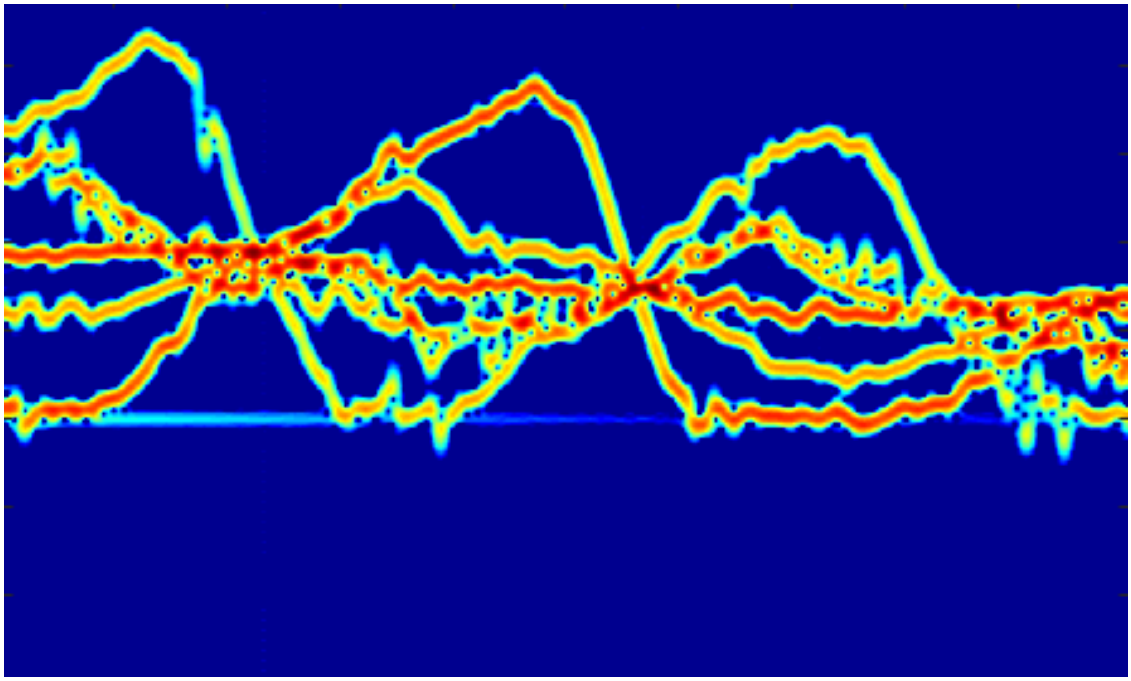
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Chalmers University of Technology

## Human activity classification using simulated micro-Dopplers and time-frequency analysis in conjunction with machine learning algorithms: a comparative study for automotive use

Master's thesis in Communication Engineering

By: Fredrik Axelsson & Pavel Gueorguiev

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017:08

# Human activity classification using simulated micro-Dopplers and time-frequency analysis in conjunction with machine learning algorithms: a comparative study for automotive use

Fredrik Axelsson & Pavel Gueorguiev


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering

Chalmers University of Technology
Gothenburg, Sweden 2017

Human activity classification using simulated micro-Dopplers and time-frequency analysis in conjunction with machine learning algorithms: a comparative study for automotive use

Fredrik Axelsson & Pavel Gueorguiev

© NAME FAMILYNAME, 2016.

Supervisor: Amer Nezirovic, Volvo Cars Company
Examiner: Lars Hammarstrand, Signals and Systems

Master's Thesis 2016:08
Department of Electrical Engineering

Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Smoothed Pseudo Wigner-Ville Distribution highlighting micro-Dopplers generated by a walking human.

Gothenburg, Sweden 2016

Human activity classification using simulated micro-Dopplers and time-frequency analysis in conjunction with machine learning algorithms: a comparative study for automotive use

Fredrik Axelsson & Pavel Gueorguiev
Department of Signals and Systems
Chalmers University of Technology

# Abstract

As vehicle automation becomes more common and encompasses more vehicle functionality early detection of vulnerable road users (VRUs) becomes a greater concern. One part of addressing this is to use on-board radars to detect micro-Doppler ($\mu$-D) signatures associated with VRUs and classify them to give early warning. With time-frequency analysis, $\mu$-D signatures can be extracted and in conjunction with machine learning algorithms (MLAs) they can also be classified. In this thesis work, done at Volvo Cars Company (VCC), different combinations of algorithms for time-frequency analysis and machine learning are compared to determine what is suitable in an automotive context.

A $\mu$-D radar return was simulated using data from the Motion Capture database available at Carnegie Mellon University's graphics lab. Three different human activities were available for classification with subjects walking, running and boxing. The $\mu$-D signatures were generated using four time-frequency analysis algorithms: Short-Time Fourier Transform (STFT), Continuous Wavelet Transform (CWT), Smoothed Pseudo Wigner-Ville Distribution (SPWVD) and Empirical Mode Decomposition (EMD). The signatures extracted using STFT, CWT and SPWVD were in image format and were classified using two machine learning algorithms: Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN).

The algorithms were applied to both noisy and noiseless data. The accuracy of classification on noisy data varied from 69.23% to 100% depending on what combination of algorithms was used. CWT in combination with an ANN resulted in classification performing perfectly, likely because the data set was too small for there to be any errors. STFT and PSWVD in conjunction with CNNs were found to have very similar performance to each other. EMD coupled with CNN proved itself to be a promising with a classification accuracy of 97.50% on noisy data.

The PSWVD algorithm was found to be unsuitable for on-board vehicular use due to extensive computation times without any major performance gain. Other algorithms performed within more reasonable time frames but only the EMD was fast enough to work in a live traffic situation with an average of 0.05 seconds. With this speed a complete classification was possible, using a CNN, in less than 0.075 seconds.

Keywords: micro-Doppler, STFT, EMD, Time-Frequency Analysis, CWT, SPWVD, ANN, CNN, Automotive, Vulnerable Road Users.

# Acknowledgements

I would like to thank my sister and parents for their help and support in this endeavour. I am also grateful to our examiner Lars Hammarstrand for his advice during the tough times of this thesis.

Fredrik Axelsson, Gothenburg, August 2017

Velim ad gratias ago meus frater, mater et pater. Amicis meis epularer. Lars advisor meas. Et regina in villam. Alea iacta est.

Pavel Gueorguiev, Gothenburg, August 2017

# Contents

# List of Figures

# List of Tables

# 1
# Introduction

Early detection and reliable classification of humans is key in avoiding collisions between motor vehicles and exposed humans. Experiments with radars within the automotive industry started as early as the late 50's but have since matured significantly. As radar technology evolved different frequency ranges and radar types became relevant for automotive use. Collision avoidance has been a powerful driver in the development of such technology and with the evolution of Monolithic Microwave Integrated Circuits and micro-controller processing power automotive radars have become much more commercially viable. Cheap modern sensors and electronics combined with modern communication technology has resulted in vehicle manufacturers including several sensors in their products. Some of these sensors are radar units.

As car accidents are a source of fatalities and serious injuries across the globe widespread use of such technology could save many lives each year. One example of successful application is when the Greyhound Lines bus company 1993 installed 24 GHz radars on 1600 buses resulting in a 21% reduction in accidents during their first year of use [1]. A radar detects movements of its targets in terms of Doppler shifts. Pedestrians, bicyclists and strollers are all examples of *Vulnerable Road Users* (VRUs) whose movements give rise to so-called micro-Doppler ($\mu$D) signatures. These ($\mu$D) signatures can be used to not only detect but classify VRUs. Quick and reliable classification would be a powerful tool for accident avoidance.

Since the original observation of $\mu$D signatures by Victor C. Chen in his defining work [2], the use of $\mu$D signatures has been evaluated for a plethora of applications. In addition to the aforementioned automotive case applications include distinguishing civilians from combatants, reconnaissance and security. Separating different $\mu$D signatures into their different frequency components has sparked interest in the signal processing community in an attempt to find good algorithms for the problem. The process of generating these time-frequency maps of ($\mu$D) signatures is called *Doppler Extraction* (DE).

Time-frequency analysis algorithms such as *Short-Time Fourier Transform* (STFT), *Continuous Wavelet Transform* (CWT) and *Wigner-Ville Distribution* (WVD) have been used for DE. Certain algorithms excel at different things such as the STFT having been shown to do well on parameter estimation [3] while for example CWT has been shown to do well on classification. WVD has performed very well for classification at the cost of introducing additional complexity. This complexity is caused by frequential cross-terms complicating the analysis. Added complexity may be acceptable depending on the application but in an automotive context computation power

is a limited resource and may require a simpler approach. DE has also been done using Independent Component Analysis (ICA) and Principal Component Analysis (PCA) [4] where simulated data with $\mu$D signatures was generated, extracted and used for classification. Lastly the *Empirical Mode Decomposition* (EMD) has been shown to be a potentially powerful method to extract the time-frequency information in a signal by separating it into its frequency components [5].

Classification also requires the use of *Machine Learning* (ML) algorithms where different algorithms may have different performance. This is an open topic for research but examples of algorithms put forward are *Support Vector Machines* (SVMs), *Dynamic Time Warping* (DTW)[7] and *k-Nearest Neighbour* (k-NN)[4], *Artificial Neural Networks* (ANNs) and *Convolutional Neural Networks* (CNNs).

An ANN was tested in combination with spectrograms generated by an STFT as DE algorithm where 12 human subjects performed seven different activities to be classified. Performance varied between 82.7-87.8%[8] depending on validation scenario. A SVM using six features have been shown to successfully classify different human activities in a controlled environment with an accuracy of over 90%[9] in a similar study using the same data set as mentioned above. Lastly, the author of the above mentioned papers did a study using a CNN with spectrograms as input. It was able to distinguish humans from animals and cars and successfully classified with 97.6% accuracy[10].

There are plenty of algorithms to choose from both in regards to DE and ML. There is no obvious answer to what combinations perform best in an automotive context. This is due to the trade-offs between complexity, observation duration and size of the data set needed for training and testing. Both choice of DE algorithm and ML algorithm impact results meaning there are numerous possible combinations.

## 1.1 Thesis Objectives

The main goal of this thesis is to evaluate combinations of DE and ML algorithms in order to determine their performance regarding VRU classification. As some DE and ML algorithm combinations may perform better than others in the automotive context this could be useful information for future work on VRU classification.

The objectives of this work can be divided into the following:

- Simulate radar response given measured motion-capture data.
- Apply a selection of DE algorithms, namely STFT, CWT, WVD and EMD, on the data set.
- Use the output from the DE algorithms as input to the selected ML algorithms: ANN and CNN.
- Evaluate and compare performance in terms of classification accuracy, computation cost and observation time.

## 1.2 Related Work

The data set used in this thesis originates from Carnegie Mellon University. It is called the CMU Graphics Lab Motion Capture Database [11] and is available freely on their website. Several subjects have been recorded performing different activities while wearing 41 markers saved as movements in space over time as the Cartesian coordinates of the markers. How this data is processed and used is discussed in-depth in Section 3.

A crucial part of the thesis builds on the simulation of a radar response given the Cartesian coordinates over time. A model for radar response simulation was proposed in a paper[12] using the Boulic model[13] as input data. The same simulation model was also shown to work with a Kinect sensor generating the data set[14]. The same study also compares CMU Motion Capture (MOCAP) data with the data from the Kinect sensor. This radar response model is discussed in-depth in Section 3.

In the present work, the Time-Frequency Toolbox (TFTB) [15], is used. It was developed primarily by François Auger and Patrick Flandrin in association with Centre National de la Recherche Scientifique (CNRS) and is an extensive toolbox available for MATLAB and GNU Octave. It contains many time-frequency analysis tools and algorithms, among them some used in this thesis. Specifically the STFT, CWV and SPWVD are all available in the toolbox. It has been subject to modifications of the framework but the algorithms themselves have remained unchanged.

## 1.3 Scope and Limitations

There is obviously a limitation to how many DE algorithms that can be used. There are numerous possible algorithms and only a few will be possible to evaluate. The four DE algorithms mentioned above will be the focus of the thesis based on how promising they have shown to be in previous studies. Overly complex algorithms can be excluded out of hand despite having shown promise as they would be practically unfeasible for use in a vehicle with very limited computation power.

The thesis was originally intended to use measured data of multiple subjects walking, jogging, running and bicycling. Measurements were both planned and conducted but there was need for extensive data pre-processing for which software was unavailable. Because of this the thesis uses MOCAP data recorded by CMU. This limits the number of possible subjects and the activities they perform. There needs to be sufficient variation in subjects and numerous repetitions for the training, validation and testing of ML algorithms to work properly. Otherwise they may not achieve their real potential. Choice of activities are dictated by what data is available. This work has limited itself to "Walking", "Running" and "Boxing".

The choice of ML algorithms is limited by their performance in previous studies and the fact that designing, training and testing can take a significant amount of time. Any method that requires a library to be present, such as k-NN or DTW, can be excluded as it is not realistic for every vehicle to carry a large and ever

growing database with it. SVMs require features to be extracted which means as more classes arise more features are required.

ANNs and CNNs have all been shown to work in combination with $\mu$D signatures in previous studies. These algorithms also have plenty of tools available due to their popularity. Because of this the thesis will limit itself to examining these three algorithms.

## 1.4 Contributions

The thesis works has resulted in the following contributions:

- Radar responses were simulated given MOCAP data and used as input to the STFT, CWT, WVD and EMD algorithms.
- An ANN and two CNNs were designed and implemented for comparison to classify human activities with the help of the above time-frequency analysis algorithms.
- ANN and CNN performance in conjunction with STFT, CWT, WVD and EMD was evaluated. Different observation times were part of the evaluation. The combination of CNN and EMD was a novel method of human activity classification. EMD showed much promise when classified even with a small network. CWT together with the ANN also proved to be a strong classifier.

## 1.5 Thesis Outline

The work that has been done during the duration of this thesis will be described in this report. The structure is as follows. First the Theory section, Chapter 2, starts with a relevant radar basics. This is followed by a presentation of the time-frequency analysis algorithms used along with the ML algorithms that have been part of the work.

The Methods section, Chapter 3, follows where the process behind data acquisition and refinement is explained in great detail. The intricacies of how different time-frequency analysis algorithms were applied are also presented. Lastly the methodology behind the design of the ANN and CNNs is described. Finally Chapters 4, 5 and 6 cover the Results, Discussion and Conclusion respectively. In the Results the performance of different combinations of time-frequency analysis algorithms and ML algorithms are presented for different kinds of input data. In the Discussion the data, models and algorithms chosen are discussed along with reflections on the results. Finally the conclusions of the work that has been done and suggestions for future work make up the last part of the report.

# 2
# Theory

This section presents the theory behind $\mu$-Doppler signatures, what kind of simulation model is used and the DE algorithms. Finally how ANNs and CNNs work is covered.

## 2.1 Doppler and $\mu$-Doppler

The frequency shift caused by the relative motion between the transmitter sending a wave and the object reflecting it is called a Doppler Shift and the phenomenon is called the Doppler Effect. Mathematically the frequency of the observed reflected waveform, $f$, can be described as following with $f_0$ being the transmitted frequency, $c$ the speed of light through the medium and the relative speed between the transmitter and reflector being $\Delta v$,

$$f = (1 + \frac{\Delta v}{c}) f_0 \tag{2.1}$$

where the Doppler Shift $\Delta f$,

$$\Delta f = \frac{\Delta v}{c} f_0 \tag{2.2}$$

where observed frequency, transmitted frequency and Doppler Shift are all related through $\Delta f = f - f_0$.

If the object reflecting the wave has several parts moving at different speeds this gives rise to what is called micro-Doppler or $\mu$D signatures. The different parts of the object generate different Doppler frequencies on top of the main Doppler created by the torso and the combination of these different frequencies is called a signature. An example of this would be a pedestrian where arms, legs, torso and head all move at different relative velocities in regards to the transmitter. Different human activities give rise to different $\mu$D signatures and those signatures will often be distinct. However, they are subject to change as the orientation and velocity of the observed objects change or as the angle of the main lobe of the array changes. This is most easily exemplified by a human walking on a flat surface. The arms, head, torso and legs all give rise to sinusoidal-like $\mu$Ds with varying frequencies and amplitudes. The ensemble of these sinusoidals form the signature. This is illustrated in figure 2.1 where different body parts are marked at different Doppler frequencies. In the example the subject is walking.

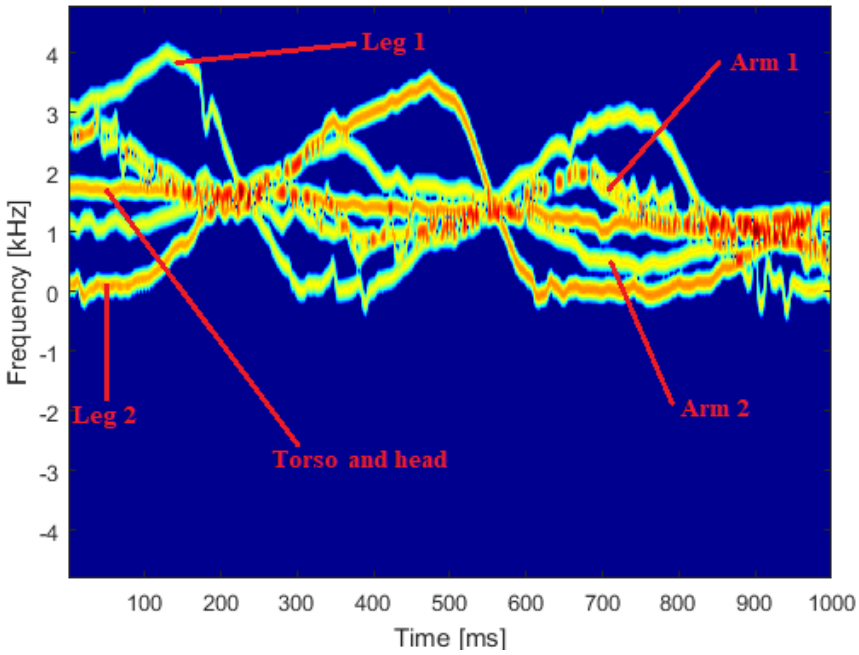



**Figure 2.1:** Body parts are marked with red with the torso and head in the center. Leg 1 is still while leg 2 moves and vice verse. The same relationship can be seen for the arms. Color represents an amplitude related to the size of the target cross section, defined in Section 2.2. The image is a spectrogram generated by the movements of subject 16 during take 8 in combination with the STFT algorithm. Details regarding this can be found in Section 3.1 and Section 3.2.1.

