sigmoid activation function is that the ReLU function never saturates.

Softmax Function

The output layer will be using the softmax activation function, which is defined as

$$\phi_s(y_{oc}, \mathbf{y}_o) = \frac{\exp(y_{oc})}{\sum_{j=1}^{C} \exp(y_{oj})} \tag{3.27}$$

where y_{oc} is the weighted sum of the inputs to the current neuron, and $\mathbf{y}_o = y_{o1}, y_{o2}, \ldots, y_{oC}$ are the weighted sums of the inputs in all the output neurons in the same layer (y_c and $\mathbf{y}_o$ correspond to the weighted sum y shown in figure 3.1 for the respective neurons).

Figure 3.13 shows how neurons in a softmax output layer connect to the other neurons in the same layer, which shows that the output from a neuron in a softmax layer depends on the weighted input sums in all neurons in the same layer. For

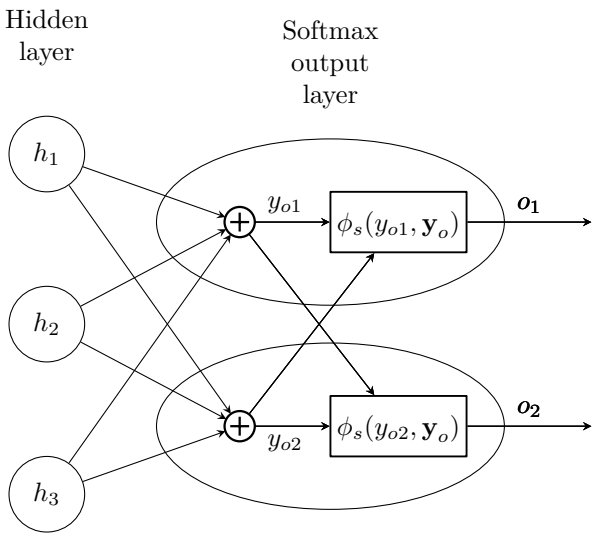

Figure 3.13.: Softmax Layer Structure

clarity, the entire inner structure of the softmax neurons is shown in the figure. The input nodes, bias neurons and their connections to the hidden and output layer are omitted.

The sum of the outputs from all neurons in a softmax layer is always 1. The most common interpretation of the output value from a softmax output neuron is that it gives the probability for the network's prediction that a set of input data belongs to the class represented by that neuron. This also ensures that the softmax function fulfills the range requirements of the input to the cross-entropy error function, as described in section 3.2.1.

We also present the derivative of the softmax function:

$$\frac{\partial \phi_s}{\partial y_c} = \phi_s(1 - \phi_s(y_{oc}, \mathbf{y}_o)) \tag{3.28}$$

4. Testing and Results

4.1. Generating synthetic Training Data

4.1.1. Micro-Doppler Simulator

A simulator for generating synthetic radar output based on a simplified model of the body of either a human or a dog was created in a previous project. For detailed explanations on how this simulator is designed, the reader is redirected to [5]. In short, the simulator takes a list of radar targets treated like point scatterers, each with their own list of parameters describing their position, motion and radar cross section. The simulator then calculates the returned radar signal from each scatterer for each point in time where a radar frame is to be generated (defined by the radar's framerate), and then combines these signals to form the radar frames.

The simulator functionally resembles a Novelda X2 radar module, and also adds receiver noise to the data. The radar signal is calculated based on the targets' radar cross section and distance (range) from the radar as described in section 2.1. The available model parameters, and how they are used for calculating the position of each scatterer and the corresponding range $R(t)$ is described in appendix A.

4.1.2. Generating a Dataset

Table 4.1 shows in pseudo-code how the synthetic training data is generated for both type 1 and type 2 feature vectors. Radar frames are generated using the simulator and the human body model developed in [5], while the calculation of the $\mathbf{G}_i$, $\boldsymbol{\mu}$ and $\mathbf{D}$ matrices was described in section 2.2. The target model to use for each iteration is chosen at random in order to improve the results from the training process, since this avoids creating false patterns in the training data, and also avoids overfitting the network to one of the target models.

A new body model for domestic cat was developed for test purposes. The reason for using a cat body model in addition to the dog body model already available was that a domestic cat was the only critter available for real life testing of the classifier. The input parameters to the process is listed in appendix B.

We will be creating test data containing both human and dog models and human and cat models, for vector types 1 and 2 (introduced in section 3.4.2), for a radar pulse repetition frequency (framerate) of both 60 Hz and 200 Hz. The ratio-

Table 4.1.: Synthetic Training Data Generation Algorithm

---

for $u = 1, 2, 3, \ldots, U$ **do**
  Choose a target class at random
  Generate random variations to body model parameters according to table B.4
  Generate temporary target model by applying the same random variations generated in the previous step to all scatterers in the target body model for the chosen target class, given in table B.1 or B.2
  for $i = 1, 2, 3, \ldots, I$ **do**
    Generate radar frame of length N
  end for
  for $i = 1, 2, 3, \ldots, I$ **do**
    Calculate range-doppler matrix $\mathbf{G}_i$
    if vector type == 2 **then**
      Calculate the extent of the target in the frame
      if extent > largest extent experienced so far **then**
        Keep current frame as frame with largest extent of the target
      end if
    end if
  end for
  Calculate the time-frequency matrix $\boldsymbol{\mu}$
  Calculate the gait-doppler matrix $\mathbf{D}$
  Truncate the $\mathbf{D}$ matrix to I_α columns
  Convert the truncated $\mathbf{D}$ matrix to a one-dimensional vector
  if vector type == 2 **then**
    Concatenate the frame with the largest extent of the target with the converted, truncated $\mathbf{D}$ matrix
    Save the resulting vector to file
  else
    Save the converted, truncated $\mathbf{D}$ matrix to file
  end if
end for

Table 4.2.: Syntetic Training Datasets

Set	Body models	Vector type	Framerate	Abbreviated
1	human, dog	1	60 Hz	T1D60
2	human, dog	1	200 Hz	T1D200
3	human, dog	2	60 Hz	T2D60
4	human, dog	2	200 Hz	T2D200
5	human, cat	1	60 Hz	T1C60
6	human, cat	1	200 Hz	T1C200
7	human, cat	2	60 Hz	T2C60
8	human, cat	2	200 Hz	T2C200

nale for using two different pulse repetition frequencies (PRFs) is that the current iteration of the Novelda X2 radar module is limited to a PRF of 60 Hz, meaning this gives us a realistic view of what is possible with the technology available today. As we experienced in [5], however, a PRF of 60 Hz will cause aliasing in the doppler spectrum for targets moving at normal walking velocities. For this reason, we also generate training data with a PRF of 200 Hz, which will avoid aliasing at normal walking velocities, to see if this will affect the performance of the neural network. Novelda expects future hardware revisions to be able to work at higher pulse repetition frequencies, and hence we want to investigate if this will also improve the classification performance.

Table 4.2 shows an overview of the eight different sets of training data that will be generated and evaluated. The abbreviations in the rightmost column will be used for referencing the different data sets in the subsequent sections.

Figures 4.1 - 4.3 shows a birds-eye two-dimensional view of the point scatterer models used in the synthetic training data generation. They are based on the parameters in tables B.1 - B.3 in Appendix B, after having moved a few hundred milliseconds according to the equations given in Appendix A. In figure 4.1, the largest scatterer represents the torso and head, the mid-size scatterers are the legs, and the smallest scatterers the arms. In figure 4.2 and 4.3 the largest scatterer represents the torso and head, the lowermost smaller scatterers are the front legs, and the uppermost smaller scatterers are the hind legs of the critter to be modeled.

In all three figures, the subjects are moving towards the radar along the y-axis.

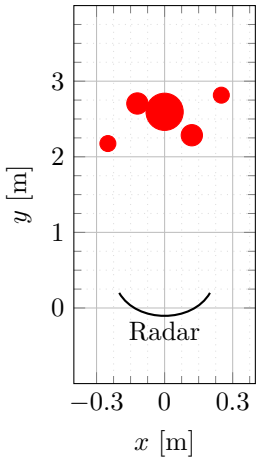

Figure 4.1.: Human Body Model

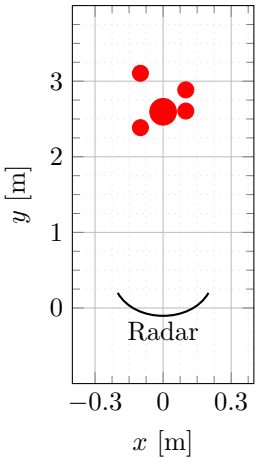

Figure 4.2.: Dog Body Model

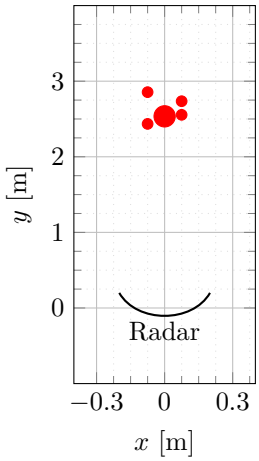

Figure 4.3.: Cat Body Model

4.2. Real Input Data

In order to test the neural network's usability in real life situations, the following recordings were made using the Novelda X2 radar module:

- Recording of adult male human walking on treadmill
- Recording of domestic cat approaching the radar indoors

These recordings were made at a framerate of 60 Hz, using the radar settings listed in table B.5. The recordings were processed to generate the $\mathbf{G}_i$ and $\mathbf{D}$ matrices, which were then used to generate a type 1 and type 2 feature vector for each recording.

Figures 4.4 and 4.5 shows the type 2 vectors generated from the recording of a cat and of a human on treadmill, respectively. By comparing them to the plots of synthetically generated type 2 vectors in figures 3.9 and 3.8, we see that the human recording is quite similar to the synthetic one, but that the cat recording contains more clutter (remembering from equation (3.19) that the first third of the samples in the type 2 vector is a portion of the radar frame where the target shows the largest distribution in range).

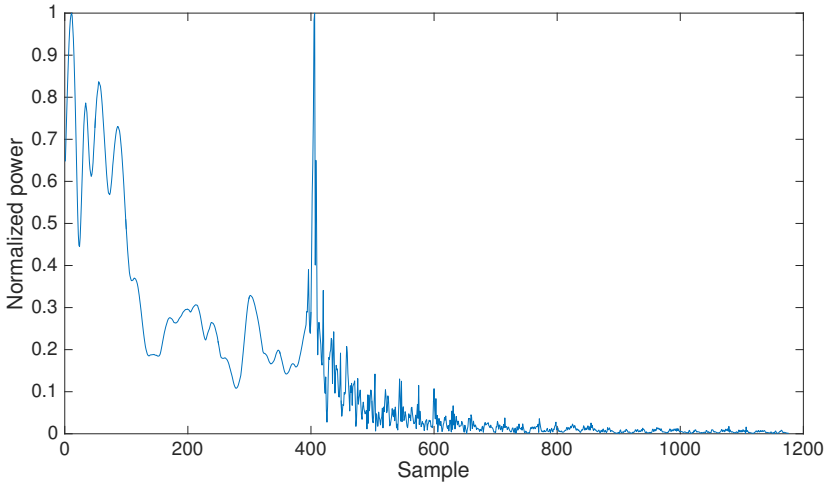

Figure 4.4.: Type 2 Feature Vector from Radar Recording of Cat

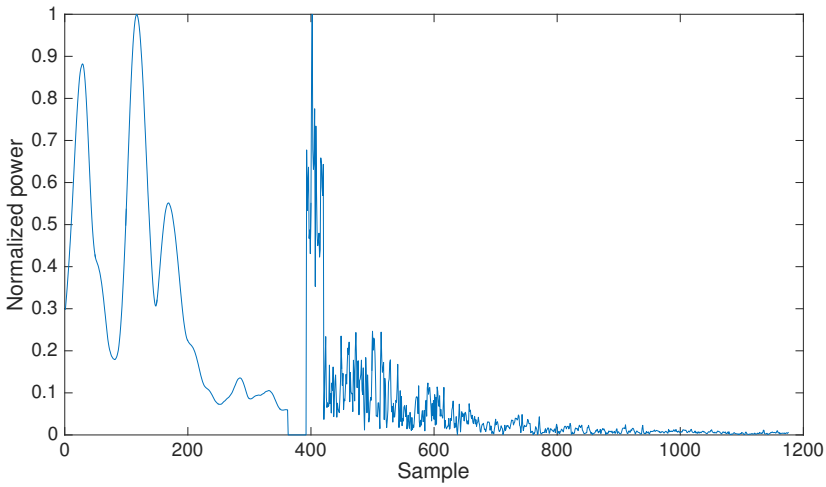

Figure 4.5.: Type 2 Feature Vector from Radar Recording of Human on Treadmill

4.3. Testing the Neural Network

4.3.1. Neural Network Training

The structure and training procedure for the target classification neural network was presented in chapter 3. Table 4.3 lists the layer sizes and number of training epochs used for training the autoencoder networks for each layer in the final, multi-layer network. We will use a somewhat different approach to training the final, multi-layer network after copying the weight values from the autoencoder networks.

The reason for this is that in order to assess the network's capabilities with the real recorded radar data, we want to find the number of training epochs where the network's performance score when processing these real data is as low as possible.