Examples of spectrograms of subjects running and boxing can be seen in figure 2.2 and figure 2.3 respectively.

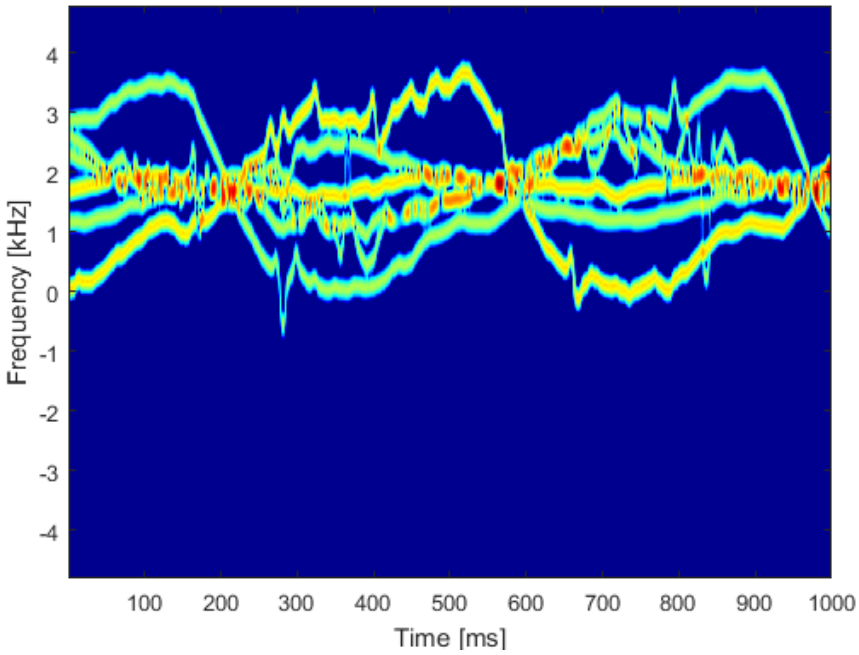



**Figure 2.2:** Spectrogram of subject running. The signature has several abrupt changes.

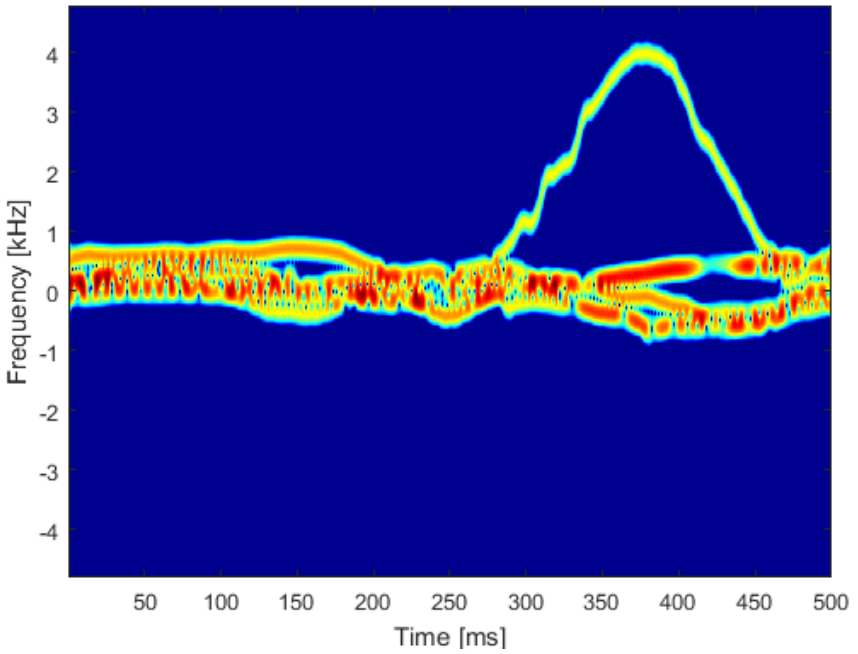



**Figure 2.3:** Spectrogram of subject boxing. The single bulge in the image is the punch being performed.

## 2.2 Radar Basics

Radars are powerful tools to determine the distances and velocities of targets but if they are to perform well they must be designed with particular tasks in mind. The relationship between radar and target properties can be expressed with the RADAR equation,

$$P_r = P_t \frac{G_t G_r \lambda_c^2}{(4\pi)^3 R^4} \sigma \tag{2.3}$$

where $P_r$ is received power, $P_t$ is transmitted power, $G_t$ and $G_r$ are transmitter and receiver antenna gains respectively and $\lambda$ is the transmitted wavelength. $\sigma$ is a property of the target called cross section which indicates how large the target is in the sense of reflectivity. Lastly $R$ is the distance between the radar and target.

The radars used on vehicles are usually Phased Array radars meaning they consist of an array of antenna elements where each element has a phase shifter. The manipulation of the phase of different elements allows for transmission in different directions up to 120° without any moving parts. The fact that nothing is moving internally within the radar units is essential if they are to be mounted on vehicles. In addition to this, carrier frequency greatly impacts radar size and how small targets a radar can resolve. Higher frequencies allow for smaller targets. A phased array radar using a high carrier frequency, for example 77 GHz, can be very physically small. This allows for multiple radar units, each no larger than a few square centimeters, to be mounted on vehicles.

To understand the radar data structure the concepts of fast-time, slow-time and channels are essential. Fast-time is the sampling rate of the radar and is a large number compared to slow-time. Slow-time is the pulse number with one pulse being a single instance of the *Pulse Repetition Interval* (PRI). The channels are the antenna elements with each element being associated with a specific channel. The relationship between these three dimensions is typically illustrated with a radar cube as seen in figure 2.4 which gives an idea of how radar data is stored.

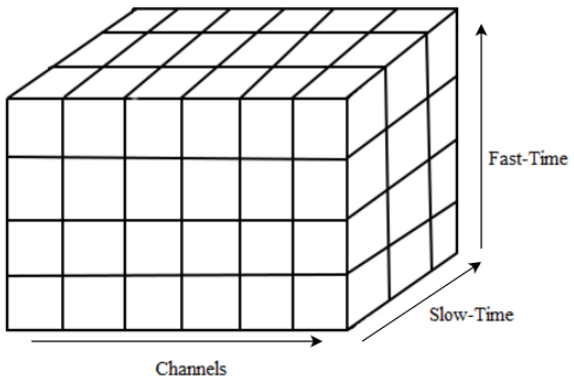



**Figure 2.4:** A radar cube showing the relationship between the channels, slow-time and fast-time.

The radar units used on vehicles are also usually Frequency-Modulated Continuous-Wave (FMCW) radars. This is also the case for the ones used on *Volvo Cars Company* (VCC) vehicles. FMCW radars have the advantage that they transmit constantly with a sliding frequency resulting in some waveform. This means the radar is not subject to the periods of waiting called *Dead Time* which other radar types are.

An example of how this sort of radar works is shown in figure 2.5 where the magenta coloured transmitted signal starts at $f_1$ and linearly increases to $f_2$ creating a saw waveform. The time delay $\Delta t$ between transmitting and receiving an echo of the same frequency is marked by the blue arrows. The measured frequency difference $\Delta f$ is marked by red arrows. The green arrows highlights $T$, which is the PRI.

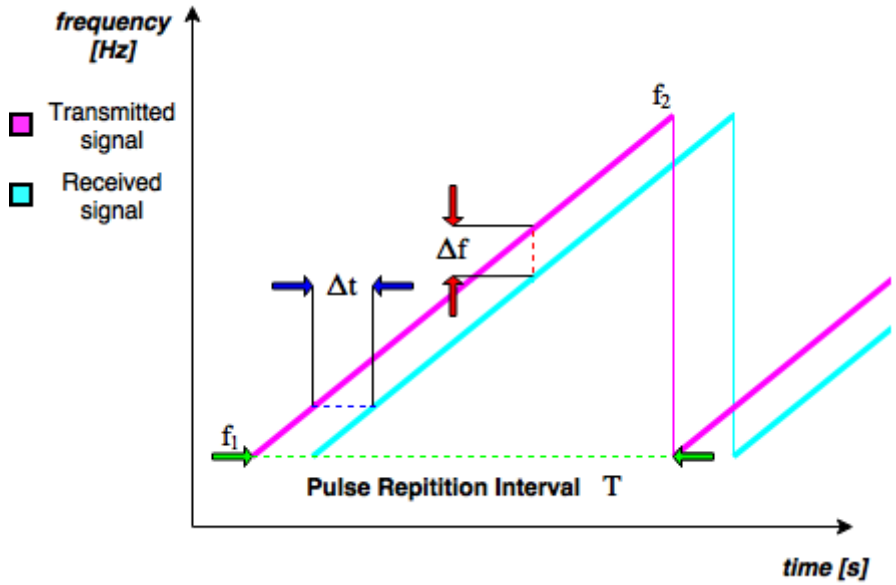



**Figure 2.5:** Illustration of a linear FMCW radar waveform from $f_1$ to $f_2$ with the transmitted signal coloured in magenta and the received echo coloured in cyan. The time delay $\Delta t$ is highlighted by the blue arrows, frequency difference $\Delta f$ by the red arrows and the PRI $T$ by the green arrows.

The distance $R$ between the radar and the reflecting target can be calculated using the expression,

$$R = \frac{c_0|\Delta t|}{2} = \frac{c_0|\Delta f|}{2(df/dt)} \tag{2.4}$$

where $df/dt$ is the slope of the waveform. This slope, the frequency shift per unit of time, is also sometimes called runtime frequency. The range resolution is directly related to the bandwidth and can be expressed as follows,

$$\Delta R = \frac{c_0}{2(f_2 - f_1)}. \tag{2.5}$$

If the target is moving, information relating to its velocity is contained in the Doppler Shift of the signal. This Doppler Shift is acquired by comparing transmitted and received frequency and compensating for runtime frequency.

### 2.2.1 Simulating a radar response

Assuming a pulsed Doppler linear FMCW radar with a single channel, the total return can be expressed as the sum over the $m$ individual points observed using the expression below [12][14],

$$s_h(n,t) = \sum_{i=1}^{m} a_{t,i} rect(\frac{\hat{t} - t_{d,i}}{\tau}) e^{j[-2\pi f_c t_{d,i} + \pi\gamma(\hat{t} - t_{d,i})^2]} \tag{2.6}$$

where $\hat{t}$ is the time relative to the start of each PRI $T$. The time $t$ is related to $\hat{t}$, pulse number $n$ and $T$ through $t = T(n-1)+\hat{t}$. The amplitude is represented by $a_{t,i}$ meaning there is a specific amplitude for each point $i$ observed at any given time. The time delay is represented by $t_{d,i}$. $\tau$ is the pulse width, $c$ the speed of light, $\gamma$ the chirp slope and $f_c$ the center frequency of the transmitted signal. $rect()$ refers to the rectangle function.

The amplitude for some point $i$ at some time $t$ is seen below [12],

$$a_{t,i} = \frac{G\lambda\sqrt{P_t \sigma_i \sigma_n}}{(4\pi)^{1.5} R_i^2 \sqrt{L_s}\sqrt{L_a}\sqrt{T_{sys}}} \tag{2.7}$$

where $G$ is antenna gain, $P_t$ is transmit power, $\lambda$ the transmitted center wavelength, $\sigma_i$ the cross section associated with a given point, $\sigma_n$ the noise standard deviation and $R_i$ the distance of point $i$. $\sqrt{L_s}$ and $\sqrt{L_a}$ are system losses and atmospheric losses respectively and finally $T_{sys}$ is the system temperature.

For the purpose of this work the format of the radar response data needed to be over slow-time at the range bin of the peak power output. To achieve this the two above equations can then be combined into the final expression below [14],

$$x_p[n] = \sum_{i=1}^{m} a_{t,i} \tau e^{-j\frac{4\pi f_c}{c_0} R_{d,i}}. \tag{2.8}$$

The simulated radar response, the complex signal $x_p[n]$, is then ready for further processing.

## 2.3 Time-Frequency analysis: Doppler Extraction

To emphasize a $\mu$D signature in a signal it is digitally processed using time-frequency analysis. This procedure is in the context of $\mu$D signatures known as Doppler Extraction. There are numerous algorithms for this purpose and they all have their strengths and weaknesses. While all DE algorithms used are explained in more detail later in this section, it should be noted that they all have some fundamental things in common. They all aim to extract time-frequency information from complex or sometimes real input signals. There is also a trade-off between time resolution, frequency resolution and complexity. In accordance with the Heisenberg-Gabor limitation it is not possible to simultaneously have high resolution in time and frequency [16]. It is however possible to work around this limitation with some of the methods presented in this section.

There are different names to describe the resulting time-frequency representations, also known as a time-frequency maps, generated by DE algorithms. One common name is *spectrogram*. In order to avoid confusion the term will be used to describe the output images from the STFT and SPWVD methods. The reader should, however, be aware that the term spectrogram, outside the context of this document, is sometimes used as a general name for time-frequency representations generated by any method presenting all the information in the same image. The results generated by Wavelet Transforms are called *scalograms* and will be referred to as such. In both spectrograms and scalograms the color axis indicates an amplitude related to the cross-sections of the target observed.

### 2.3.1 The Short-Time Fourier Transform

The Short-Time Fourier Transform, STFT, is the simplest extraction algorithm considered and is essentially a windowed Fourier transform. It can be expressed as,

$$X_s(\omega, \tau) = \int_{-\infty}^{\infty} x(t) e^{(-jt\omega t)} w(t - \tau) dt \tag{2.9}$$

where $w(t - \tau)$ is a window function and $\tau$ is the center of the window. The signal $x(t)$ is then slid through the window with some overlap from one windowing to the next. This procedure brings forward the spectral content related to a given time interval. Along with overlap, the size and shape of the window determines what the resolution will be in time and frequency. For example; a regular Fourier transform can be thought of as a STFT with an infinite window. It has the best possible spectral resolution but no temporal resolution. Frequency and time resolutions $\Delta\omega$ and $\Delta t$ respectively are related through $\Delta\omega\Delta t = C$ where $C$ is a constant meaning there is a trade-off between resolutions.

Consider a signal containing two frequency components that abruptly change at some point in time. Such a signal can be seen in figure 2.6 which is a time domain signal used as input to a STFT using a Kaiser window function. The result is the spectrogram seen in figure 2.7. An abrupt change in frequency over time is clearly visible in both figures.

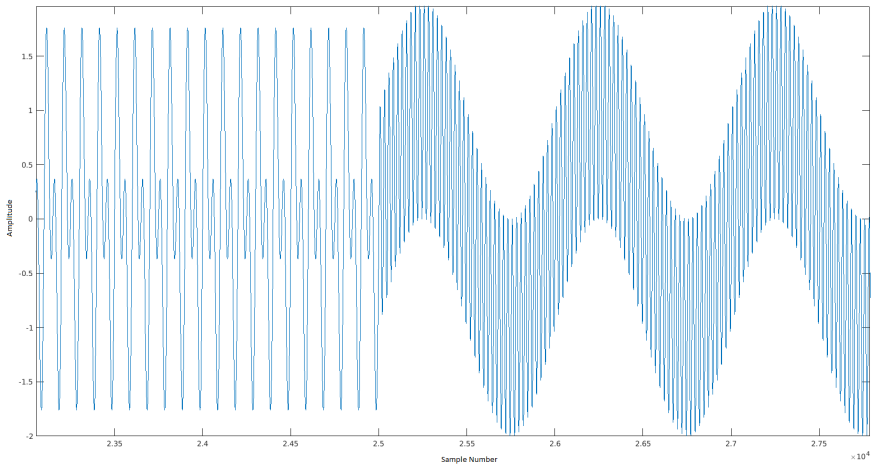



**Figure 2.6:** An example signal containing two frequency components that abruptly changes to two other frequency components at 25 000 samples while amplitude remains unchanged.

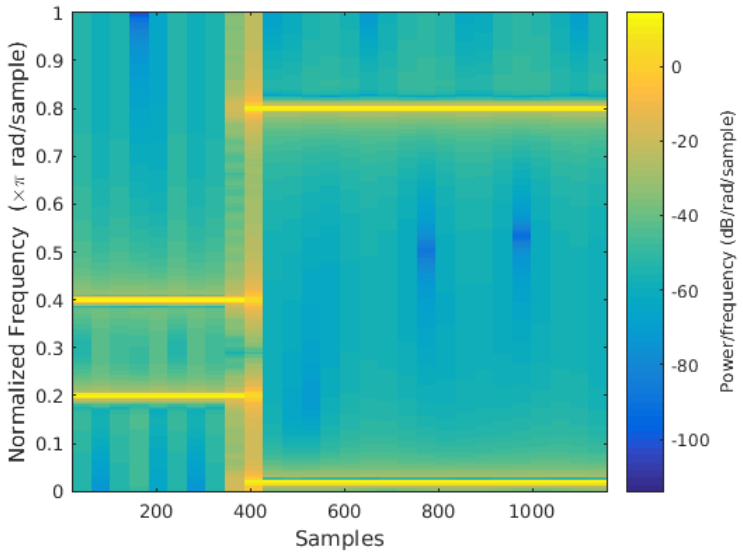



**Figure 2.7:** The spectrogram resulting from the input signal in figure 2.6. The abrupt frequency change at 25 000 samples can clearly be seen at sample 400 of the spectrogram. The number of samples is different in the spectrogram compared to the original signal as a result of the size of the chosen window function.