We have also experienced that the performance achieved with real data can in some events deteriorate when the number of training epochs increase beyond a certain number. One reason for this is that the network is overfitted to suit the synthetic data, which will exaggerate the differences between the synthetic data and the real data. To find the optimum number of training epochs, we train the final network 100 epochs a time, and evaluate the performance for each 100 training epochs. Continuous evaluation of the performance also enables us to detect a situation where the initial weight values makes it impossible to train the network. An elaboration of this phenomena can be found in section 5.1.1.

The network is trained with 4500 of the 5000 elements in the synthetic data set, while the last 500 elements are used for evaluation.

The procedure for training and evaluating the network for one dataset is given in pseudocode in table 4.5. Table 4.4 gives a description of the variables used in the pseudocode.

Table 4.3.: Neural Network Structure and Training Parameters

Symbol	Parameter	Value
H_1	Size of neural network hidden layer 1	100
H_2	Size of neural network hidden layer 2	50
C	Size of output layer	2
	Training epochs for hidden layer 1	500
	Training epochs for hidden layer 2	100
	Training epochs for output layer	400
	Training epochs for final network	up to 2000
	Number of times the network is trained using each dataset	10

Table 4.4.: Description of Variables in Training Algorithm Pseudocode

variable name	description
bPDataset	best performance achieved for this dataset
dSIterations	Number of times to repeat the training process for each dataset
optEpochs	The number of training epochs giving the best (lowest) performance score for this iteration
bPIteration	best performance achieved during this iteration
maxEpochs	Maximum number of training epochs to run during one iteration
cPerformance	Performance score calculated over the available real recorded data

Table 4.5.: Target Classification Network Training Algorithm

```
bPDataset = 0
for a = 1, 2, 3, ... ,dSIterations do
    Initialize and train first autoencoder using synthetic training data as input
    Generate featureSet1 by using the first autoencoder to process the synthetic training data

    Initialize and train second autoencoder using featureSet1 as input
    Generate featureSet2 by using the second autoencoder to process featureSet1

    Initialize and train softmax output layer supervised using featureSet2 as input and the labels from the synthetic training data

    Initialize final network and copy weight values from first and second autoencoder networks and the softmax output layer network
    Copy final network with new weight values to temporary network

    optEpochs = 0
    bPIteration = 0
    for b = 1, 2, 3, ... ,(maxEpochs/100) do
        Train temporary network 100 epochs
        cPerformance = performance score in current state
        if cPerformance > bPIteration then
            bPIteration = cPerformance
            optEpochs = b · 100
        end if
    end for
    Train final network optEpochs epochs
    if bPIteration > bPDataset then
        bPDataset = bPIteration
    end if
end for
```

4.3.2. Performance Measurements

Performance measurements have been made for the target classification network after training it with the datasets listed in table 4.2, with both the sigmoid function (equation (3.23)) and the ReLU function (equation (3.25)) as activation functions in the hidden layers. The network was trained ten times with each training dataset and for each activation function according to the procedure in table 4.5, each time with new initial weight values, in order to find good initial weight values. The importance of finding good initial weight values is explained in section 5.1.1. The performance measurements listed in this section are the best achieved for each dataset.

The synthetic training datasets and the real recorded data contains targets belonging to either human or pet animal target classes. For brevity, we assign numbers to the target classes as follows:

- Target class 1: Pet animal (domestic cat or dog)
- Target class 2: Human

The following performance metrics are measured

- Performance score, synthetic is calculated using the cross-entropy error function (given in equation (3.3)) over the 500 elements of the training dataset reserved for evaluation. As we recall from section 3.2.1, a lower score is better.
- Accuracy is the percentage of correctly classified targets in the 500 elements of the training dataset reserved for evaluation. It is read from the confusion plots generated for each dataset and activation function, which is found in appendix C.
- Performance score, real is calculated using the cross-entropy error function (equation (3.3)) over the available real recorded data.

Table 4.6 lists the performance metrics achieved with the ReLU activation function. Table 4.7 lists the performance metrics achieved with the logistic sigmoid activation function. The names of the datasets are referenced to table 4.2.

Table 4.6.: Performance Metrics - ReLU Network

Dataset	T1D60	T1D200	T2D60	T2D200
Performance, synthetic	0.369	0.471	0.204	0.449
Class 1 accuracy	59.0%	80.7%	83.2%	84.4%
Class 2 accuracy	63.7%	75.0%	84.0%	83.6%
Total accuracy	61.4%	77.8%	83.6%	84.0%
Performance, real	0.053	0.000	0.091	0.000

Dataset	T1C60	T1C200	T2C60	T2C200
Performance, synthetic	0.328	0.218	0.232	0.368
Class 1 accuracy	73.8%	84.4%	79.5%	88.9%
Class 2 accuracy	77.0%	84.4%	80.5%	88.7%
Total accuracy	75.4%	84.4%	80.0%	88.8%
Performance, real	0.050	0.001	0.011	0.000

Table 4.7.: Performance Metrics - Sigmoid Network

Dataset	T1D60	T1D200	T2D60	T2D200
Performance, synthetic	0.690	1.019	0.778	0.962
Class 1 accuracy	51.6%	73.8%	82.8%	88.1%
Class 2 accuracy	63.7%	71.1%	78.1%	83.2%
Total accuracy	57.8%	72.4%	80.4%	85.6%
Performance, real	0.000	0.000	0.000	0.000

Dataset	T1C60	T1C200	T2C60	T2C200
Performance, synthetic	0.460	0.636	0.730	0.753
Class 1 accuracy	65.2%	85.2%	75.8%	91.4%
Class 2 accuracy	80.9%	80.5%	78.9%	82.4%
Total accuracy	73.2%	82.8%	77.4%	86.8%
Performance, real	0.000	0.000	0.000	0.000

4.3.3. Comments to Performance Measurements

Synthetic Data Classification Accuracy

The classification accuracy for each dataset listed in table 4.6 shows that the higher pulse repetition frequency improves the accuracy for all vector types and target models. We also see that for both types of target models and for all pulse repetition frequencies, the accuracy is better with the type 2 input vector. This indicates that some merit should be given to the following hypotheses:

- A higher pulse repetition frequency (that will reduce or avoid aliasing in the doppler spectrum) will improve the ability to classify targets using neural networks (discussed in section 4.1.2)
- The range distribution information which is discarded when the range-doppler matrices $\mathbf{G}_i$ are summed to form the time-frequency matrix $\boldsymbol{\mu}$, will aid the network in correctly classifying targets and should not be (entirely) discarded (discussed in section 3.4.4)

Synthetic Data Performance Score

The performance score achieved with the synthetic data actually shows that the performance is better (lower score) for the lowest pulse repetition frequency for all but one dataset/vector type. We also see that the ReLU network consistently achieved better results than the sigmoid network. This tells us that even though the aliasing experienced at the lower pulse repetition frequency warps the micro-doppler frequency data, the patterns in the aliased data might actually be more easily recognized by a neural network than the patterns found in the alias-free data generated at higher pulse repetition frequencies.