### 2.3.2 Continuous Wavelet Transform

Continuous Wavelet Transform is a popular algorithm for image processing and time-frequency representation. It adapts its shape to give good resolution in time and frequency as needed. Wavelet transforms use a function $\psi$ called the mother Wavelet which is an oscillation of finite length with some shape chosen in relation to

the shape of the input signal. The CWT $F(a, b)$ for some wavelet $\psi$ can be described as,

$$F(a,b) = \frac{1}{\sqrt{|a|}} \int_{\Re} \psi\Big(\frac{t-b}{a}\Big) f(t)dt \tag{2.10}$$

where $f(t)$ is the input signal, $a$ the scaling factor and $b$ the translation. The scaling factor compresses or dilates the mother wavelet while the translation factor shifts it along the time axis. This ability to manipulate the wavelet through two parameters allows for powerful time-frequency analysis when the input signal has rapid and abrupt changes. The performance of the transform is strongly tied to the choice of mother wavelet as the shape should correspond to the shape of the signal analysed. This is both a strength and a weakness as it allows for much versatility when analysing a specific signal but may suffer if input signals differ to much in their behavior.

In this work the choice of mother wavelet is the Morlet wavelet, a shape that is popular in the field [17]. As mentioned in Section 2.1 the expected shape is sinusoidal or partially sinusoidal. The Morlet wavelet is mathematically described as following [30],

$$\Psi(t) = e^{j2\pi\omega_0 t} e^{\frac{-t^2}{\sigma}} \tag{2.11}$$

where $\omega_0$ is the central frequency and $\sigma$ is the bandwidth parameter. A Morlet wavelet is easily generated in MATLAB and has a shape as seen in figure 2.8.

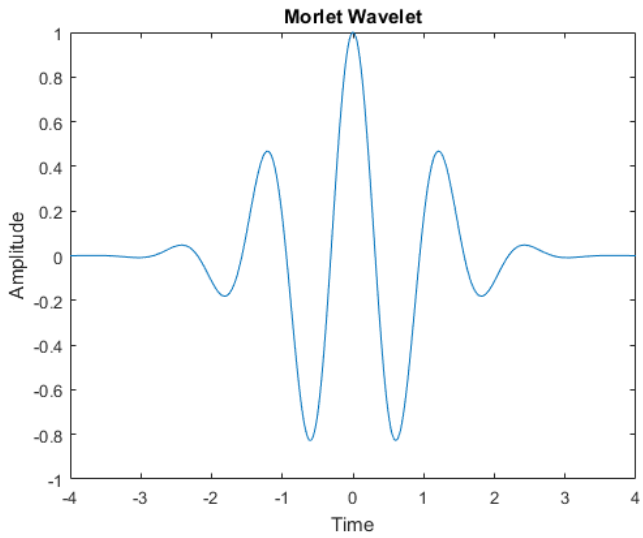



**Figure 2.8:** The shape of a Morlet Wavelet where X-axis and Y-axis are generic indications of time and amplitude respectively.

### 2.3.3 Smoothed Pseudo Wigner-Ville Distribution

Wigner-Ville Distribution (WVD) is a form of bilinear analysis that allows for high resolution in both time and frequency at the price of the signal components interfering with one another. The interference is called cross-terms and must be filtered out

for the time-frequency representation to be useful which introduces more complexity. In its most basic form it is defined as the Fourier transform of the auto-correlation of $s(t)$. Mathematically it can be expressed as,

$$S(t,f)_{WV} = \int s(t+\frac{\tau}{2})s^*(t-\frac{\tau}{2})e^{(-j2\pi f\tau)}d\tau. \tag{2.12}$$

As mentioned above it introduces cross-terms which disturb the interpretation of the output signal. Cross-terms arise when a signal contains multiple components in time-frequency and they can be up to twice the size of the desired terms. To deal with this interference a low-pass filtered WVD can be used. This results in a small loss of resolution in time and frequency for a large reduction in cross-term interference. A WVD in combination with a linear LP filter is part of the Cohen's class which can be mathematically expressed as below [18],

$$S(t,f)_{WVLP} = \int\int s(x+\frac{\tau}{2})s^*(x-\frac{\tau}{2})\phi(t-x,\tau)e^{(-j2\pi f\tau)}dxd\tau \tag{2.13}$$

where $\phi(t,\tau)$ is a LP filter and the corresponding Fourier Transform is $\Phi(\psi,\tau)$. The Fourier Transformed filter is known as a kernel function and the choice of kernel function impacts performance. Examples of variations of WVD using different kernels is the Smoothed Pseudo Wigner-Ville Distribution or the Choi-Williams Distribution.

In this work the Smoothed Pseudo Wigner-Ville Distribution (SPWVD) is used in order to minimize the cross-terms that arise from bilinear transforms. This obviously adds complexity to the method but also improves its quality. The mathematical expression for the SPWVD is [19],

$$SPWVD(t,f) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(u)h(\tau)x(t-u+\frac{\tau}{2})x^*(t-u-\frac{\tau}{2})e^{-j2\pi ft}dud\tau \tag{2.14}$$

where $g(u)$ and $h(\tau)$ are real-valued window functions and $x(t)$ is the input signal. One window function is chosen in regards to performance in the time domain and the other in regards to performance in the frequency domain.

### 2.3.4 Empirical Mode Decomposition

Empirical mode decomposition (EMD) is an algorithmic approach to analyze non-stationary and non-linear time varying signals. A non-stationary process refers to one that does not change when shifted in time. In other words the mean and variance, if defined, remain the same. EMD is a integral part of the Hilbert-Huang transform and breaks down signals into several components allowing for good resolution for each particular component [20].
The purpose of the EMD is to break down the signal into *Intrinsic Mode Functions* (IMFs) [21]. The idea of the algorithm is simple. Given an observation $x(t)$ a transform is applied to get a representation in the form:

$$x(t) = \sum_{k=1}^{K} a_k(t)\phi_k(t) \tag{2.15}$$

The algorithm performs best when a signal is composed of a fast oscillation on top of a slow oscillation. The algorithm locally identifies the fastest oscillation, removes it from the original signal, and then continues iterating. Pseudo-code for this algorithm is as follows:

1. Identify local maxima and minima of the signal
2. Form an upper and lower envelope by interpolation, usually cubic splines
   (a) subtract the average of the two envelopes from the signal. This is called an *iteration*
   (b) iterate until: number of extrema is equal to number of zeros $\pm 1$
3. Now the IMF is obtained, saved and subtracted from the signal
4. The remaining signal is called the residual. The process is continued, generating IMFs until a certain number of iterations has been completed.

Consider the signal $x(t)$ below,

$$x(t) = \sin(\pi t) + \sin(2\pi t) + \sin(6\pi t) + \sin(13\pi t) + \sin(17\pi t) \tag{2.16}$$

The signal above can be seen at the top of figure 2.9 after which three IMFs follow. The residual is seen in figure 2.10 at the very bottom of the image.

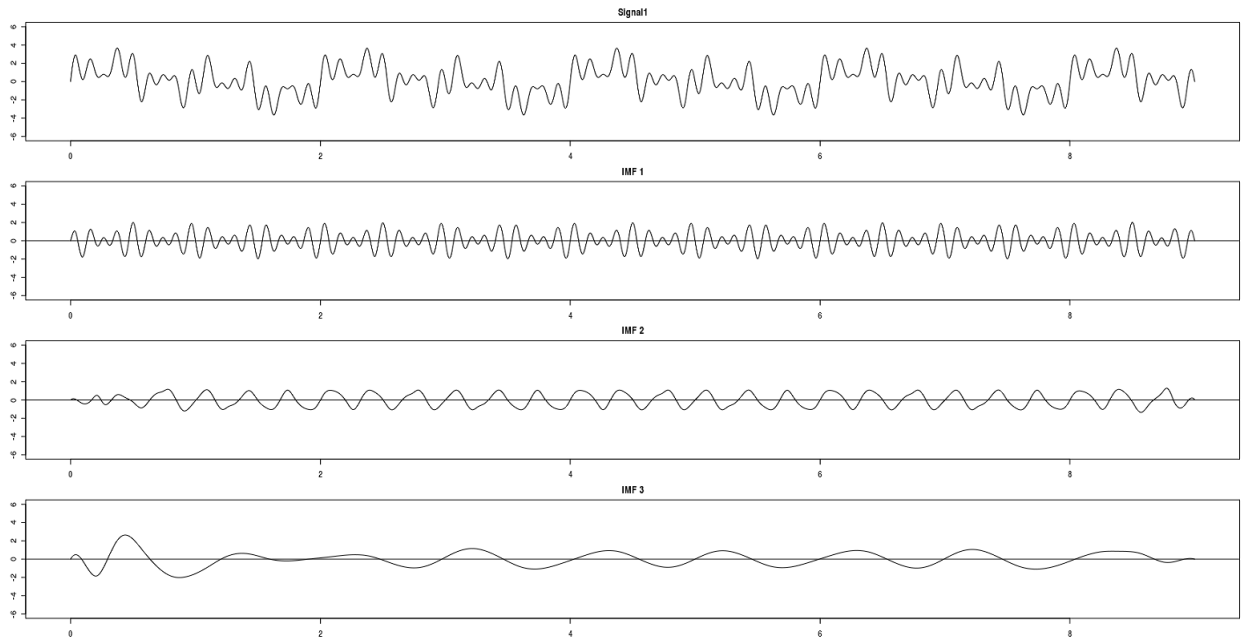



**Figure 2.9:** The signal described by equation 2.16 can be seen at the very top with the three extracted IMFs following.

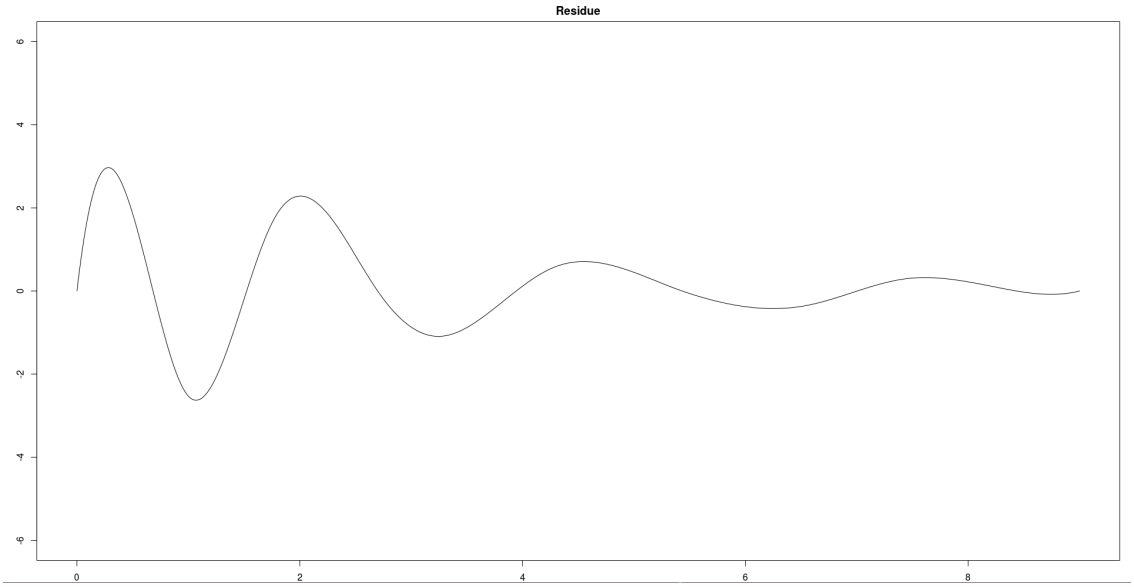



**Figure 2.10:** Residue remaining after employing the EMD algorithm on the original signal.

The EMD can be expanded to take complex arguments by the use of the bivariate EMD algorithm, where the signal is no longer contained in a fast varying envelope but rather a rotating cylinder travelling through three-dimensional complex space.

## 2.4 Machine Learning algorithms

Machine Learning (ML) algorithms are a class of algorithms aiming to give computers the ability to perform specific tasks without explicitly being programmed to do so. More concretely ML algorithms have been summarized by Tom M. Mitchell with the succinct quote [22]:

*"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."*

Experience refers to the data-driven approach where the program is taught to "learn" through examples. Generally speaking as more examples are fed into a ML algorithm the performance improves. The task could be anything from classifying images into categories to identifying a speaker from a voice recording. Performance is measured by how well the ML algorithm does this categorization.

ML algorithms can be divided into the categories supervised and unsupervised. Supervised learning refers to a training process in which the examples are coupled with the correct answer, called a label or tag, which the algorithm uses to alter its state and improve as more training examples are supplied. A simple example of this occurs if the task is to classify animals, the training examples will consist of images of an assortment of animals while the label will be the corresponding name, e.g. an image of a cat paired with the label "cat", an image of a dog paired with the label "dog" and so on.

Unsupervised learning refers to training sets where the labels are not provided, such as a cluster of points on a two dimensional Cartesian plane. The task of the machine is then to find its own weights to separate the data set.

There is a lot of variation when it comes to ML algorithms and the choice is dependent on the task at hand, computation time, memory constraints, and the nature and size of the available data set. In this work only supervised algorithms, specifically ANN and CNN, are used.

### 2.4.1 Deep learning and Artificial Neural Networks

Neural networks for the purpose of this thesis comprise of ANN and CNN. Deep learning commonly refers to neural networks (NN) with many layers and the CNN can be thought of as an extension of ANN.

ANN is a ML algorithm featuring a network of neurons, also called nodes, interconnected by vertices. A diagram of a typical ANN topology with a single hidden layer is shown in figure 2.11 where an ANN with a size $N$ input layer, a size $P$ hidden layer and a size $K$ output layer can be seen. The size of the input layer is determined by the size of the input data. Input to ANNs are usually in the form of images and that is also the case in this work. Image format is quantified by height and width in pixels along with a colour channel usually ranging from 1 (for grey scale) to 3 (for RGB images). Input layer size is calculated with $N = a \times b \times c$ given an image with number of pixels $a$, $b$ and $c$ in height, width and depth. The output layer size is based on the number of classes the data is going to be split into.

The hidden layer is designed according to the classification problem to be solved. One common design choice is depth, in other words the number if hidden layers. Another is the number of neurons in each layer. Neurons are simply a set of memory locations holding the computation values at each step of the program. Each neuron is connected to the next layer by vertices (also called axons) which are the weights. Both the input layer and the hidden layer can have a bias which helps with learning by making the network more adaptive. The weights and biases are determined by training the network, giving the network the ability to learn and make decisions. The network is inspired by the biological neural networks in animal brains from which it derives its name.

### 2.4.2 Initializing and Training

When a network is first initialized the weights must be set to some values. Often a network is initialized with randomly generated numbers according to some distribution. This avoids the network getting stuck in local minima, which might happen if it was initialized with zeros. This is commonly referred to as symmetry breaking. As NNs require training to function, suitable data must be selected and labelled according to what class it belongs to. The labelled data is then further divided into a standard split of training set, validation set and testing set. This is usually in 70%, 15% and 15% proportions for training, validation and testing respectively. The training set is then used as input to the network with the training process consisting of forward and backward propagation. During this process the training weights are

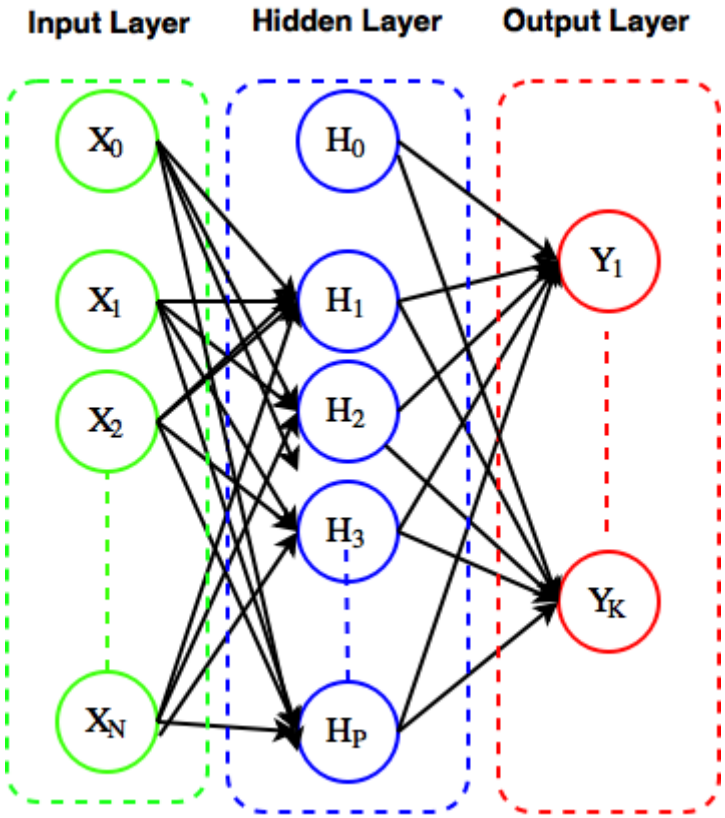



**Figure 2.11:** The layout of an ANN with the $N$ input layers coloured in green, $P$ hidden layers coloured in blue and $K$ output layers coloured in red. $X_0$ and $H_0$ are bias neurons. The connecting black arrows are known as vertices and function as weights.

adjusted on each iteration according to some pre-selected optimization routine until some convergence criteria has been reached.

The update procedure is done by the forward and backward propagation routines. In plain terms the forward propagation will take an input image and propagate it through all the layers of the network eventually producing an signal at the output layer. This signal is a class the network predicted using the current weights. Upon matching of this result to the true value provided by the labelled data an error term will be produced. Now using backpropagation this error term will be propagated through the network towards the input layer, at each stage producing a possible update to the weights in order to minimize this prediction error.