It is a quite curious find to see that the performance score and accuracy does not seem to be closely correlated - the datasets that gave the best accuracy did not give the best performance score for neither network. This tells us that the performance score is not a perfectly accurate predictor for the network's ability to correctly classify targets - at least when deciding the target class based on the output node with the highest value alone. In section 5.2.3 we will discuss alternate approaches to interpreting the network output.

Real Data Performance Score

The number of available real recorded data is quite low, and the performance score calculated for these data cannot be regarded as a qualitative assessment of

the network's performance. It does, however, show that the network, after being trained using synthetic data, is applicable to real data as well. In the rare event that any difference between the different networks and datasets was observed, the sigmoid network was consistently better, and the networks trained using synthetic data generated at 200 Hz also produced better results - an interesting result given the fact that the real data was recorded at 60 Hz. If this tendency manifests itself when evaluating larger sets of real data, this might indicate that training the network with synthetic data generated at 200 Hz can in some events produce better results than training the network with real data recorded at 60 Hz.

5. Discussion

5.1. Comments to the Testing Procedure

5.1.1. Effect of the Network Weight Initialization

As we briefly touched in section 3.3, the weights in the neural network are initialized to random values, and as Heaton mentions, research has shown that the way the random initial values for the weights are generated will affect the results of the neural network training [8].

Recall from section 3.4.6 that the output from a softmax output neuron corresponds to the probability (as predicted by the network) that a certain input feature set belongs to the class associated with that neuron. Let the output from the neuron associated with a target's correct class (when using labeled training data we know the correct "answer" to the classification task) be known as the neural network's confidence, which shows the certainty of which the network believes that the feature set belongs to the correct class. The confidence ranges from 0 to 1 (the confidence is directly related to the performance score, but is more easily interpreted in this setting).

When training the neural networks with the synthetic training data and then testing the confidence using the real recorded data, it was observed that the confidence could develop quite differently from one test run to another as the number of training epochs increased. This is due to how well the initial weight values generated are suitable for the classification problem at hand. Figure 5.1 shows an example of a scenario where the initial values of the weights turned out to be quite good for achieving good confidence scores. We observe that the confidence keeps increasing towards the end of the plot, meaning we could probably increase the confidence score even more by increasing the number of training epochs.

Figure 5.2, however, shows a scenario where the initial weight values turned out to be quite poor for this application. The confidence for the class 1 target drops rapidly and continues to shrink towards the end of the plot. If such a situation is experienced, one should discard the current weight values and restart training with new initial weight values.

Figure 5.1.: Development of Confidence with good initial Weights

Figure 5.2.: Development of Confidence with poor initial Weights

5.1.2. Advantages and Disadvantages of using synthetic Training Data

It should come as no surprise that when using synthetically modeled data to simulate the behavior of real, physical events, the efficacy of these simulations are limited by the degree of accuracy of which the model correlates with reality. The extensive use of synthetic data for both training and evaluation of the solution developed in this project hence limits the certainty of which the solution can be declared as suitable for the problem it was designed to solve.

With that being said, the results presented in section 4.3.2 clearly demonstrates that using synthetic data for training can be a viable path for achieving acceptable performance even when preparing the network to process real data. This will in turn significantly reduce the amount of real data that need to be gathered for training, which can be a costly, time-consuming task. If we are able to generate synthetic data with a quality sufficient for training purposes, we could utilize them for the following training approaches:

1. Train the network using synthetic data, then evaluate the network using real data
2. Use synthetic data for initial training, then re-train using real data

Searching through the available bibliography has not revealed any general recommendations for the amount of training data elements required for successful training of a neural network - not even as a rule of thumb. The general response to this question when asked at different internet bulletin boards is that the answer in any event depends on the problem to be solved - and that the training dataset should

to an as large extent as possible cover all expected variations of the input data, which in most cases will be an exercise solved by experience.

In our testing, we eventually found it sufficient to have 5000 training data elements available, of which 4500 were used for training and 500 for evaluation. If we could train the network successfully for operating on real data using 4500 synthetic training data elements and 500 real training data elements, this would reduce the required amount of data that has to be collected by a factor of 10 compared to having to collect 5000 real training data elements.

5.1.3. PRF Limitations and Aliasing

The current iteration of Novelda radar modules available at the time of writing is limited to a maximum pulse repetition frequency of 60 Hz. In order to understand how this limits the feature extraction process and hence the accuracy of the data input to the target classification process, we recollect from section 2.2.3 that the DFT that forms the range-doppler matrix $\mathbf{G}_i$ is calculated over the slow-time scale, which means that the pulse repetition frequency f_p effectively becomes the sampling frequency of the motion pattern. Section 2.1.8 showed that the frequency content of the signal, more specifically the frame-to-frame phase difference of the signal depends the target's velocity.

For the radar to be able to accurately measure a target's velocity (which for a non-rigid body with independently oscillating parts will be higher than just the average locomotion velocity), the target cannot move more than one half wavelength radially from one frame to the next. In radar terminology, the velocity where this criterion is exceeded is known as the *first blind speed* [3], and is calculated as

$$v_1 = \frac{\lambda f_p}{2} \tag{5.1}$$

where f_p is the pulse repetition frequency and λ the carrier wavelength.

With normal walking velocities residing in the range from 0.8 to 1.5 m/s, and individual body parts with an oscillating motion exceeding this in the worst-case scenarios, we find that the pulse repetion frequency of 60 Hz along with the carrier frequency of 6.8 GHz will cause aliasing when the velocity exceeds 1.32 m/s.

In section 4.3.2 we found that using a higher pulse repetition frequency of 200 Hz when generating the synthetic training data increased the accuracy, and we also experienced minor improvements in the performance score with real recorded data. For these reasons, we also expect that newer, improved radar modules able

to work at higher pulse repetition frequencies will experience better accuracy. For the reference, a pulse repetition frequency of 200 Hz gives a first blind speed of 4.41 m/s.

5.1.4. No Clutter in synthetic Data

Removing the effects of unwanted objects in the radar signal, known as clutter, is an important step in the signal processing that takes place in a radar system designed to detect moving targets. The simulator developed for the project does not support any method for simulating distributed clutter types, such as surface or volume clutter, which both appear frequently in real world scenarios; the only type of clutter that can be added is in the form of stationary point scatterers.

In the testing performed using real recorded radar data, it was experienced that the clutter removal approach presented in section 2.2.2 did a sufficiently good job removing the clutter present in the recordings, to a degree where the level of clutter in the real recordings were comparable to the synthetic data (where no clutter was added).

5.2. Further Work

5.2.1. Gathering real Training Data

Given our preceding comments about the limited amount of available real data, it should come as no surprise that the collection of real radar data for training and evaluation purposes should be a prioritized task in the continuation of this project. Even though the findings discussed in section 5.1.2 allows us to get away with collecting less than the 5000 elements of data used for training during synthetic testing, a certain number of real data recordings still needs to be collected at least in order to verify the systems applicability in real world situations.

5.2.2. Implementation and live Processing

All testing so far has been performed as post-processing, ie. the data has been generated/recorded first, then analyzed batch-wise afterwards. With respect to the intended usage in presence-detecting alarm systems, this approach will not be overly useful, since one generally wants the alarm system to respond to events as they occur.

In a live alarm system, the radar will be recording and analysing data continuously. One suggested approach to solve the continuous processing task could be as follows:

- every 1 second, the data recorded during the previous 2 seconds is analyzed. The time span of 2 seconds has been found to be a good compromise between gathering enough data to cover at least one gait cycle at even very slow paces, and not including too much noise in the data. The half period overlap ensures that no events are missed.
- The process will first remove the stationary clutter from the signal, and then calculate the remaining power in the recorded signal. If the signal power with clutter removed is below a certain threshold level, no moving subject is present, and no further processing is required.
- If the signal power exceed the threshold level, generate a feature vector and have the neural network process it, and trigger the alarm if the neural network signals a human intruder.

The network will be trained offline, which means that one uses a comparatively powerful computer to train the network in order to find the best weight values for the network, and then copy these weight values to the network implemented on the processing unit attached to the radar. The reason for this is that the radar's attached processing unit will most probably be a microcontroller unit (MCU), which is both a cost-, energy-, and space-efficient choice, but does not possess the required computational capabilities to perform the training process. In a mass production scenario it also makes sense to have the software, including the network prepared in advance to be able to install the software quickly to the microcontroller's flash memory.

5.2.3. Create Decision Limits for separating Classes

So far we have treated the output from the network as a simple search for the highest output value to determine whether a target belongs to the human or pet animal class. This means that if the output from the softmax output layer at a node is above or equal to 0.5 (for a system with two outputs), the network decides that the target belongs to the class associated with that node.

In the alarm system application, knowing for certain which class the target belongs to is not necessarily the most important piece of information. Rather, it is more interesting to know whether a target is human or not human. While this might

sound like two different ways of saying the exact same thing, we should acknowledge the fact that the consequences of making the wrong decision are larger in one event than the other. If the alarm mistakenly triggers when passed by a pet animal, it forces the alarm company to investigate the protected premises unnecessarily, which costs money, but no property is damaged or stolen. If, however, the alarm mistakenly decides not to trigger when a human intruder is present, the intruder is free to execute criminal actions without worrying about (immediate) prosecution.

A false negative is hence much worse than a false positive, and the degree of certainty required for the alarm not to trigger when it thinks that a moving target is a pet animal should be larger than that required to trigger the alarm when it thinks that an intruder is present. This can be achieved by setting the decision limit between the classes at a different level than above or below 0.5, e.g. one could require that the output node associated with the pet animal class (class 1) should output a value larger than 0.8 in order to not trigger when a moving target is present.

Experiences from real world testing will reveal if such biased decision limits are required to minimize the frequency of false negative alarms.

5.2.4. Optimizing Feature Extraction for Machine Learning

Currently, the feature extraction process is focused around producing metrics that relate to physical properties of a target's motion pattern. When using machine learning to classify targets, this might neither be the best approach nor may it be necessary, since the machine does not even know how to interpret such metrics in the first place. Perhaps there are other ways to process the incoming radar frames in a way that exaggerates the differences between humans and pet animals in a way that makes it easier for a neural network to separate them - such possibilities can be investigated as a part of future improvements.

One suggestion has been to simply feed the raw radar frames unprocessed to the network, and let the network figure out the feature extraction process (including clutter removal) completely by itself. This will though create an issue with the size of the input data, since each frame contains 768 values (or more if one wants to increase the area covered by the radar), and one has to include more than one frame to retain the information about the target's motion pattern.

5.2.5. Try different Network and Input Data Configurations

There are many ways to configure the neural network that we haven't had the time to evaluate. The choices we have made has largely been based on the recommendations in the cited works. There might, however, exist other configurations that will be better suited for our specific problem. Parameters that can be changed in this respect include (but are not limited to:

- Network layer size
- Activation functions
- Number of layers
- Learning rate and momentum
- Error function
- Training function
- Use sparsity and regularization (these topics are not covered in this text)

6. Conclusion

We have shown through this project that using artificial neural networks is an applicable solution for classifying radar targets. A combination of the radar target's micro-doppler signature and the raw, unprocessed radar frames proved to be suitable as input data for classifying the targets, and yielded an accuracy of 89% when classifying 500 synthetically generated data elements.

We have shown that synthetically generated data can be used for training of a network when a sufficient number of collected real data is not available. This is an important find, since the process of gathering training data can be a comprehensive and time-consuming task, and will have a substantial impact on the cost and duration of the implementation process of a machine-learning system for radar target classification.

At the same time, the extensive use of synthetic data and limited testing with real data makes it impossible to say something conclusive about the solution's applicability and performance in real-world conditions. Thorough testing with real data is required to address this limitation.

We conclude that the path we have chosen for the task of discriminating humans from pet animals using Novelda radar technology has yielded a promising solution.

Bibliography

[1] Chen, Victor C (2011): *The Micro-Doppler Effect in Radar*. Artech House.

[2] Fossum, Thor Øyvind (2015): *Exploration of Micro-Doppler Signatures Associated with Humans and Dogs using UWB Radar*. Master's thesis written at NTNU, Department of Electronics and Telecommunications.

[3] Skolnik, Merill I. (2002): *Introduction to Radar Systems*. 3rd edition. McGraw-Hill Education.

[4] Novelda X2 module datasheet

[5] Langen, Helge (2015): *Simulation of Micro-Doppler Signatures in Ultra-Wideband Radar*. Specialization project at NTNU, Department of Electronics and Telecommunications.

[6] Y. Bengio (2009): *Learning Deep Architectures for AI*, FNT in Machine Learning, vol. 2, no. 1, pp. 1–127.

[7] Bell, Jason (2015): *Machine Learning: Hands-On for Developers and Technical Professionals*. John Wiley & Sons.

[8] Heaton, Jeff (2015): *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks*. Heaton Research, Inc.p

Appendix A.