Mathematically forward propagation is comprised of matrix multiplications with the weights then passed through an activation neuron. The Sigmoid function is a common choice as an activation function. The Sigmoid decision function is defined as [23],

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.17}$$

and then a hidden layer computation can then be expressed as,

$$a^{(l+1)} = g(W^T x + b) \tag{2.18}$$

where $a$ refers to activation, $l$ is the current layer, $x$ is the input, $W$ represents the adjustable weights and $b$ is the bias. Letting the parameters $W$ and $b$ be represented by $\Theta$, the cost function is calculated as below,

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[-y_k^{(i)}log((h_\theta(x^{(i)})_k) - (1-y_k^{(i)})log(1-(h_\theta(x^{(i)})_k)] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\theta_{ji}^{(l)})^2 \tag{2.19}$$

where $m$ is the total number of training examples, $K$ is the number of output classes and $h_\theta$ is the hypothesis of the network. In the second term $\lambda$ is the weight decay, $L$ is the number of layers, $i$ is the marking of the destination node and $j$ is the origin node. Note that the bias terms are not summed over as summations start at 1. The first term is the cross-entropy of the loss, summing over all the training examples and classes. The second is called a regularization used on the weights to prevent overfitting on the training data.

In mathematical terms, backpropagation is the process of back-tracking and observing how the network reacts to a certain input with its current weights and biases. As an example, let a network have three layers. An input layer, a single hidden layer and an output layer. Starting from the output nodes the result is compared to the labels as below,

$$\delta_k^{(3)} = (a_k^{(3)} - y_k) \tag{2.20}$$

where $a_k^{(3)}$ is the result of the 3rd layer and $y_k$ is a one-hot-encoded output vector. Next the result is backpropagated using the derivative of the Sigmoid function as follows,

$$\delta^{(2)} = (\theta^{(2)})^T\delta^{(3)} \cdot g'(z^{(2)}) \tag{2.21}$$

where $g'$ is the derivative of the Sigmoid function. Equation 2.21 calculates the error values at the hidden layer $\delta^{(2)}$ by reversing the operations. The gradient is accumulated for $\theta^{(1)}$ and $\theta^{(2)}$ in the temporary variables $\Delta^{(l)}$. To simplify understanding the equation is not in vectorized form.

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + \delta_i^{(l+1)}(a_j^{(l)}) \tag{2.22}$$

In vectorized form, which is used in efficient programming implementations

$$\boldsymbol{\Delta}^{(\boldsymbol{l})} := \boldsymbol{\Delta}^{(\boldsymbol{l})} + \boldsymbol{\delta}^{(\boldsymbol{l+1})}(\boldsymbol{a}^{(\boldsymbol{l})}) \tag{2.23}$$

Yet again $l$ is the layer, $j$ is the node in the current layer and $i$ is the error in the targeted layer. The $\Delta$ terms accumulate a weighted average of the errors by iterating through all training examples. From this it is possible to normalize and calculate the partial derivatives for all nodes.

$$\frac{\partial}{\partial\theta_{ij}^{l}}J(\theta) = \frac{1}{m}\Delta_{ij}^{(l)} \tag{2.24}$$

The cost function and its partial derivatives can now be passed on to the optimization algorithm.

### 2.4.3 Backpropagation and the Loss function

The Loss function, also known as Cost function, is typically defined with the softmax function above. The purpose of the Loss function is as follow: when training the network will have the goal to minimize the loss by adjusting the weights and biases such that the softmax outputs are as close to 1 as possible for the true class and as close to 0 as possible for the other classes. An interpretation of the softmax from a probilistic can be seen below,

$$P(y_i|x_i;W) = \frac{e^{f_{yi}}}{\sum_j e^{f_j}} \tag{2.25}$$

where $y_i$ is the true label, $x_i$ is the image, parameterized by $W$. The relation is such that softmax looks at $f_{yi}$ as logarithmic probability that when exponentiated gives unnormalized probabilities. These are then normalized by the division of the sum, ensuring a total summation of 1. This can be seen as a maximum likelihood estimation.

For finding the best values of the parameters $W$ and $b$, the backpropagation algorithm is necessary. Backpropagation is a way to compute gradients of expressions using multiple applications of the Chain rule. The gradients will be required in the optimization since the optimizer needs them to calculate how to change $W$ and $b$ such that the softmax classifier outputs a higher value for the true class. Numerical calculation of the gradient is not possible when the network consists of hundreds of parameters. Backpropagation is best explained with an example:

$$f(x, y, z) = z(x + y) \tag{2.26}$$

let $s = x + y$, such that $f(s, z) = sz$. To find find the change of $f$ in relation to $x$ one must find $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial s}\frac{\partial s}{\partial x}$ with the Chain rule. As such it is trivial to find the relations of all three variables to the output $f$.

$$\begin{aligned} \frac{\partial f}{\partial x} &= z \\ \frac{\partial f}{\partial y} &= z \\ \frac{\partial f}{\partial z} &= x + y \end{aligned} \tag{2.27}$$

Gradients can be calculated locally and chained back all the way to the input. The only prerequisite is knowing the gradient of the functions. This is sufficient information to know how the parameters must be altered in order to change $f$ in the right direction. Good examples of this can be found at [28].

### 2.4.4 Optimization, Dropout and Testing

With the gradients the network can be guided in the right direction by the optimizer which will go down the slope with greatest descent. There are different available optimizers such as Momentum, Nesterov Momentum and Adam. They alternate in the way they update their learning rates.

A recent technique used in training DNNs is called Dropout. It is where a certain percentage of the neurons are turned off during a step in the training. This forces neurons to develop redundant features in the network. As the network learns that it cannot depend on one specific neuron for information, since that neuron might not be there on the next iteration, the neurons become more self-sufficient. This has been shown to improve results and prevent overfitting [25].

After the training process is done and the network has converged on some minimized loss it is ready for the test set. The network then makes predictions on this data set and the accuracy is recorded. A possible set of images called the validation images can also be used, in which different hyper-parameters such as learning rate can be adjusted prior to using the test set.

### 2.4.5 Testing NN performance

As the training process continues the network will have converged on some minimized loss. A possible set of tagged data called the validation set can be used, in which different hyper-parameters such as learning rate and regularization rate can be adjusted before test set is used. Finally a new set of data and its corresponding tags are fed into the network. This is called the test set. The network then makes predictions on these images and the accuracy is recorded. That way the network is not indirectly trained on the data, and the test set remains a true set of inputs that the network has never seen before.

### 2.4.6 Convolutional Neural Networks

Convolutional Neural Networks (CNN or ConvNets) are a specific type of NN. CNNs are commonly referred to as deep artificial neural networks containing many layers controlled in height, width and depth. The term "deep" refers to their number of layers compared with its predecessors the ANNs. Although their structure is different from that of ANNs their goal is the same. The main differences between the ANN and CNN are the amount of nodes and layers in the network and the global connectivity in the ANN vs the local connectivity in the CNN. They are currently the state-of-the-art image classification algorithm winning the Large Scale Visual Recognition Challenge (LSVRC) with the winners AlexNet (2012), VGGNet(2013), GoogLeNet(2014), ResNet(2015) all being CNNs[24]. The GoogLeNet team achieved image classification error of 6.7% versus the 5.1% achieved by the human annotator [27].

CNNs are able to make strong and correct assumptions about the nature of images such as the locality of pixel dependencies and stationarity of statistics[25]. The general idea of CNNs exploit the spatial similarity between the neighbouring pixels (for image analysis) by generating small features that are matched to different parts of the image and are then combined together. The layered network is generally understood to provide a structure where each part of the network becomes capable of solving a simple image recognition task which is then voted in by each respective part to make a final decision of the image. A typical layout for a CNN can be seen in figure 2.12

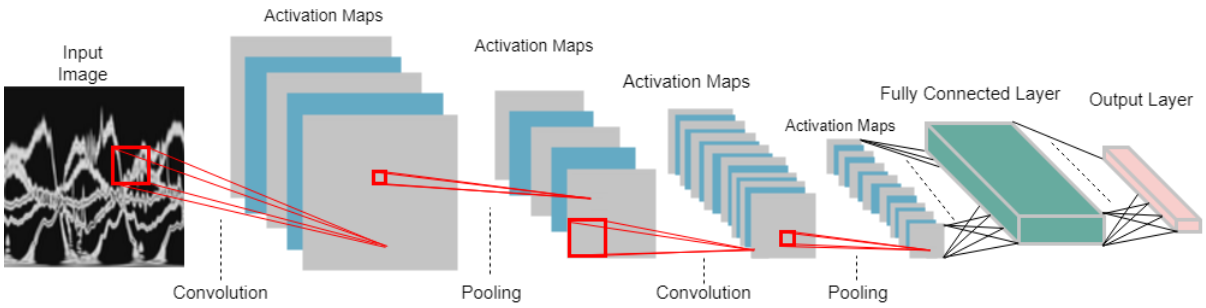



**Figure 2.12:** The layout of an CNN

The network's layers have activation functions designed to introduce competitiveness in the network. An activation function commonly used is the *Rectified Linear Unit* (ReLU) which facilitates competition between input sums. Another example is the max-pool layer which provides an explicit competition between the input neurons [26]. These functions are explained in the following sections.

Just as for ANNs the input to the CNNs is usually in the form of images. Typical image sizes vary from 28x28 pixels to 100x100 pixels or more. Larger images exponentially grow the network down the line and as such computation power sets the limit. An image the size of 100x100x3 results in a 30 000 neuron input layer.

#### 2.4.6.1 Running the network

Just like for the ANN, weights and biases are usually initialized randomly to avoid local minima. Unlike the ANN however, the CNN has a more intricate hidden layer. The first part of a CNN is the convolutional layer. True to its name the convolutional layer applies a convolution operator to a part of the input image.

The convolution is done using a filter on the image, which is small in comparison. In a CNN context a convolution is mathematically described as,

$$w^T x + b \tag{2.28}$$

where $w$ are the weights of the filter, $x$ is the subsection of the input image that the filter covers and $b$ is the bias. $w$ and $b$ are hyper-parameters to be determined later by the optimization routine. The result of this convolution gives a single point in the next layer called the activation map. After a single convolution the filter is then slid a certain distance, called a stride, and the operation is repeated. The filters, also referred to as kernels, are typically 1x1 to 20x20 pixels in size, although they can be larger. The filter depth always matches the previous layer's depth.

As an example, consider a filter of size 11x11x3, where the 3 is inherited from the previous layer's depth and the 11 is an adjustable hyper-parameter. Since the filter generates a single point on the activation map it has to be moved by a certain number of pixels before convolution is repeated. This goes on until the entire image has been covered. See figure 2.13 for an illustration of this process.

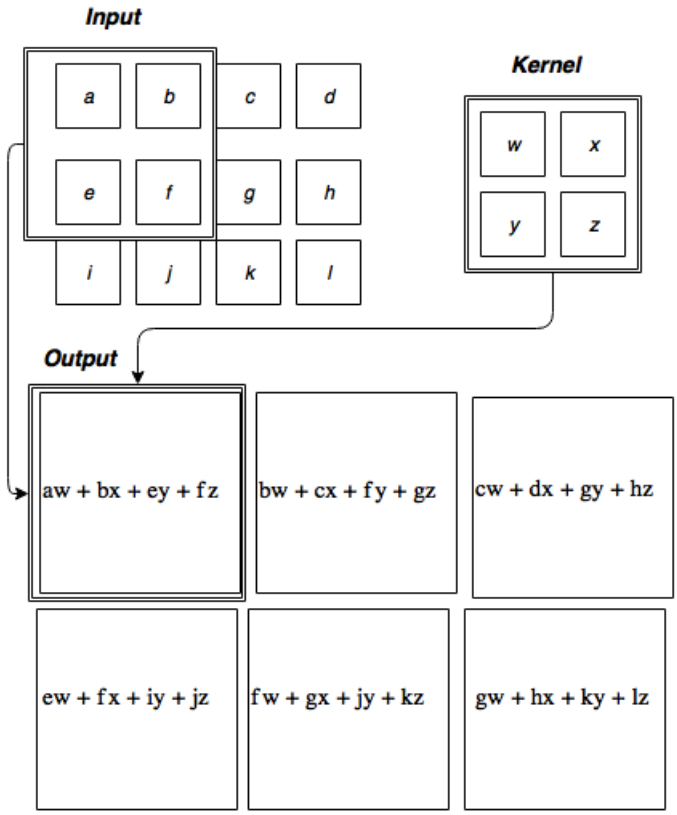



**Figure 2.13:** This is a detailed description of the convolution operation. The filter, here called a kernel, is 2x2 operating on a 3x4 input resulting in a 2x3 activation map.

The resulting activation map will have a size according to the expression,

$$A = \frac{N - F + 2P}{S} + 1 \tag{2.29}$$

where $A$ is the activation map size, $N$ is the input size, $F$ is filter size, $P$ is called a pad and $S$ is the stride. The process of padding means zero-padding around the image to keep its size consistent. This is best clarified with an numerical example. If the stride is 1 and the filter size is 13, no padding means the output map will be of size 88x88. After a few successive convolutions the image will shrink by an unacceptable amount. Another example is applying 10 filters of shape 13x13x3, with stride 1 and pad 6. This will result in 100x100x10 output neurons, with (11x11 + 1)$\times$ 10 parameters (+1 is for the bias). For a visual illustration see figure 2.14.

There are a number of filters applied at each convolutional layer. The filters generally develop simple shapes such as lines and circles. This usually occurs in the first layers while more complex shapes form in the following layers.

To introduce non-linearity into the model an activation function is used. Previously the Sigmoid function was a common choice but since the introduction of the ReLU nearly all CNNs today use this activation unit. This is due to fast computation time and because it doesn't have a vanishing gradient. Below is the equation describing the ReLU function,

$$f(x) = max(0, x) \tag{2.30}$$

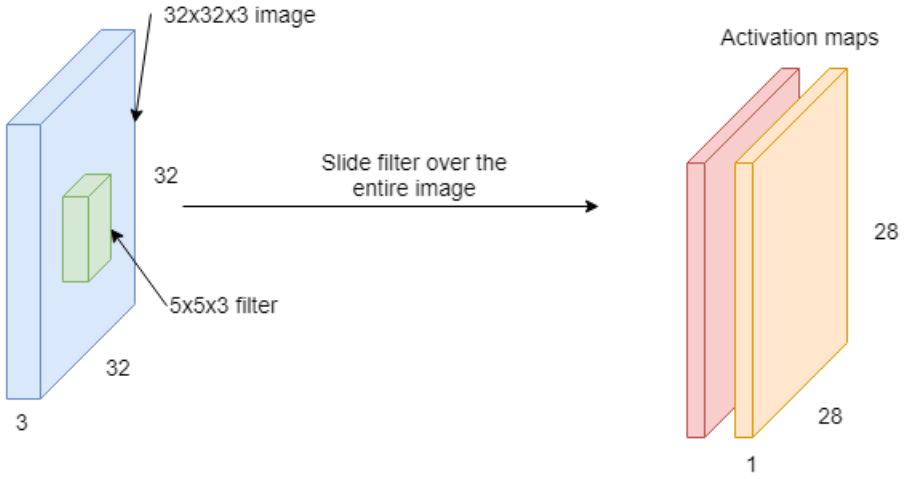



**Figure 2.14:** A 5x5x3 filter is applied on a 32x32x3 image. As it is slid over the entire input image it generates one activation map (orange) of size 28x28x1. Another filter generates the second activation map (red).

#### 2.4.6.2 Max-pooling, FC Layer and Readout Layer

The pooling layer is a layer that down-samples the input image. There are many variations of this layer, but a very popular variant is the max-pool layer. It operates on the input by taking the highest number out of the grid and move on by one stride. The general equation describing the filter size is,

$$w_2 = \frac{w_1 - f}{s} + 1 \tag{2.31}$$

$$h_2 = \frac{h_1 - f}{s} + 1 \tag{2.32}$$

$$d_2 = d_1 \tag{2.33}$$

where $w$ is width, $h$ is height and $d$ is depth of the output, $f$ is filter size and $s$ the stride. The subscripts indicate the layer the variables belong to. A 2x2 filter with a stride of two would reduce a 100x100x10 to 50x50x10, effectively taking the maximum of each neighbouring 2x2 block and reducing it to one number. In total such a reduction would be to a quarter of the original size. For a clear illustration of the max-pooling operation see figure 2.15. The max values in each quadrant and picked and turned into a new quadrant.

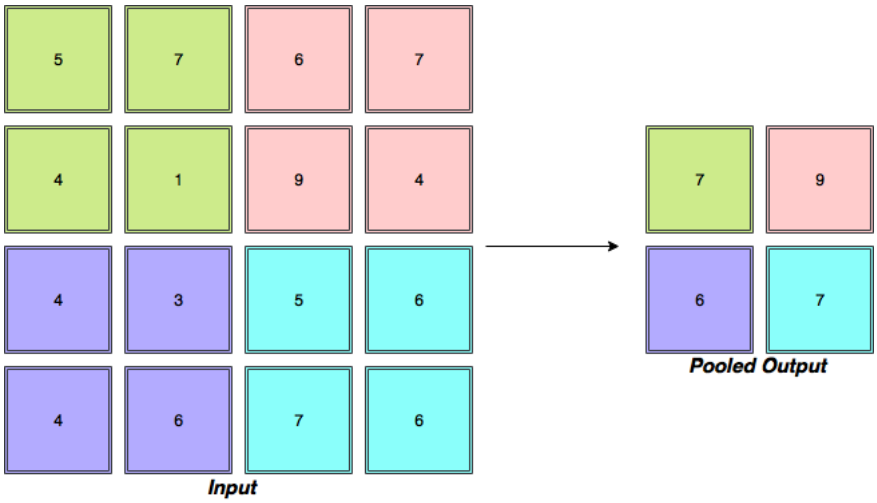



**Figure 2.15:** The maximum value in each quadrant of the Input square is selected and separated into a new quadrant called the Pooled Output.