Target Model Parameters, Position and Range calculation

Let $[x(t), y(t), z(t)]$ describe the time-varying position of a radar target in a coordinate system where the radar is placed at the origin, as seen in figure A.1. The components will vary according to the following equations:

$$x(t) = x_0 + v_{x0}t + \sum_{k=1}^{3} A_{x2k-1} \sin((2k-1)\omega_x t + \phi_x) \tag{A.1}$$

$$y(t) = y_0 + v_{y0}t + \sum_{k=1}^{3} A_{y2k-1} \sin((2k-1)\omega_y t + \phi_y) \tag{A.2}$$

$$z(t) = z_0 + v_{z0}t + \sum_{k=1}^{3} A_{z2k-1} \sin((2k-1)\omega_z t + \phi_z) \tag{A.3}$$

where the parameters represent

- Initial position (x_0, y_0, z_0)
- Velocity (v_{x0}, v_{y0}, v_{z0})
- Harmonic frequency $(\omega_x, \omega_y, \omega_z)$
- Harmonic phase (ϕ_x, ϕ_y, ϕ_z)
- Fundamental amplitude (A_{x1}, A_{y1}, A_{z1})
- 3rd harm. amplitude (A_{x3}, A_{y3}, A_{z3})
- 5th harm. amplitude (A_{x5}, A_{y5}, A_{z5})

Still refering to figure A.1, the range from the radar to the target $R(t)$ is given by

$$R(t) = \sqrt{x(t)^2 + y(t)^2 + z(t)^2} \tag{A.4}$$

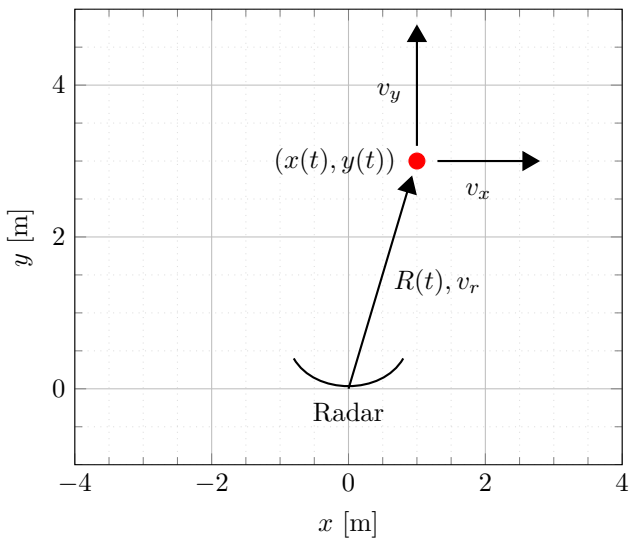

Figure A.1.: Coordinate System

Appendix B.

Synthetic Training Data Parameters

This chapter presents the parameters used when generating synthetic training data with the simulator described in [5].

Tables B.1, B.2 and B.3 lists the input parameters for the radar target model of a human, a dog and a cat, respectively. Table B.4 lists the standard deviation of the normally distributed random variation applied to the target models for each element in the training data set. A brief explanation of each parameter is given in appendix A. For a more in-depth description of the parameters and how they affect the resulting radar signal, please refer to [5].

Parameters not listed in tables B.1, B.2 and B.3 are set to zero. For parameters not listed in table B.4, no random variation is applied.

Table B.5 lists the radar settings used in synthetic training data generation. These parameters are explained in section 2.1.

Table B.1.: Human Body Model Scatterer Parameters used in synthetic Data Generation

Scatterer (Unit)	x_0 m	y_0 m	z_0 m	v_{y0} m/s	ω_y Hz	ϕ_y rad	$A_{y,1}$ m	$A_{y,3}$ m	σ m^2
Torso	0	2.5	0	-1	2	$\pi/2$	.1	0	10000
Left arm	.25	2.5	0	-1	1	$\pi/2$	.3	.02	500
Right arm	-.25	2.5	0	-1	1	$-\pi/2$	.3	.02	500
Left leg	.12	2.5	-0.6	-1	1	$-\pi/2$	.2	.01	1000
Right leg	-.12	2.5	-0.6	-1	1	$\pi/2$	.2	.01	1000

Table B.2.: Dog Body Model Scatterer Parameters used in synthetic Data Generation

Scatterer (Unit)	x_0 m	y_0 m	z_0 m	v_{y0} m/s	ω_y Hz	ϕ_y rad	$A_{y,1}$ m	$A_{y,3}$ m	σ m^2
Torso	0	2.5	0	-1	2	$\pi/2$	.1	0	5000
Left front leg	.10	2.5	0	-1	1	$\pi/2$	.1	.01	500
Right front leg	-.10	2.5	0	-1	1	$-\pi/2$	.1	.01	500
Left hind leg	.10	3.0	0	-1	1	$-\pi/2$	.1	.01	500
Right hind leg	-.10	3.0	0	-1	1	$\pi/2$	.1	.01	500

Table B.3.: Cat Body Model Scatterer Parameters used in synthetic Data Generation

Scatterer (Unit)	x_0 m	y_0 m	z_0 m	v_{y0} m/s	ω_y Hz	ϕ_y rad	$A_{y,1}$ m	$A_{y,3}$ m	σ m^2
Torso	0	2.5	0	-1	2.6	$\pi/2$	.04	0	2000
Left front leg	.075	2.5	0	-1	1.3	$\pi/2$	.05	.01	200
Right front leg	-.075	2.5	0	-1	1.3	$-\pi/2$	.05	.01	200
Left hind leg	.075	2.8	0	-1	1.3	$-\pi/2$	.05	.01	200
Right hind leg	-.075	2.8	0	-1	1.3	$\pi/2$	.05	.01	200

Table B.4.: Standard Deviations for the random Variations applied to each Body Model Parameter in synthetic Data Generation

Parameter (Unit)	x_0 m	y_0 m	v_{y0} m/s	ω_y Hz	ϕ_y rad	$A_{y,1}$ m	σ %
Standard deviation	0.75	1	0.5	0.25	$\pi/4$	0.05	30

Table B.5.: Radar Parameters used in synthetic Data Generation

Symbol	Parameter	Value
f_c	Carrier frequency	6.8 GHz
λ	Carrier wavelength	0.044 m
P_t	Transmitter power	0 dBm
T_p	Pulse duration	1 ns
σ_p^2	Pulse filter variance	0.2 ns
f_s	ADC Frequency	41 GHz
t_s	Frame offset	4 ns
f_p	Framerate/PRF	60 Hz 200 Hz
N	Samples per frame	768
F_n	Receiver noise figure	13 dB
G_{Rx}	Receiver LNA gain	10 dB
G_t	Tx antenna gain	4 dB
G_r	Rx antenna gain	4 dB

Table B.6.: Feature Set, Feature Extraction and Classification Parameters used in synthetic Data Generation

Symbol	Parameter	Value
T_{sim}	Simulation time	2 seconds
I	Number of frames	120 400
U	Number of feature sets (and labels) in training data	5000
C	Number of target classes/size of label vector	2
Q_1	type 1 feature vector size	784
Q_2	type 2 feature vector size	1176
M	Window length	28
$P_{\mathrm{th},i}$	Threshold level between signal and noise	$P_{avg,i} + 0.2\cdot$ SAR
I_α	Number of columns in truncated $\mathbf{D}$ matrix	28

Appendix C.