The *Fully Connected* (FC) layer resembles an ANN network where there is no local connectivity but all the nodes in the current FC layer are connected to all nodes in the previous layer. This can be seen in figure 2.12 on the right end of the network.

In effect this combines all of the previous layer's decisions into several hundred neurons. The previous layer is flattened and a matrix multiplication operation between the weights of the FC layer and the input from the previous layer completes the operation.

The Readout Layer is typically the last layer of the network. It is where the FC layer is funnelled into all the class outputs. After this a softmax function can assign probabilities to each class according to,

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.34}$$

which acts as a normalization function so that all class scores have a range from [0,1] which can be interpreted as a probability.

# 3
# Methods

This chapter covers the procedure of going from the original MOCAP data files to a final result, describing the DE and ML algorithms used. The procedure is visualized in figure 3.1 where red represents modelling and pre-processing, orange represents DE algorithms, yellow highlights ML algorithms and green is the final analysis of the result.

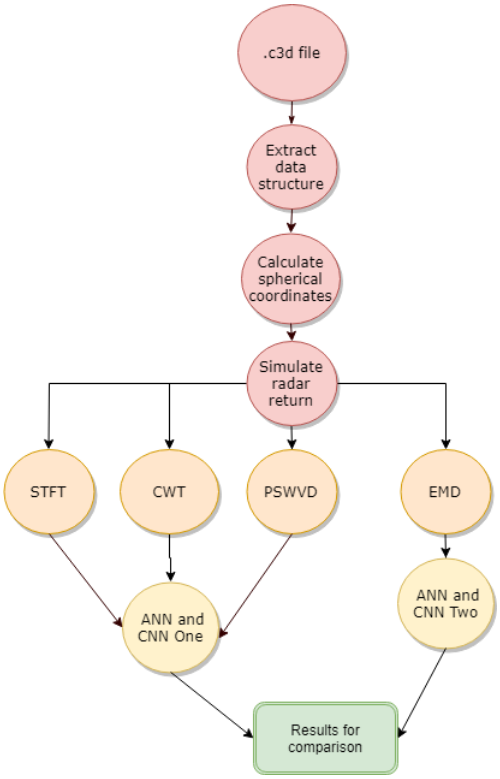



**Figure 3.1:** A flow chart of the thesis methodology with data pre-processing (red), DE algorithms (orange), ML algorithms (yellow) and final analysis (green) highlighted. STFT, CWT and PSWVD are inputs to an ANN and CNN One. EMD is input to the ANN and CNN Two. The meaning on CNN One and Two are explained towards the end of this chapter.

## 3.1 Data

Data and simulation tools were picked based on availability and how well they fit with the goals of this work. The decision was made to combine real-world MOCAP

data of humans and simulate the radar responses using that data. The process of turning MOCAP data into useful radar returns is presented below.

### 3.1.1 Choosing MOCAP data

The first step when using this data was to select relevant categories. Category choice was based on how similar it was to the real-world measurements that had been conducted as part of the original plan for the thesis. Several subjects walking and running at various speeds were available in the MOCAP database. There were also many repetitions of these activities and a lot of variation in gait. As many categories as possible were desirable and there was sufficient MOCAP data for three categories. "Boxing" was selected as the third category, in addition to "Walking" and "Running". It was selected despite being an uncommon pedestrian activity since there was some variation in subjects and a fair number of repetitions available. The "Walking" category had the largest number of subjects with 18 individuals. The category "Running" was limited to 5 individuals and "Boxing" had 4.

### 3.1.2 Preparing MOCAP data

The MOCAP data was available in multiple formats. The format used to generate Cartesian time-space coordinates was the *.c3d* format. There were also filmed versions (*.mpg*) and animated versions (*.avi*) of each recording. These were used to verify the usefulness of a given recording. As previously mentioned, the subjects carried 41 markers spread evenly across their bodies.

On the CMD website some support functions were provided to translate the *.c3d* files into a structure containing the desired data. It should be noted that even though that all *.c3d* appear to be similar some are in the format VAX-D (also called DEC) and first need to be converted into PC format requiring an additional function. Whether or not this step was required varied on a subject by subject basis.

Once converted the structure consisted of coordinates $(x, y, z)$ over the time $t$ for each of the 41 markers. Six of these markers were selected, spread across the head, torso, arms and legs. The motivation behind this was the fact that no realistic automotive radar would have the resolution to pick up $\mu$Ds from markers more densely packed.
The recordings used 120 frames per second which was significantly lower than what was needed to generate realistic $\mu$D signatures. Because of this the data from the six selected markers was interpolated by a factor of 80 turning 120 data points per second into 9600 points per second for each spatial direction. The interpolation factor 80 was kept as low as possible in order to keep down computation cost while still giving smooth curves. The spatial coordinates were also shifted in space so that they always moved towards or away from the point of observation rather than past it.

In order for the coordinates to resemble what a radar unit would observe they were converted from the Cartesian plane $(x, y, z)$ into the Spherical plane $(R, \phi, \theta)$. The last step was to assign each marker with a cross section $\sigma$. The cross sections were assumed to be constant with the legs and torso having twice the size of the arms

and head. They were also assumed to be spherical in shape allowing the $\phi$ and $\theta$ dimensions to be discarded. This choice is discussed further in Chapter 5.

### 3.1.3 Generating a radar response

The radar response simulation started with equation 2.7. To calculate the amplitude the different constants were set to values seen in table 3.1. Choice for antenna gain, pulse width, carrier frequency and transmit power were based on the radar unit used by VCC. System losses, atmospheric losses and noise standard deviation were for simplicity chosen so they would have minimal impact. The same was true regarding system temperature.

| | |
|---|---|
| Antenna Gain $G$ | 6 dB |
| Transmit Power $P_t$ | 10 mW |
| Carrier Frequency $f_c$ | 77 GHz |
| Pulse Width $\tau$ | 5 $\mu$s |
| Noise Standard Deviation $\sigma_n$ | 0 dB |
| System Losses $L_s$ | 0 dB |
| Atmospheric Losses $L_a$ | 0 dB |
| System Temperature $T_{sys}$ | 290 K |

**Table 3.1:** Table listing the constants and system parameters used for simulating the radar return.

Once the amplitude was calculated it was used in equation 2.8. The complex output $x_p[n]$ was then ready to be used as input to the different DE algorithms.

## 3.2 Doppler Extraction configurations

Once the radar responses were simulated they were used as input to the STFT, CWT, SPWVD and EMD algorithms. The ML-algorithms used different formats of information and thus additional steps were sometimes required. Gray scaling or squaring images are examples of such steps. To illustrate the process and show how results could look subject 16 take 8 from the CMD database was selected. The inputs to the DE algorithms consisted of complex signals but for illustration the absolute value of such a signal can be seen in figure 3.2.

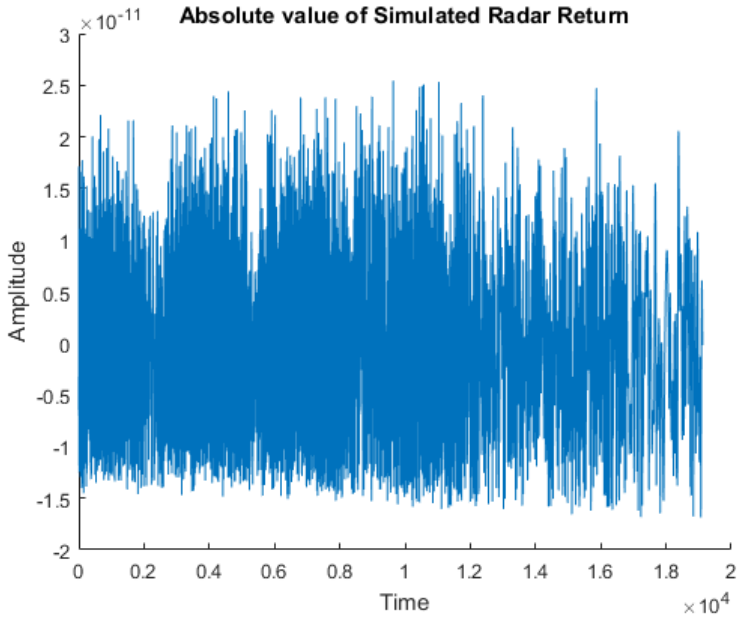



**Figure 3.2:** The absolute value of the simulated radar return from subject 16 take 8 from the CMD database.

The algorithms were run both with and without noise for all cases. In the case where noise was added it was in the form of white noise added to the spherical $R$-coordinates meaning the positions of the sensors themselves got shifted. Attempts were initially made to add the noise to the simulated radar response instead as this is where noise would realistically appear. It was however found to be very unstable and unpredictable with SNR varying wildly, sometimes completely destroying the signal and sometimes being unnoticeable. The noise level chosen resulted in a SNR of about 30 $dB$. The reason for this instability could not be determined and so noise level was chosen empirically. This noise level was however chosen for coloured images and upon visual inspection it seemed like the gray scale images had significantly lower SNR than that. Properly quantifying noise levels from gray-scale images proved to be very difficult however and was ultimately unsuccessful. Noise is discussed further in Chapter 5.

To evaluate the effect of observation time DE algorithms were run on the same data twice. First when the signal was divided into half second snapshots and again when it was divided into one second snapshots. This was important as using less observation time could mean a lot for avoiding accidents in a real life traffic situation where a reaction half a second earlier could have a large impact on the outcome. It was also important because less observation time resulted in fewer data points meaning algorithms would run faster. The complexity of these different algorithms was important to quantify due to the limited computation power available on a vehicle.

All images were saved in both grey-scale (146x110) *.png* format for the ANN and grey-scale square (110x110) *.png* format for the CNN. Monochrome images were used to speed up ML algorithm run times. In addition to this DE algorithms were timed and the timings were saved for evaluation.

### 3.2.1 The STFT algorithm

The particular STFT used in this thesis used 512 frequency bins, a Dolph–Chebyshev window function of length 71 and combined this with the complex input signal mentioned in the previous section. In figure 2.1 a spectrogram of the complex signal using STFT is seen. Note that the image is in color only to make it easier for the reader, the images used for the ML algorithms were gray-scaled versions. The peaks with highest frequencies represent the legs moving at a Doppler frequency of 4.2 $kHz$ meaning over 16 $m/s$ and the torso being centered around 1.8 $kHz$ meaning around 7 $m/s$. These speeds are clearly to high and the torso should be centered somewhere around $500 - 800$ $kHz$ with the rest of the movements shifting accordingly.

The reason for the offset was unknown but suspected to be a result of some of the time-frequency analysis tools not being designed to handle negative frequencies. Fortunately it was not important for the ML algorithms using images as because the images had their axis removed to compress them as it was considered to be redundant data, same for all images. The shift was also consistent for different subjects and repetitions.

Transformed into gray scale figure 2.1 is transformed into 3.3 which is the sort of image used as input to the ANNs.

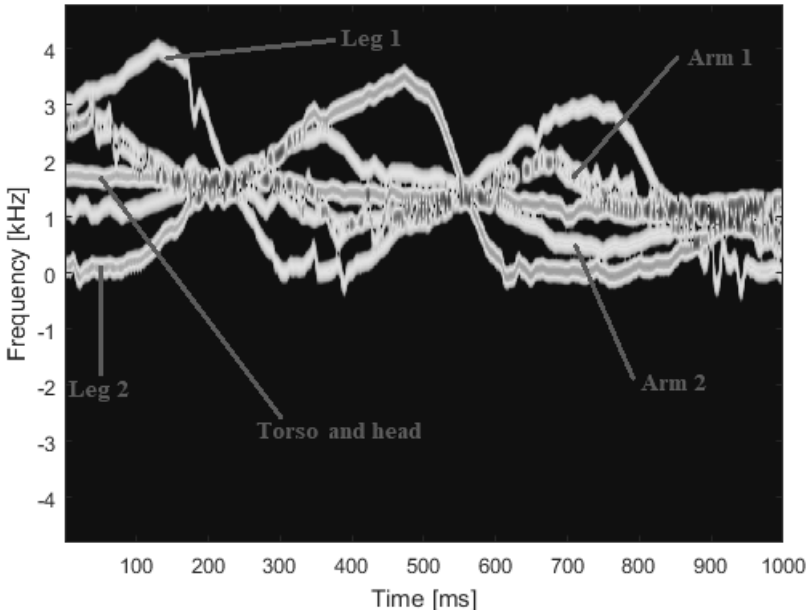



**Figure 3.3:** Figure 2.1 transformed into gray scale.

A noisy version of the same image can be seen in figure 3.4 with the gray scaled version seen in figure 3.5

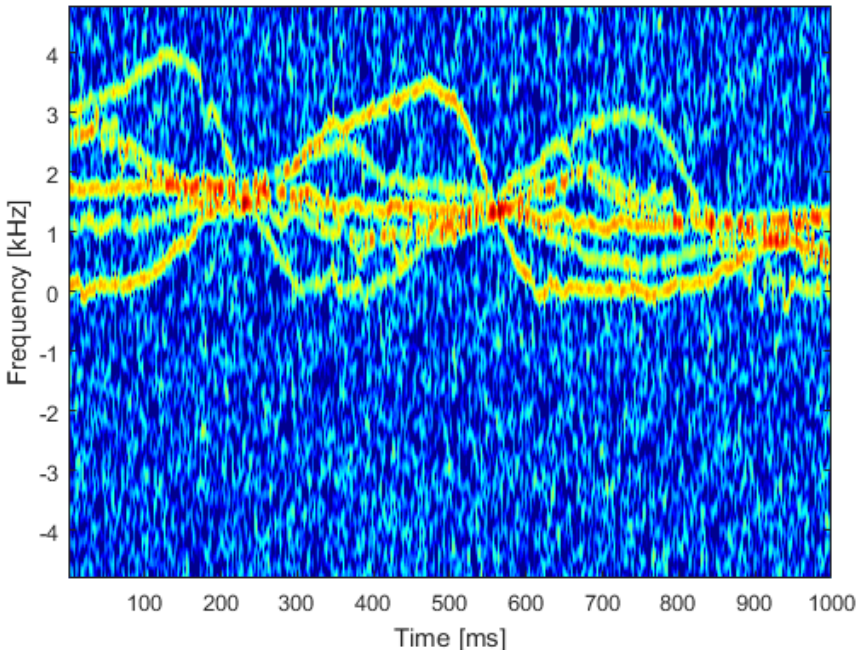



**Figure 3.4:** The spectrogram generated by the movements of subject 16 during take 8 when white noise has been added.

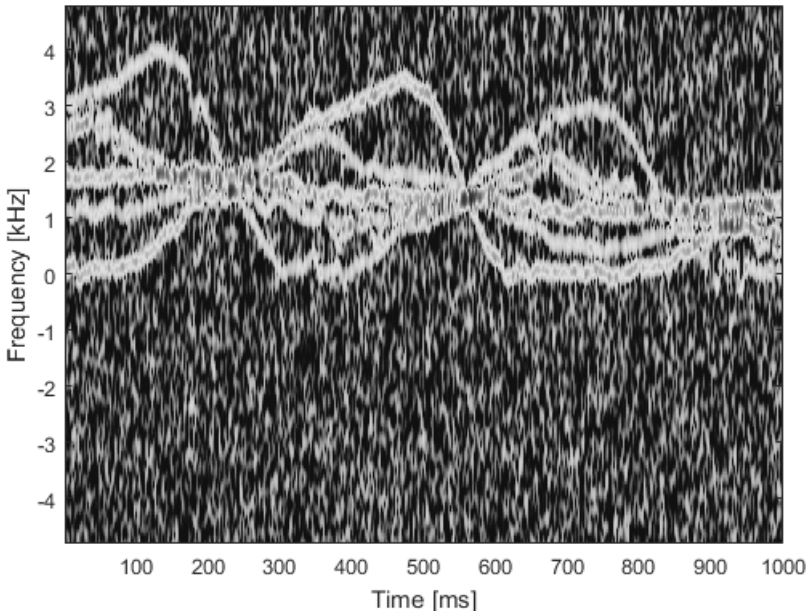



**Figure 3.5:** Figure 3.5 in gray scale. The signal is less clear in relation to the noise in comparison to the coloured version of the image.

### 3.2.2 CWT using the Morlet mother wavelet

The CWT transform took the same complex input as the STFT did. As previously mentioned a Morlet wavelet was used and it was found to be very difficult to tune settings to get acceptable performance both for very high and very low frequencies. High resolution for low frequencies can be seen in figure 3.6 where the upper part of the scalogram is fuzzy while the lower parts are sharp. The opposite is true in figure 3.7 where the lower part of the scalogram is fuzzy while high frequencies are distinct. This fuzziness was clearly undesirable and because of this the CWT was run twice, once for high frequencies and once for low frequencies. The resulting images were then cut and merged in such a way that there was high resolution across the entire frequency interval. In all scalograms negative frequencies were also folded into the zero frequency. This folding meant there was a loss of information when comparing with the STFT algorithm as it correctly represented negative frequencies. The result of cutting two images together as described above can be seen in figure 3.8. This third scalogram can be compared to the STFT spectrogram in figure 2.1 and is noticeably sharper with frequencies being more well-defined over time.

One obvious downside to using CWT was the smearing around the zero frequency limit. Another was that running two CWTs obviously took more time but it was required to get an image of comparable quality in regards to the STFT method. The final scalograms were turned into gray scale and used as input to the ANNs and CNNs.

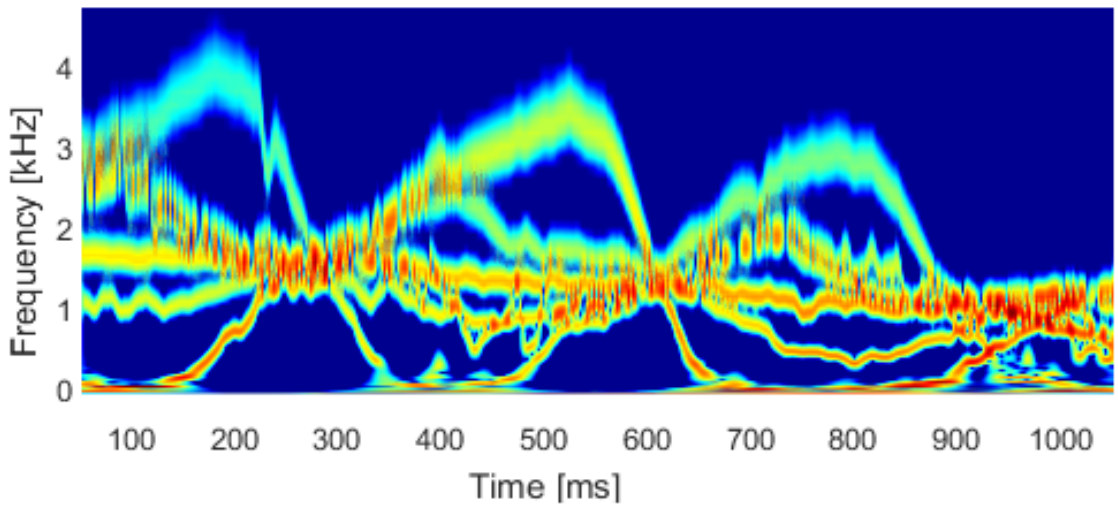



**Figure 3.6:** A scalogram of the signal generated by the movements of subject 16 during take 8. High frequencies are smeared while low frequencies are shown with high resolution.

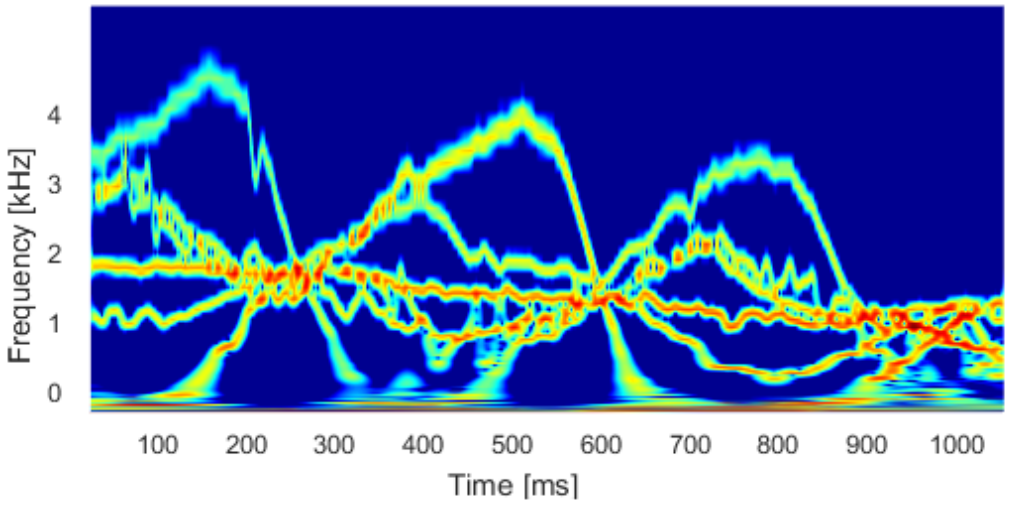



**Figure 3.7:** A scalogram of the signal generated by the movements of subject 16 during take 8. Low frequencies are smeared while high frequencies are shown with high resolution.

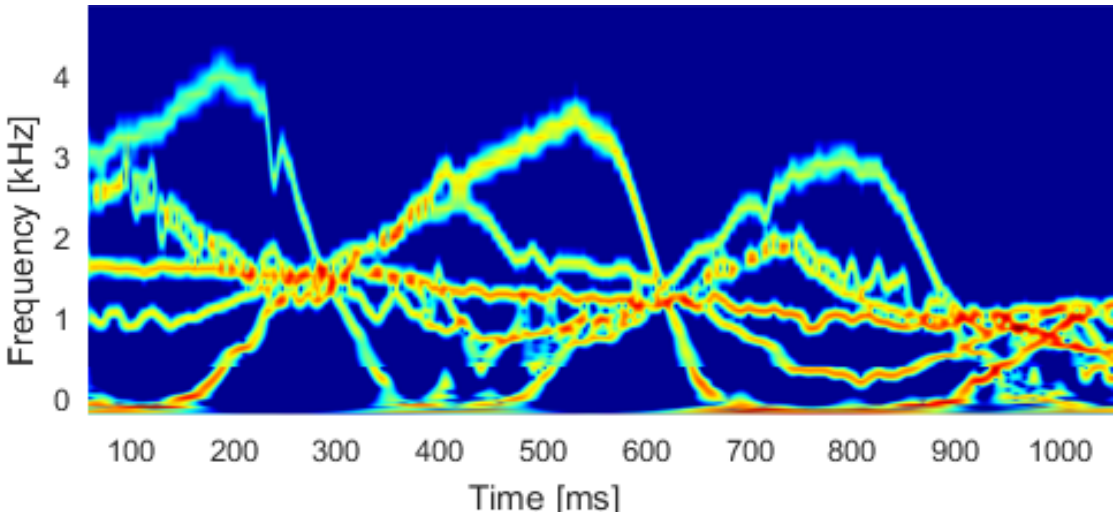



**Figure 3.8:** A scalogram resulting from the merge of the two above to get resolution in both high and low frequencies simultaneously. Negative frequencies are folded into the zero frequency resulting in a loss of information.

### 3.2.3 SPWVD

The SPWVD required two window functions. The time smoothing window used was a Dolph–Chebyshev of length 61. For frequency smoothing another Dolph–Chebyshev window of length 71 was used. Several different window functions and window lengths were tested to find a good trade-off between time resolution, frequency resolution and computation time. Longer windows gave better results at the cost of much longer computation times. The result can be seen in figure 3.9 where the

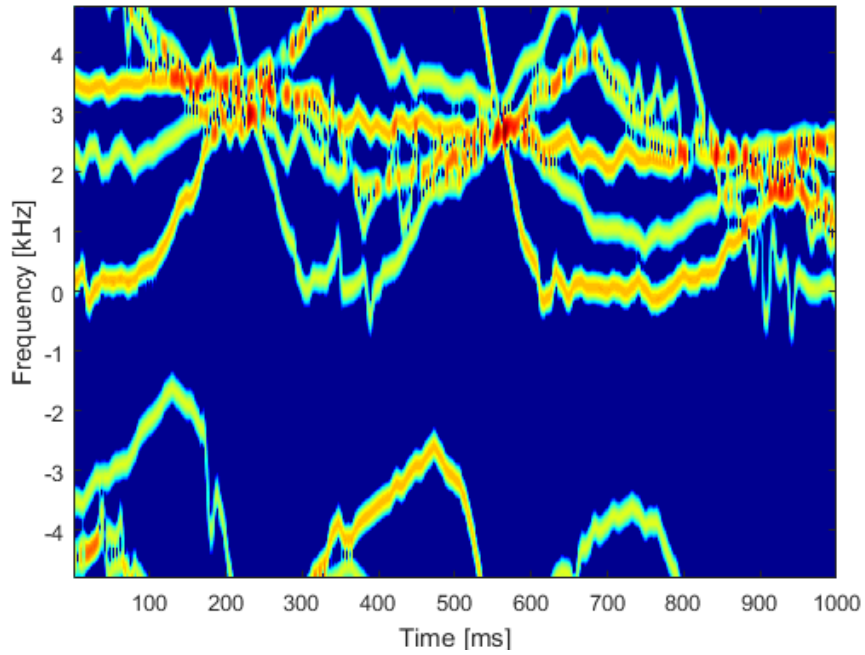



**Figure 3.9:** A spectrogram generated with the SPWVD algorithm. High frequencies have folded into the negative frequency domain destroying locality.

highest frequencies ended up in the negative frequency domain when using the same complex input as to the STFT and CWT methods.
The reason for this couldn't be determined but it was obviously undesirable. The signal frequencies were therefore shifted so they ended up in the right place using the STFT and CWT algorithms as points of reference. This shift was achieved by division inside the exponential in equation 2.8. The resulting equation was then

$$x_p[n] = \sum_{i=1}^{m} a_{t,i}\tau e^{-j\frac{4\pi f_c}{1.85c_0}R_{d,i}} \tag{3.1}$$

with the particular number 1.85 chosen as it aligned the SPWVD spectrogram frequencies with the STFT spectrogram frequencies.
The image generated after this change is seen in figure 3.10 where the high frequencies are no longer appearing in the negative frequency domain. This came at the cost of a worse time-frequency representation with loss of sharpness and some cross-terms marked in red appearing. Locality was important for CNNs and so related information in an image had to be found in nearby pixels. After this change that was the case. These images were saved in the same fashion as the outputs from the STFT and CWT algorithms.

### 3.2.4 EMD

The output from the EMD algorithm was treated a bit differently. In figure 3.11 the IMFs extracted from the complex signal by subject 16 take 8 are shown with the real part of the IMFs to the left and the imaginary part to the right. The real

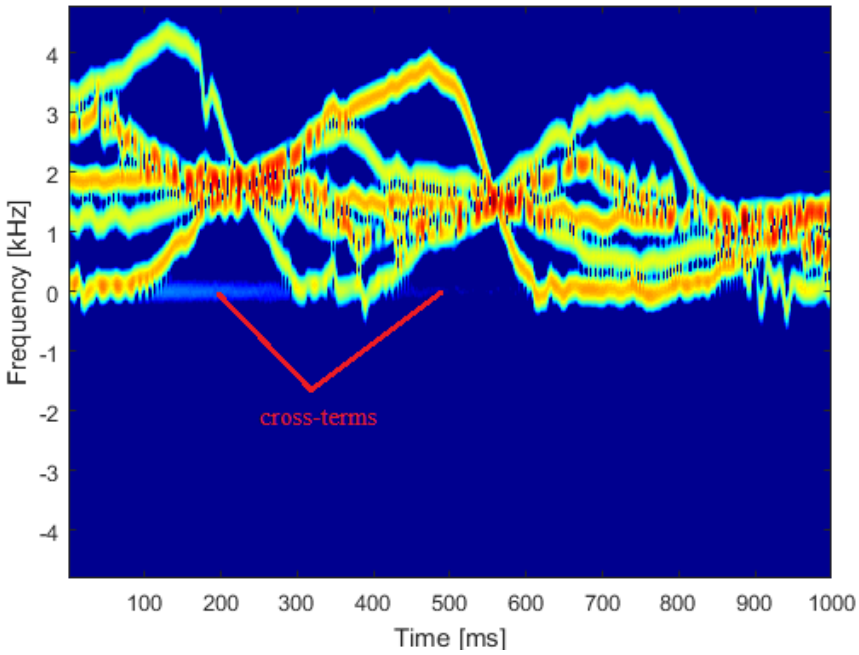



**Figure 3.10:** A spectrogram generated using the SPWVD method with shifted frequencies that represents the quality of image used. High frequencies are now where they should be but the spectrogram has less sharpness than in the previous figure. Cross-terms marked with red have also been introduced.

and imaginary parts are very similar with only a slight time shift and amplitude change between them. The IMFs in the upper part of the image contain mostly high frequency components, with lower frequency components showing themselves at higher order IMFs. The resulting images are inappropriate as input to ANNs or CNNs purely as images. This is because the images would have to have a very high resolution to retain their level of detail and images that large would take excessively long to train and test on.

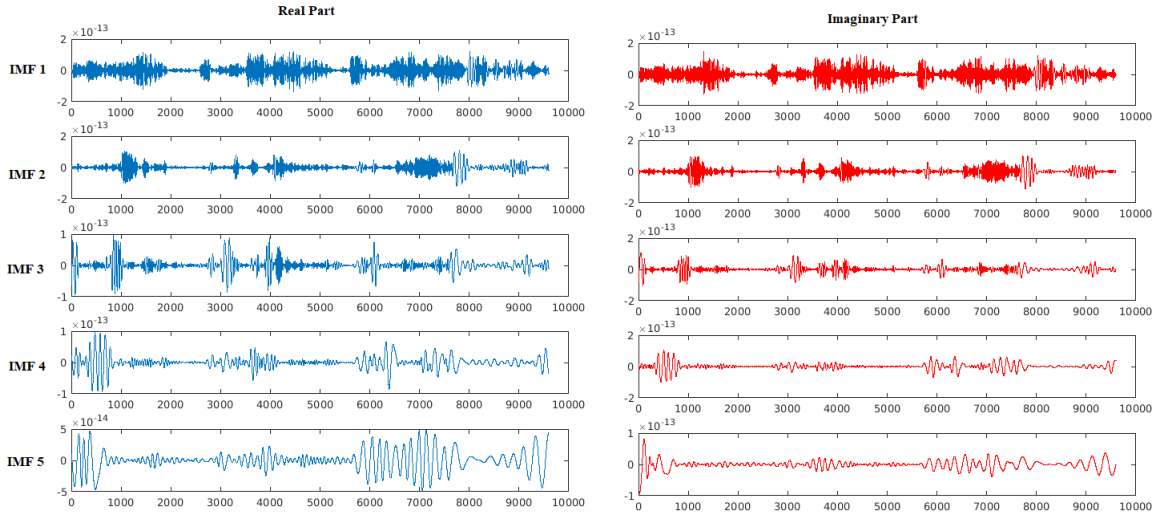



**Figure 3.11:** The first five IMFs generated after EMD has been applied to the signal generated by subject 16 take 8. A complex signal generates complex IMFs so for visualization the real and imaginary parts have been separated. The real part on the left side of the image is coloured blue and the imaginary part on the right side is coloured in red.

To turn the IMFs into a potentially suitable input they are instead first split into their real and imaginary parts. The real parts are discarded and the imaginary parts are downsampled by a factor of two. After this up to six IMFs are concatenated into one long vector. The reason only the imaginary part IMFs are used is because they need to fit into 146x110 (ANN format) or 110x110 (CNN format). If an EMD of a signal converges early and the generated IMF are insufficient to fill the vector of length 16060 (ANN) or 12100 (CNN) then the vector was zero-padded to get the right length. This is motivated by the fact that no IMFs indicate that the signal was encapsulated in the first few IMFs with very little to no residue. The input from EMD to ANN was in the form of matrices and the input from EMD to CNN was in the form of images.

There is a loss of information when part of the complex IMFs are discarded but to make sure the signal was not destroyed images like the one seen in figure 3.12 were generated using an STFT. In that particular image all the real parts of the IMFs from a signal were added together and used as input into a spectrogram. The imaginary parts were discarded. The same was done for the imaginary parts, discarding the real parts, giving the same kind of results. This can be compared to an image where the complex IMFs were added together and then used as input to a STFT, seen in figure 3.13. While there is clearly some loss of information when using only real or imaginary parts as opposed to complex, the $\mu$D signature is mostly intact.

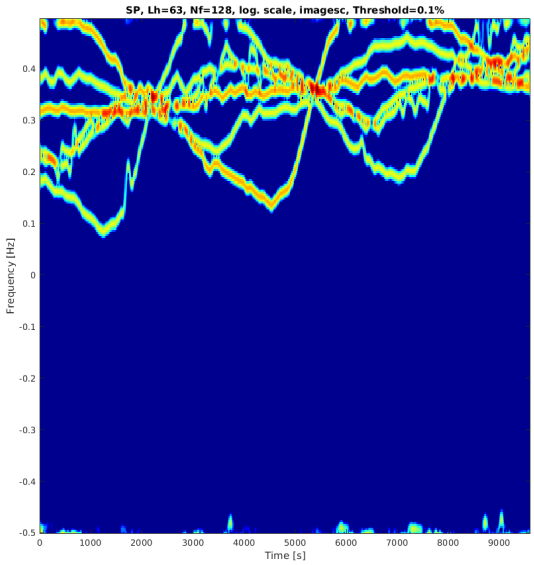



**Figure 3.12:** Reconstruction of the original signal from complex IMFs. A spectrogram was generated with an STFT from a signal made by summing all complex IMFs together.

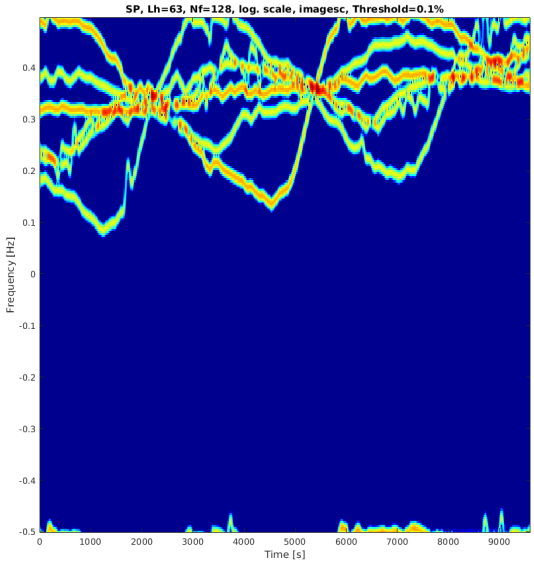



**Figure 3.13:** Reconstruction of the original signal by using only real parts of the IMF functions. A spectrogram was generated by an STFT by summing the real parts of all IMFs together. Noting that a spectrogram made from only the imaginary parts looks nearly identical.

To illustrate how different the output images from the EMD after downsampling, concatenation and reshaping, subject 16 take 8 yet again features in figure 3.14.

The lower order IMFs contain the high frequency components and can be seen in the upper part of the image.