Confusion Plots for Network Evaluation

The names of the datasets referenced in the figure captions refers to those listed in table 4.2. Confusion plot metrics are explained in table 3.1.

C.1. Confusion Plots for ReLU Network

The confusion plots in this section was generated when using the rectified linear unit (ReLU) activation function for the hidden layers, as defined in equation (3.25).

Figures C.1 - C.4 shows the confusion plots generated with the dog body model and the human body model present in the training data, while figures C.5 - C.8 shows confusion plots with the domestic cat body model and the human body model present in the training data.

C.2. Confusion Plots for Sigmoid Network

The confusion plots in this section was generated when using the logistic sigmoid activation function for the hidden layers, as defined in equation (3.23).

Figures C.9 - C.12 shows the confusion plots generated with the dog body model and the human body model present in the training data, while figures C.13 - C.16 shows confusion plots with the domestic cat body model and the human body model present in the training data.

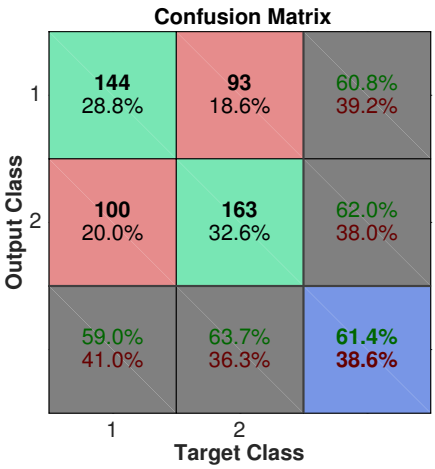

Figure C.1.: Confusion Plot, ReLU, T1D60

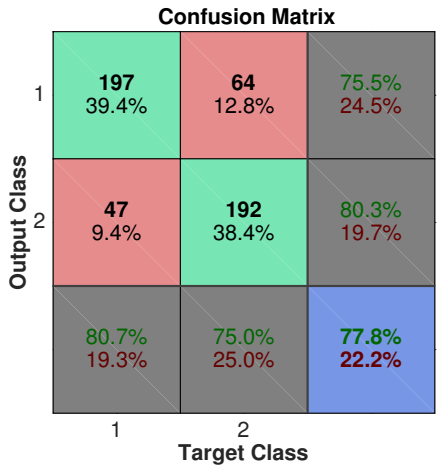

Figure C.2.: Confusion Plot, ReLU, T1D200

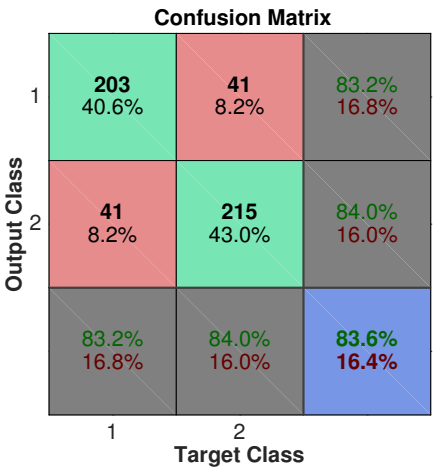

Figure C.3.: Confusion Plot, ReLU, T2D60

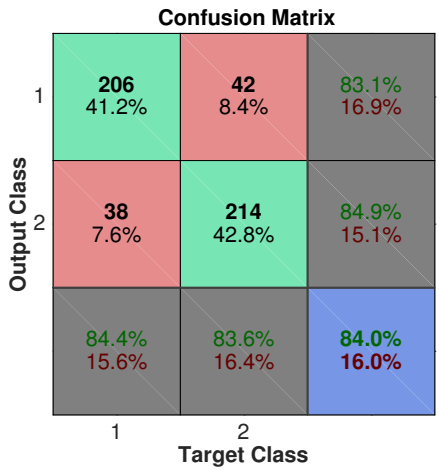

Figure C.4.: Confusion Plot, ReLU, T2D200

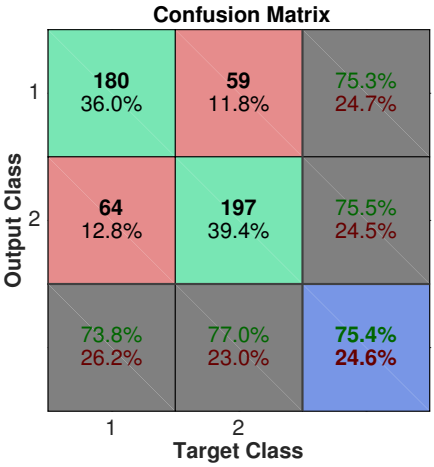

Figure C.5.: Confusion Plot, ReLU, T1C60

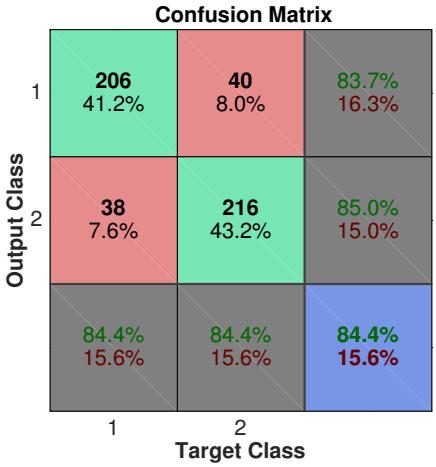

Figure C.6.: Confusion Plot, ReLU, T1C200

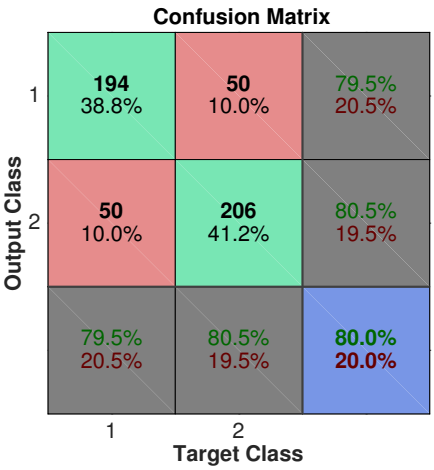

Figure C.7.: Confusion Plot, ReLU, T2C60

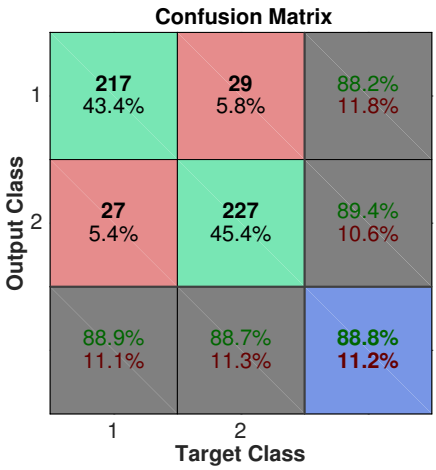

Figure C.8.: Confusion Plot, ReLU, T2C200

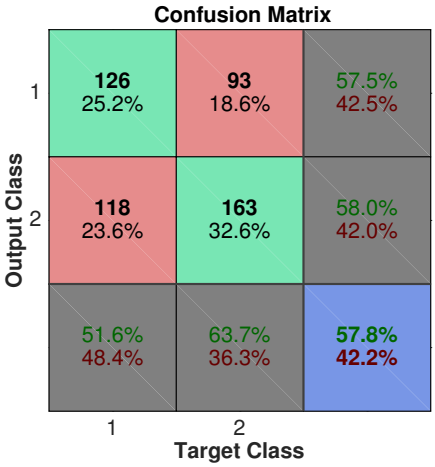

Figure C.9.: Confusion Plot, Sigmoid, T1D60

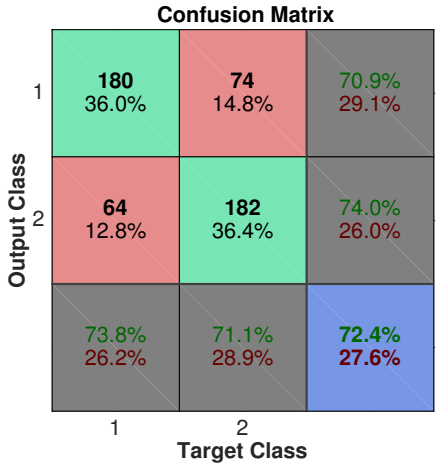

Figure C.10.: Confusion Plot, Sigmoid, T1D200

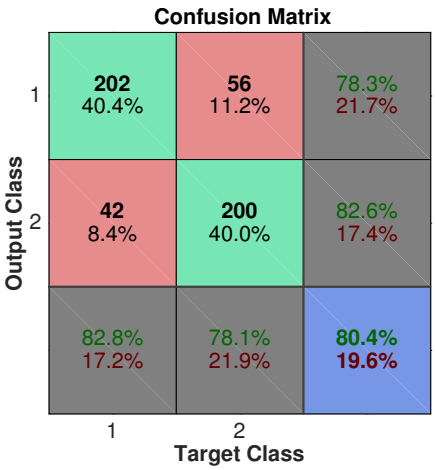

Figure C.11.: Confusion Plot, Sigmoid, T2D60

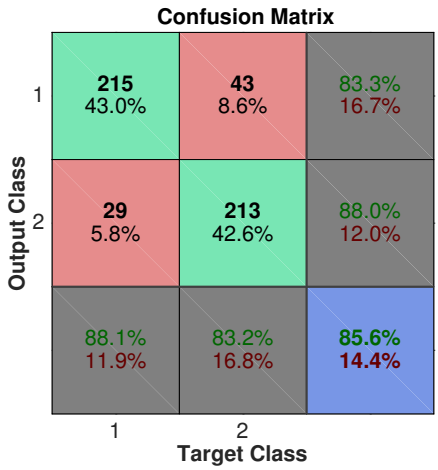

Figure C.12.: Confusion Plot, Sigmoid, T2D200

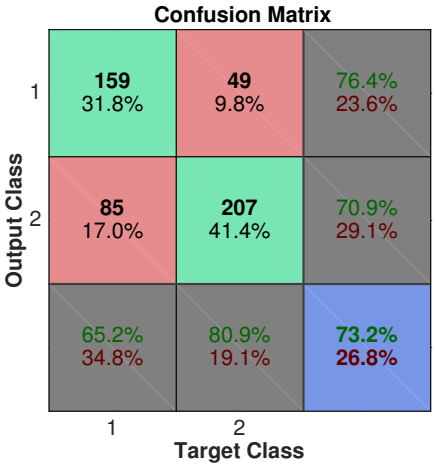

Figure C.13.: Confusion Plot, Sigmoid, T1C60

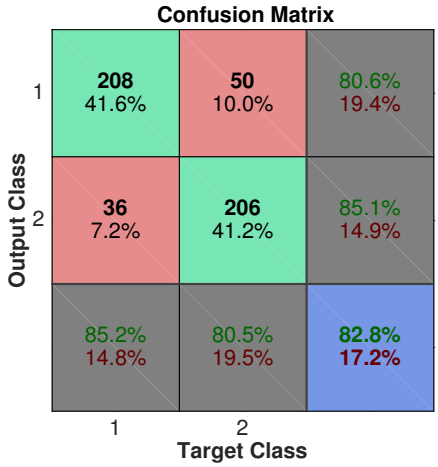

Figure C.14.: Confusion Plot, Sigmoid, T1C200

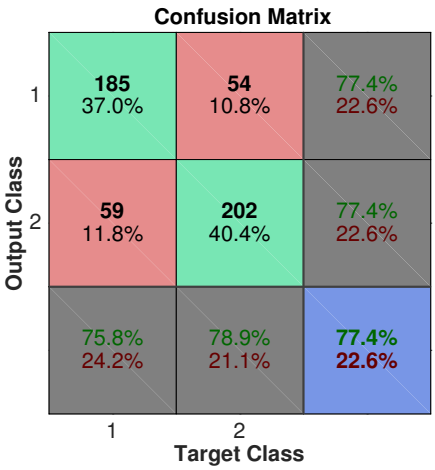

Figure C.15.: Confusion Plot, Sigmoid, T2C60

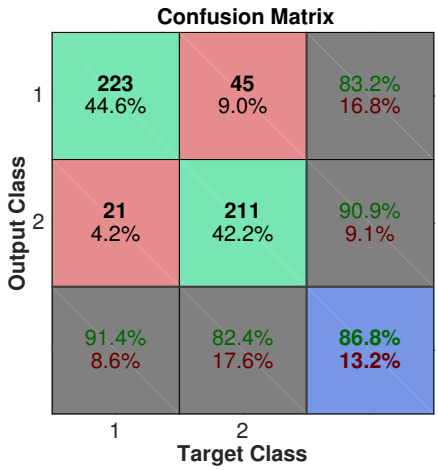

Figure C.16.: Confusion Plot, Sigmoid, T2C200