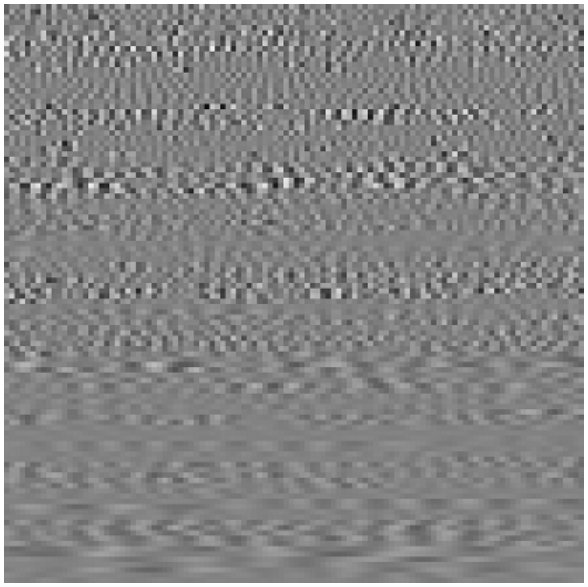


**Figure 3.14:** An immaculate image of subject 16 take 8 generated by adding the imaginary parts of six IMFs. In the upper part of the image high frequencies can be seen as oscillations in shades of grey.

## 3.3 ML Classification

The data generated by the DE algorithms served as input to the ML algorithms. The idea was to create networks classifying the three human activities "Walking", "Running" and "Boxing". The particular algorithms were selected due to their supreme results in image recognition (CNN) and simplicity and ability to work with small data sets (ANN). Both ML algorithms were run on a PC with the specifications seen in table 3.2.

| | |
|---|---|
| CPU | Intel Core i5-3570K @ 3.40GHz |
| RAM | 16384 MB DDR3 1333 MHz |
| GPU | NVIDIA GeForce GTX 680 |
| OS | Windows 10 Pro 64-bit |

**Table 3.2:** Specifications for the computer on which the ML algorithms were run.

Inputs to the CNNs and ANNs were the same images but re-scaled to different sizes. The images were divided into training sets and testing sets and tagged in accordance to their classes. As such the output classes of the ANN and CNN were a 3 class output vector in one-hot-encoding format. One-hot-encoding format means a vector where all entries are 0 except the true class which is a 1. For example if walk was the second class its vector would be [0, 1, 0].

The images were divided into training sets (70 %), validation sets (15 %) and test sets (15 %). The training sets consisted of 188 or 378 images depending on observation time while test set consisted of 41 or 81 images. Generally CNNs work on training sets consisting of millions of images. Without this luxury, images had to be augmented in order to raise the number available. In this case images in the "Running" category were horizontally flipped. This was justified since the horizontal flip of the spectrograms and scalograms simply meant the target had reversed direction, if it was moving towards the radar unit before the flip it is going to be an image of a target moving away after the flip.

All ML algorithms evaluated for one second and half second observation times for the three activities "Walking", "Running" and "Boxing". The limit to the number of available images was set by the "Running" category. With the help of image augmentation there were 90 images with good $\mu$D-signatures available while using one second observations and 180 images while using half second observations. Three Cases were investigated:

- Case One - one second observations and 90 images/vectors per category
- Case Two - half second observations and 90 images/vectors per category
- Case Three - half second observations and 180 images/vectors per category

Cases One and Two were chosen for comparison with each other and Case Three to exploit the larger number of images available.

### 3.3.1 Artificial Neural Networks

Images were in format 110x146 pixels in gray-scale. The network had one hidden layer made up of 45 neurons. There were three output neurons as previously mentioned. Using images of the above stated size resulted in an input layer with a size of 16060. In the ANN designed there were 722700 weights from the input layer to hidden layer and 135 from the hidden layer into the output layer. These weights were randomly initialized using a normal distribution.

The optimizer chosen was a nonlinear conjugate gradient method with a Polak–Ribière flavour. Max number of iterations was chosen to be 200. The learning rate step $\lambda_{ANN}$ that had the best results was 0.001 as there the network did not show any oscillatory behaviour, which is indicative of too large steps. It is also important to note that convergence was adequately fast on this setting, since slow convergence is an indication of too small steps. Hidden layer size, number of iterations and $\lambda_{ANN}$ were all found by training and running the ANN on the validation set. The hyperparameters that gave the best result were then chosen and used for the ANNs on the test sets.

A visualization of trained weights can be seen in figure 3.15. These weights would be acting like templates for the incoming test images.

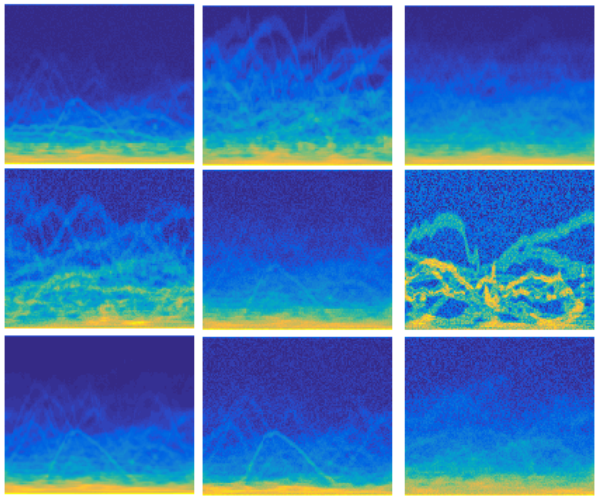


**Figure 3.15:** A visualization of a subset of the trained weights. What can be seen are features captured by the network after training. There are units that are clearly looking for "boxing-like" features (the bottom row), "running-like" features (top row middle) and "walk-like" features (middle right). These units would get activated upon input images matching their respective class. The bias unit is not visualized.

The network was implemented using MATLAB. The forward- and backpropagation code was written in vectorized format, avoiding loops due to the costly computation time in MATLAB. Despite this the network was not suitable for expansion. A larger depth would result in greatly increased training times.

Using 188 images the network took on average 602.38 seconds to train while testing with 41 images on average took 0.13 seconds. When using 378 images for training the average training time was 1141.84 seconds and using 81 images for testing took on average 0.284 seconds.

### 3.3.2 Convolutional Neural Networks

Images used for input to the CNNs were simply reshaped from 110x146 to 110x110. This was due to the convention of CNNs using square inputs. Like before the images were in gray-scale with the output classes being the same. The data set was split exactly the same way as for the ANN. In total two CNNs were designed. CNN One was designed for the STFT, CWT and SPWVD algorithms. The images from these three algorithms were similar. Images generated by the EMD looked very different and performed extremely poorly when tried on CNN One. Because of this CNN Two was designed specifically for the images from the EMD. The design parameters for both CNN One and CNN Two are presented later in this section.

The main tool used in the design and execution of the CNN was Google's *Tensorflow* (TF) in conjunction with Python v3.5.2. TF provides a *Application Programming Interface* (API) which included efficient convolutional operations as well as ReLUs, max-pool functions and other useful tools. It also allowed for GPU computation, which significantly sped up the training process. NVIDIA has developed support

for their more modern cards and the one used, the NVIDIA GTX 680, was the first generation to benefit from this support.

The Modified National Institute of Standards and Technology (MNIST) database was also used. It is a collection of hand written images and there is about 60 000 training images and 10 000 test images. The database is commonly used to compare algorithms, on a relatively simple problem, as state-of-the-art networks get around 99.8% accuracy.

This served as a sanity check and aided in the calibration of the networks. CNN One was able to achieve a test accuracy of 95.42 % which was not exceptional but it did prove that the network was not incorrectly designed. CNN Two achieved an accuracy of 98.52 % in the same test. The design was a process of trial and error with the networks constructed, tested using MNIST, trained and validated on the data of interest and modified to achieve better performance. Once a good accuracy was achieved on the validation sets the CNNs were fed the test sets.

### 3.3.3 CNN Design

The CNN design used had three convolutional layers with the first layer using two filters, the second layer using three filters and the third layer using three filters. Figure 2.12 from Section 2.4.2, which consists of two convolutional layers, is a useful tool to visualize this.

Recall that input images started with a size of $110x110x1$ pixels. Two 11x11x1 filters were then applied to a padded version of the input image so that two activation maps with sizes 110x110x1 were generated. Next the ReLU activation function was applied to introduce non-linearities in order to aid training. The image is pooled by a 2x2 max-pool operator with a stride of [2,2] (vertical and horizontal). The result are two downsampled images of size 55x55. Next three 5x5x2 filters convolve the padded image. ReLU and max-pooling (at same size and stride) are again applied, resulting in three 28x28x3 images. In a similar fashion the last layer outputs a 14x14x3 image. The fully connected layer flattens the image and multiplies it with a size of 8. In addition to this, the dropout layer randomly deactivates half of the neurons during the training of the network. Finally the output layer returns a 1x3 matrix as the output of the network.

The design parameters used for CNN One and CNN Two can be seen in 3.4. CNN One is a simpler network compared to CNN Two.
The number of neurons are calculated as follows,

$$N = whd \tag{3.2}$$

where $N$ is the total number of neurons in the layer, $w$, $h$, and $d$ are the width, height and depth of the activation map respectively. The parameters of the first filter are calculated as below,

$$F_p = (whc + 1)d \tag{3.3}$$

where $F_p$ is the number of parameters in the filter, $w$ and $h$ are the width and height, respectively, $c$ is the filter depth, the 1 is because of the bias unit and $d$ is

| Parameter | CNN One | CNN Two |
|---|---|---|
| Filter One Size | 11x11x1 | 11x11x1 |
| Filter One Depth | 2 | 5 |
| Filter One Stride | 1 | 1 |
| Pool One | 2x2 | 2x2 |
| Pool Stride One | [2,2] | [2,2] |
| | | |
| Filter Two Size | 5x5x2 | 5x5x2 |
| Filter Two Depth | 3 | 10 |
| Filter Two Stride | 1 | 1 |
| Pool Two | 2x2 | 2x2 |
| Pool Stride Two | [2,2] | [2,2] |
| | | |
| Filter Three Size | 3x3x3 | 3x3x3 |
| Filter Three Depth | 3 | 20 |
| Filter Three Stride | 1 | 1 |
| Pool Three | 2x2 | 2x2 |
| Pool Stride Three | [2,2] | [2,2] |
| | | |
| Input Nodes | 110x110x1 | 110x110x1 |
| FC Layer Size | 8 | 25 |
| Output Nodes | 3 | 3 |
| Batch Size | 30 | 30 |
| No. Iterations | 3500 | 3500 |
| Training Step | 1e-4 | 0.5e-4 |
| Dropout rate | 50% | 50% |

**Table 3.3:** Table of design parameters used for CNN One and CNN Two.

the number of filters.

Using 188 images CNN One took on average 65.62 seconds to train while testing with 41 images on average took 1.09 seconds. When using 378 images for training the average training time was 122.82 seconds and using 81 images for testing took on average 1.96 seconds.

Using 188 images CNN Two took on average 106.21 seconds to train while testing with 41 images on average took 1.32 seconds. When using 378 images for training the average training time was 300.52 seconds and using 81 images for testing took on average 2.46 seconds.

| Parameter | CNN One | CNN Two |
|---|---|---|
| Layer 1 Neurons | 6050 | 15125 |
| Layer 2 Neurons | 2352 | 7840 |
| Layer 3 Neurons | 588 | 3920 |
| Layer FC Neurons | 8 | 25 |
| Total Neurons | 8998 | 26910 |
| | | |
| Layer 1 Parameters | 244 | 610 |
| Layer 2 Parameters | 153 | 510 |
| Layer 3 Parameters | 84 | 560 |
| Layer FC Parameters | 4712 | 98025 |
| Total Parameters | 5193 | 99705 |

**Table 3.4:** Table of comparison between the neurons and parameters for CNN One and CNN Two.

# 4
# Results

This section presents the performance of the ANN and CNN in combination with the STFT, CWT, SPWVD and EMD algorithms. The computation times for the DE algorithms.

## 4.1 DE algorithm computation times

The average of the computation times across all different categories for the different algorithms are presented in table 4.1. The column marked Case A represent computation time given observations that are one second long and the column marked Case B represents computation time given observations that are half a second long. This means Case A uses twice as many data points per calculation as Case B.

Not surprisingly computation times for Case A are longer than for Case B. For Case B the EMD has the best performance by far at 0.005 seconds. The SPWVD takes significantly longer to run, Case B takes 3.585 seconds to process. For Case A the best performance is again with the EMD at 0.110 seconds and the SPWVD is the slowest by far at 7.098 seconds.

| Algorithm | Case A computation time [s] | Case B computation time [s] |
|---|---|---|
| STFT | 0.441 | 0.280 |
| CWT | 1.398 | 0.785 |
| SPWVD | 7.098 | 3.585 |
| EMD | 0.110 | 0.050 |

**Table 4.1:** Table of computation times for the STFT, CWT, SPWVD and EMD algorithms for both half second (Case A) and one second (Case B) observation times.

## 4.2 ANN performance

The results using noiseless images as input in conjunction with ANNs can be seen in table 4.2. With the exception of the EMD, there is no difference in performance for any DE-algorithm when comparing Cases 1 and 2. Performance does however improve for both the STFT and SPWVD algorithms while the CWT algorithm performed perfectly in all cases. 100% accuracy was unexpected and would certainly drop somewhat with a larger data set. It is also notable that there is no gain in accuracy when comparing Case 1 and Case 2. This indicates that half a second

of observation is sufficient for the $\mu$D-signatures to be classified correctly. EMD performance was on par with pure guessing for all Cases.

| Algorithm | Accuracy Case 1 | Accuracy Case 2 | Accuracy Case 3 |
|---|---|---|---|
| STFT | 87.18 % | 87.18 % | 98.72 % |
| CWT | 100 % | 100 % | 100 % |
| SPWVD | 84.61 % | 84.61 % | 94.87 % |
| EMD | 33.33 % | 33.33 % | 33.33 % |

**Table 4.2:** Table of ANN performance for different Cases and DE algorithms.

The results using noisy images as input can be seen in table 4.3 where the STFT algorithm performs much worse for all Cases compared to the noiseless scenario. The STFT accuracy peaks for case three at 80.77% while the SPWVD peaks at 89.74% meaning the SPWVD performs better than the STFT when noise is added. CWT performs the best with a 100% accuracy yet again. With noisy input performance does get worse from Case 2 to 1. This is expected as one second snapshots contain more information than half second snapshots. The EMD performance for Cases 1 and 2 at 33% was on par with pure guessing and for Case 3 it was even worse at 32.5%.

| Algorithm | Accuracy Case 1 | Accuracy Case 2 | Accuracy Case 3 |
|---|---|---|---|
| STFT | 76.92 % | 69.23 % | 80.77 % |
| CWT | 100 % | 100 % | 100 % |
| SPWVD | 87.18% | 84.61 % | 89.74 % |
| EMD | 33.33 % | 33.33 % | 32.5% |

**Table 4.3:** Table of ANN performance for different Cases and DE algorithms when noise has been added.

## 4.3 CNN performance

The results using noiseless images as input in conjunction with CNNs can be seen in table 4.4. For all DE algorithms one second observations still outperform or match the performance of half second observations when the same amount of images are used. This means there is a significant gain in performance with increased observation time. CWT performs best for Case 1 but performs poorly for Case 3 where SPWVD and STFT have the same performance at 97.44 % accuracy. The benefits of more training data seems to outweigh the loss of information when reducing observation time from one second in case one to half a second in case three. EMD performance was not best for any of the Cases but performed on par with CWT for Case 2.
The results using noisy images as input in conjunction with CNNs can be seen in table 4.5. Again Case 1 outperforms Case 2 with the largest difference being found for the CWT with 97.44 % compared to 83.33 %. Performance decreases for case

| Algorithm | Accuracy Case 1 | Accuracy Case 2 | Accuracy Case 3 |
|---|---|---|---|
| STFT | 94.87 % | 94.87 % | 97.44 % |
| CWT | 97.79 % | 87.18 % | 84.61 % |
| SPWVD | 94.87 % | 92.31 % | 97.44 % |
| EMD | 92.31 % | 87.18 % | 88.75 % |

**Table 4.4:** Table of CNN performance for different cases and DE algorithms with noiseless input.

three where the SPWVD barely slightly outperforms both the STFT and CWT algorithms. Yet again the results indicate that one second observations are superior to half second observations. With noisy input to the CNNs there was not a gain in case three with more images available. EMD has the best performance of all possible combinations, seen in Case 3, at 97.50% but also the worst performances for Cases 1 and 2.

| Algorithm | Accuracy Case 1 | Accuracy Case 2 | Accuracy Case 3 |
|---|---|---|---|
| STFT | 92.31 % | 87.18 % | 87.18 % |
| CWT | 97.44 % | 83.33 % | 87.18 % |
| SPWVD | 92.31 % | 82.05 % | 89.74 % |
| EMD | 89.74 % | 79.49 % | 97.50 % |

**Table 4.5:** Table of CNN performance for different cases and DE algorithms when noise has been added.

# 5
# Discussion

In this section data, simulation, choices regarding DE and ML algorithms as well as results are discussed.

## 5.1 Limitations with MOCAP data and simulated radar returns

As mentioned in Chapter 3, the original plan was to use data from real-world measurements. These contained subjects walking, jogging, running, biking and multiple persons walking next to each other. When it became clear that the gathered data would be unusable the choice was made to turn to another solution. The best alternative was the MOCAP library but it still came with a number of limitations. There were only three classes with an acceptable balance between subjects and repetitions and there was obviously no way of controlling what the recordings contained in terms of duration, distance or angle. Without a doubt the greatest limitation was the number of repetitions with the "Running" category setting the limit to 90/180 high quality images with distinct $\mu$D signatures if a balance between the number of images from each category was to be achieved. In contrast to this there were over 300/600 images available from the "Boxing" category and around 250/500 images from the "Walking" category. This could not be addressed given the fixed data set but it is a weakness that undoubtedly impacted the performance of all ML algorithms. Preferably there would've been thousands or even millions of available recordings of multiple subjects performing several common VRU activities. It should also be noted that the universe recorded by the MOCAP only contained the 41 points meaning all of the environment became invisible.

The choice to use constant cross-sections shaped as spheres was made to simplify the problem. It would have been more realistic to shape them as cylinders with cross-section depending on the angle between the radar and the limbs but that sort of model would have required significantly more work to implement.

The simulation model of the radar returns also had some limitations. It was only possible to simulate a single channel so the subject could only be observed from one angle. It also only allowed for a linear frequency modulation meaning only a sawtooth waveform was possible. The model did however do a good job in simulating what it set out to and $\mu$D signatures generated with for example the STFT looked similar to real radar returns seen in for example [9][32]. The model only generated the range bins at peak power output along slow-time and not an entire radar data

cube but as there was no appropriate tool for generating an entire environment with both subjects and objects this was a non-issue.

## 5.2 Choice of window functions and mother wavelet

How well different DE algorithms performed depended heavily on user choices. Window functions had to be chosen for STFT and SPWVD. Both shape and window length had a large impact. For the STFT this was relatively straightforward with a balance between time and frequency resolution revealing itself rather quickly. For the SPWVD this was not the case. As there were two window functions with their own lengths it was very difficult to find a balance and the process was one of exhaustive trial and error. Choice of kernel function also impacted performance and the particular setup used for this thesis aimed to optimize time-frequency representation by minimizing potential disturbing cross-terms.

If time-frequency representation had been the primary concern SPWVD would without a doubt be the most powerful algorithm of the three. This is illustrated with figure 5.1 which is a spectrogram of subject 16 take 8 which was previously used as an example in the method section. Resolution is very good in both domains but generating this image required window functions of length 171. This means it took over 15 seconds to run the algorithm, more than twice that of the average time for the corresponding SPWVD images used this thesis. The constraint on computation time restricted the quality of image possible but should there be a leap in computation power it could be a strong candidate. It could also be useful for other applications where the limitation to time is not a factor.

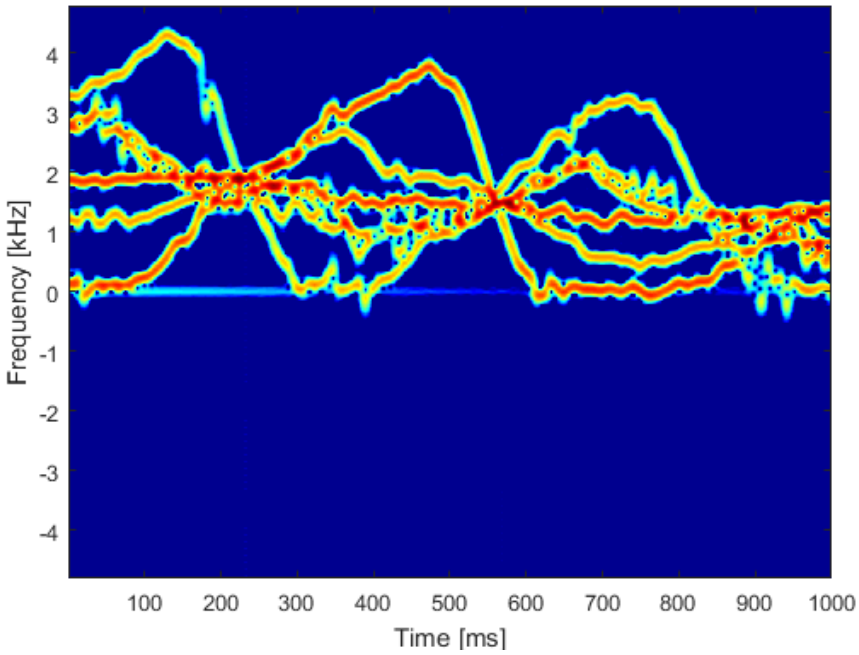



**Figure 5.1:** The same spectrogram as above but using 171 length windows instead. The image shows how good resolution is possible with the method if computation time constraints are less of a concern.

In the case of the CWT the Morlet mother wavelet was used. It was chosen based on success in previous publications and it did perform very well in combination with the ANN. An accuracy of 100 % is likely a bit optimistic and with a larger data set it would likely decrease but the reader should be aware that efforts were made to "pressure" the ANN in the sense that several different data configurations were attempted. Images were moved around between the testing and training sets to make sure there were no mistakes, repetitions or copies. After a lot of effort it was found that the CWT in conjunction with the ANN always predicted the three classes correctly. It may be due to the use of the Morlet wavelet. The shape matches very well with the $\mu$D signatures generated by these three classes allowing for such a strong performance. It would likely be less suitable for other common activities such as biking, rollerskating or skateboarding where $\mu$D signatures would be a lot less sinusoidal. Perfect performance was not achieved when combining CWT and CNN and the reason for this is unknown. It is possible performance could have been better if a CNN had been designed and implemented specifically for the CWT. As the task of designing and implementation required extensive work it was not done for the CWT as opposed to for the EMD. This is because the EMD performed worse than the CWT.

## 5.3 Choices regarding EMD

The EMD algorithm was simple to given a complex input signal but the output from it was trickier. There were a few alternative methods to generalize the EMD algorithm to the complex domain. It was determined that the bivariate EMD performed the best when the signal was reconstructed. Since the signal was broken down into multiple IMFs, each a complex vector with the same length as the original signal, the information contained in the signal was "inflated", spread over a larger number of sample points. Since the size of input to the neural networks was very limited this caused some problems.

The choice to discard the real part of these IMFs obviously led to loss of information. It did not seem to be that crucial to include both real and imaginary as the reconstruction from either real or imaginary channel produced a satisfying result confirmed by the spectrogram. Another form of reducing the outgoing information from the EMD was to downsample the IMFs but that had a significant impact on the reconstruction and performance. Even worse, in the cases where signals had to many IMFs for them to fit inside the allocated neural network input size they had to be completely ignored meaning low frequency components were completely lost.

For EMD to show its true potential a larger neural network, designed with EMD in mind, would be necessary. There is also a case where the EMD would complete its breakdown of the signal and there would not be enough IMF functions, and as such the input to the neural network had to be padded with zeros. This effectively was a waste of input where useful information could have been injected.

The EMD algorithm has a very neat way of extracting the fundamental time-varying frequencies from the signal. The IMFs containing these components can be manipulated in several ways possibly producing many different inputs to be fed into the neural networks. Comparing to the other DE methods, which did not allow for separation of the output data, the EMD allows for much more freedom. For example, images need not be fed into the neural network but matrices of the IMFs can be used as an input instead.

## 5.4 Regarding noise

The noise that was used was not successfully quantified but instead chosen based on visual inspection. It would have been desirable to be able to set a SNR and then get the images with a correct noise level. This couldn't be done however, and because of the large amount of time it took to select good images to serve as ML inputs, multiple noise levels were not experimented with. A more structured approach would have been desirable.

## 5.5 Computation times - C vs MATLAB

STFT, CWT and SPWVD were all run in MATLAB using MATLAB functions meaning they had comparable results. The functions were not well optimized and

ran slowly compared to the EMD. Part of the reason the EMD ran so fast was because it was implemented in C. When run in MATLAB, EMD took on average 1.1 seconds to run. This is very comparable to both STFT and CWT. An acceptable time frame for the entire tool chain is likely around 0.25 seconds meaning DE and classification must be done within that duration. Only the EMD, implemented in C, falls within that interval but it is very probable that both CWT and STFT would run at sufficiently fast speeds if implemented in a more efficient manner.

## 5.6 Comments on classification

ANNs and CNNs were used with primarily images as input, the exception being the ANN combined with the EMD. Not translating the output matrices into images was possibly a mistake as there was a huge difference in performance between the CNN and the ANN that took EMD input. To investigate if the issue was the matrices the images made for the CNN was used as input to a slightly modified version of the ANN but the performance was still no better than guessing. A possible explanation for the bad results of the ANN is that it was not deep enough with only one layer. The original CNN, CNN One, also performed poorly, it classified correctly around 75 % of the time, and so CNN Two was designed. CNN Two was significantly larger than CNN One.

The reason grey scaled images were used was that upon network construction it was a concern that it would train and test slowly though this turned out to not be a problem. The assessment was that no useful information was contained in the colours and that removing them would have no real impact on performance. On grey-scaled images with noise added it was harder to distinguish different classes for the human eye than in the colored case but this should not have impacted the ML algorithms.

The fact that the ANN in some cases performed better than the CNN can almost certainly be attributed to the size of the data set. As mentioned previously in the report CNNs are commonly trained and tested on massive data sets and the one used in this case is orders of magnitude smaller. ANNs can on the other hand perform well even on small sets and so it is no surprise that it did just that.

The tests conducted with the MNIST database prove that the CNN was not poorly constructed and the larger networks that were originally designed performed even better but proved to be unusable with so little data available. Therefore no major conclusions should be drawn from the relatively poor performance of the CNNs. It is plausible that a larger data set would allow a CNN to outperform the ANNs and be a strong contender.

# 6
# Conclusion

In this work MOCAP data was used together with a model for simulated radar response in order to generate data containing $\mu$D signatures of human activities, suitable for time-frequency analysis and classification. STFT, CWT, WVD and EMD algorithms were run on the generated data and the output from those algorithms was then classified using the ANN and CNN algorithms according to the categories "Walking", "Running" and "Boxing".

The most notable finds was the perfect classification accuracy of CWT in combination with a ANN, even in the presence of noise, as well as the great potential shown by EMD in combination with a CNN which achieved a classification accuracy of 97.50% on noisy signals. The EMD performance was that high despite downsampling but it was also found to perform extremely poorly when combined with a small network such as the ANN. Classification accuracies were at 33% or less meaning they were worse than guessing indicating that larger networks may be required when classifying with EMD.

The SPWVD performed well on noiseless data regardless of which neural network was used but was found to be unsuitable as a DE algorithm for automotive use due to excessively long computation times. The shortest computation time, given half a second of observation time, was EMD with an average of 0.05 seconds, owing to its implementation in C. This should be compared to its implementation in MATLAB, averaging at 1.1 seconds. The STFT, implemented in MATLAB, was on second place with 0.280 seconds for the same scenario. With the implementations used only EMD had an acceptable computation time. For the classification algorithms the running time for the ANN was 0.024 seconds and for the CNN was 0.030 seconds. These are by well within the appropriate ranges for the application. Paired with an EMD the total running time to classify a target would theoretically only take 0.075 seconds, much faster than the time it would take for a human to do the same task. Considering this work was done mostly in MATLAB and C, further optimization can be done on dedicated hardware which could further bring down the computational time.

## 6.1 Future work

The single most important thing to do in the future is to test combinations of these algorithms on a large real data set. With simulations results can appear promising but since this thesis was geared towards potential real world applications it must be verified with a non-simulated radar signal in a realistic environment, such as a

busy intersection. Additionally, due to the limited computation power available on a vehicle's ECU any kind of solution must be modified and optimized in order to have a suitable running time in a real-time scenario.

A large data set must be gathered in order to make any kind of ML algorithms usable. Recordings must be made on the order of 100-1000 times the current dataset. Other common VRU activities such as biking, skateboarding or pushing a stroller were not evaluated and must be studied. No tests were conducted with multiple subjects at the same time which is also very important.

Specifically the different configurations of the algorithms can be further explored, such as using the EMD in a deeper network with more data. The EMD is also very versatile, allowing for many different possible input formats. The EMD had a fast run-time, proved to be very resilient to noise and strongly benefited from having more data. These are all great qualities for the automotive setting. A suitable data format used in conjunction with a ML algorithm should be investigated as it is not obvious that image is the best choice for exploiting all the information in the EMD output.
There are more combinations of ML algorithms along with newly developed time-frequency algorithms that can possibly give greater classification and computational performance in the future.

# Bibliography


[1] Galati, Gaspare. '100 years of radar', p. 296, 2015, Springer

[2] Chen, Victor C. and Lipps, Ronald D. 'Time frequency signatures of micro-Doppler phenomenon for feature extraction', Proc. Society of Photo-Optical Instrumentation Engineers, 2000, volume 4056, p. 220-226

[3] Rodriguez-Hervas, Berta and Maile, Michael and Flores, Benjamin C. 'Comparative of signal processing techniques for micro-Doppler signature extraction with automotive radar systems', 2014, *Proc. SPIE 9077, Radar Sensor Technology XVIII*

[4] Yang, Yinan and Lei, Jiajin and Zhang, Wenxue and Lu, Chao. 'Target Classification and Pattern Recognition Using Micro-Doppler Radar Signatures', 2006, IEEE, volume 2006, p. 213-217

[5] Wang,Yan and Wu,Xi and Li,Wenzao and Li,Zhi and Zhang,Yi and Zhou,Jiliu. 'Analysis of micro-Doppler signatures of vibration targets using EMD and SP-WVD', *Neurocomputing*, 2015, Volume 171, Pages 48–56

[6] Louis, A. K. and Maass, P. and Rieder, A., 'Wavelets: theory and applications', 1997, p. 2-3

[7] Smith,Graeme E. and Woodbridge,Karl and Baker,Chris J.,'Radar Micro-Doppler Signature Classification using Dynamic Time Warping', *IEEE Transactions on Aerospace and Electronic Systems*, 2010, volume 46, p. 1078-1096.

[8] Kim,Youngwook and Ling,Hao. 'Human activity classification based on micro-Doppler signatures using an artificial neural network', *Antennas and Propagation Society International Symposium*, 2008

[9] Kim,Youngwook and Ling,Hao. 'Human Activity Classification Based on Micro-Doppler Signatures Using a Support Vector Machine', *IEEE Transactions on Geoscience and Remote Sensing*, 2009, volume 47 p. 1328-1337

[10] Kim,Youngwook and Moon,Taesup. 'Human Detection and Activity Classification Based on Micro-Doppler Signatures Using Deep Convolutional Neural Networks', *2016, IEEE Geoscience and Remote Sensing Letters*, volume 13, p. 8-12

[11] CMU.edu, CMU Graphics Lab Motion Capture Database. Available: http://mocap.cs.cmu.edu/ [Accessed: 8- May- 2017].

[12] Gurbuz,Sevgi Z. and Melvin, William L. and Williams, Douglas B., 'A Nonlinear-Phase Model-Based Human Detector for Radar', *IEEE Transactions on Aerospace and Electronic Systems*, 2011, volume 47, p. 2502-2505

[13] Boulic, Ronan and Thalmann, Nadia M. and Thalmann, Daniel. 'A global human walking model with real-time kinematic personification', *The Visual Computer, 1990*, volume 6, p. 344-358

[14] Erol,Baris and Karabacak,Cesur and Gurbuz,Sevgi Z. and Gurbuz,Ali C. 'Simulation of human micro-Doppler signatures with Kinect sensor', *Radar Conference, 2014 IEEE*, Cincinnati, OH, USA, p. 0863-0868.
[15] http://tftb.nongnu.org/. the Time-Frequency Toolbox. Available: http://ftp.acc.umu.se/mirror/gnu.org/savannah/tftb/ [Accessed: 2- March- 2017].
[16] D. Gabor. 'Theory of Communication', 1946, J. lEE London
[17] Chen,Victor C. and Tahmoush, David and Miceli, William J. 'Radar micro-Doppler signatures: processing and applications', 2014, volume 34, p. 103-104
[18] Chen,Victor C. and Tahmoush,David and Miceli,William J. 'Radar micro-Doppler signatures: processing and applications', Institution of Engineering and Technology, 2014, volume 34
[19] Wang,Ran and Jiang,Yi-Cheng. 'ISAR Ship Imaging Based on Reassigned Smoothed Pseudo Wigner-Ville Distribution', 2010 p. 2
[20] Huang, Norden E and Shen, Zheng and Long, Steven R and Wu, Manli C and Shih, Hsing H and Zheng, Quanan and Yen, Nai-Chyuan and Tung, Chi Chao and Liu, Henry H. 'The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis', 1998, volume 454, p. 903-995
[21] Donghoh Kim, and Hee-Seok Oh. 'EMD: A Package for Empirical Mode Decomposition and Hilbert Spectrum', The R Journal Vol. 1/1, May 2009
[22] Mitchell,Tom M., 'Machine learning', 1997, p. 2
[23] Andrew Ng, 'CS294A - Research Project in Artificial Intelligence Lecture notes', Stanford University, 2011.
[24] Fei-Fei Li, 'Convolutional Neural Networks', Stanford University, 2016
[25] Krizhevsky,Alex and Sutskever,Ilya and Hinton,Geoffrey. 'ImageNet classification with deep convolutional neural networks', *Communications of the ACM, 2017*, volume 60, p. 84-90
[26] Liao,Zhibin and Carneiro,Gustavo.'Competitive Multi-scale Convolution', 2015, p. 1-9
[27] Russakovsky, Olga and Deng, Jia and Su, Hao and Krause, Jonathan and Satheesh, Sanjeev and Ma, Sean and Huang, Zhiheng and Karpathy, Andrej and Khosla, Aditya and Bernstein, Michael. 'Imagenet large scale visual recognition challenge', 2015, volume 115, p. 211-252
[28] Andrej Karpathy, 'CS231n Convolutional Neural Networks for Visual Recognition', 2015. [Online]. Available: http://cs231n.github.io/optimization-2/. [Accessed: 23- July- 2017].
[29] Chen,Victor C. and Tahmoush, David and Miceli, William J. 'Radar micro-Doppler signatures: processing and applications', 2014, volume 34, p. 43-45
[30] Ayad,Mouloud and Chikouche,Djamel and Boukazzoula,Nacereddine and Rezki,Mohamed. 'Search of a robust defect signature in gear systems across adaptive Morlet wavelet of vibration signals', 2014
[31] Chen, Victor C. 'The micro-doppler effect in radar', 2011, p. 209-245
[32] Narayanan,Ram M. and Zenaldin,Matthew. 'Radar micro-Doppler signatures of various human activities',*IET Radar, Sonar Navigation* 2015, volume 9, pages 1205-1215