NTNU

Norwegian University of
Science and Technology

Deep Learning for the Classification of EEG Time-Frequency Representations

Audun Eltvik

Master of Science in Cybernetics and Robotics
Submission date: July 2018
Supervisor: Marta Maria Cabrera Molinas, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This thesis is a report on work that was carried out in the spring of 2018 in the application of deep learning in time-frequency representations of EEG data, with a view towards BCI applications. This work extends on the work done by me and Olav Ljosland during our specialization project in the autumn of 2017.

As part of this work, the Hilbert-Huang transform was implemented in Python. This, together with implementations of the short-time Fourier transform and the continuous wavelet transform, both implemented in the SciPy library, was applied to create time-frequency representations of signals.

Time-frequency representations were generated both for a synthetic dataset created by me and a publicly available dataset of labeled EEG recordings.

Four different convolutional neural network models were designed using the Keras library and run using Tensorflow in an attempt to classify the time-frequency representations. The results of using different time-frequency representations was compared. The results were also compared to previous attempts to classify the same data to evaluate the method as a whole.

The matplotlib library was used to create plots and other figures for this thesis.

Trondheim, July 2, 2018

Audun Eltvik

Acknowledgment

This thesis marks the end of my Masters degree program in Cybernetics and Robotics at NTNU. I would like to thank all the lecturers and staff at NTNU for the excellent education I have received here. In particular, I would like to thank Professor Marta Molinas for the invaluable guidance she has provided as my supervisor during my specialization project and Master thesis.

I would also like to thank my friends and family for their support and encouragement throughout my time at NTNU.

Abstract

This thesis is a report on the implementation and evaluation of a new method classifying EEG signals. The method involves applying either the Short-time Fourier Transform (STFT), Continuous Wavelet Transform (CWT) or Hilbert-Huang Transform (HHT) to produce a two-dimensional time-frequency representation of the signal, known as spectrograms, scalograms and Hilbert spectra, respectively. These two-dimensional representations are then classified using a Convolutional Neural Network (CNN).

The evaluation of this method involved testing it on two different datasets. The first is a synthetic dataset generated by simulating a non-stationary and noisy process, and contains signals belonging to one out of three classes. The second dataset consists of real EEG data available as part of one of the tasks in the BCI Competition III. It contains 1,400 EEG recordings consisting of 3.5 seconds of the subject imagining movement in either the right hand or in the left food, and the task is to decide which of these the subject was imagining during the recording.

Four different CNN architectures were evaluated using k-fold cross-validation with each of the three representations. When evaluated using the spectrogram and Hilbert spectrum representation of the synthetic data, the best classification accuracy was 98.3% and 88.19%, respectively. The scalogram representation fared less well, with 59.29% as the highest achieved classification accuracy.

Results were similar when evaluating the classifiers on real data. The highest accuracy achieved when classifying EEG spectrograms was 72.50%. For Hilbert spectra it was 58.00% and for scalograms it was 55.93%.

Sammendrag

Denne masteroppgaven er en rapport av implementasjonen og evalueringen av en ny måte å klassifisere EEG data. Denne metoden går ut på å anvende enten STFT, CWT eller HHT for å produsere en todimensjonal representasjon av signalet, respektivt omtalt som spektrogrammer, skalogrammer og Hilbertspektra. Disse todimensjonale representasjonene blir så klassifisert ved bruk av et konvolusjonelt nevralt nettverk (CNN).

Evalueringen av denne metoden involverte å teste den på to forskjellige dataset. The første er et syntetisk dataset generert av å simulerte en ikkestasjonær og strøyete prosess, og inneholder 10.000 signaler som er del av en av tre klasser. Det andre datasettet består av ekte EEG data og er tilgjengelig som del av BCI Competition III. Det inneholder 1.400 EEG innspillinger op 3,5 sekunder bestående hvor deltakeren innbiller seg at han beveger enten høyre hånd eller venstre fot under innspillingen.

Fire forskjellige CNN arkitetkturer ble vurdert med k-fold kryssvalidering med hver av de tre representasjonene. Når de ble evaluert ved bruk av spektrogram- og Hilbertspektrumrepresentasjoner av den syntetiske dataen, oppnådde de en treffsikkerhet på respektivt 98,3% og 88,19%. Skalogramrepresentasjonen kom dårligere ut, med 59,29% som den høyeste oppnådde treffsikkerheten.

Resultatene ved å evaluere klassifikatorene på ekte data var liknende. Høyeste oppnådde treffsikkerhet ved klassifisering av EEG spektrogrammer was 72,50%. For Hilbertspektra var den 58.00% og for skalogrammer 55.93%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Brain Computer Interface (BCI) is the field of research concerned with developing technology that can allow direct communication between the brain and some device, without the use of the brains usual neuromuscular pathways. BCI requires some neuroimaging technique of which the most popular is electroencephalography (EEG). This technique involves placing a number of electrodes on the scalp of the subject to measure changes in the electrical field due to brain activity.

EEG technology has traditionally been expensive and cumbersome, and as such most BCI research has largely been restricted to academic research and biomedical applications for those with severe motor disabilities. Recent advances in hardware has, however, led to the development of portable and affordable EEG equipment. This has spurred research into applications within fields such as gaming and education[14].

On the software side, the greatest challenge in BCI is accurately mapping brain activity to a particular command. This problem can be reduced to classifying a given EEG reading as belonging to one of several classes, each corresponding to some activity in the brain. A common form of mental activity for BCI research is motor imagery, which involves a subject being asked to imagine a particular motor activity, and recording the resulting EEG activity. The challenge is to recognize the activity which is being imagined by the subject based solely on the EEG reading during the motor imagery. The particular problem chosen for this project involves attempting

to classify a 3.5 second EEG reading as corresponding to imagined movement of either the right hand or the left foot.

EEG generally gives rise to highly non-stationary and nonlinear signals[34]. Such signals are commonly analyzed using time-frequency analysis, where the signal is represented as a two-dimensional image showing time along one axis and frequency along the other. In this paper, the three techniques Short-time Fourier Transform (STFT), Continuous Wavelet Transform (CWT) and Hilbert-Huang Transform (HHT) are applied to create three different time-frequency representations. These methods are described in detail in section 3.

It is very difficult to manually design a method for differentiating between EEG segments. Usually, this is done using machine learning. In machine learning, rather than designing a task-specific solution to the problem, data is used to "learn" a method of classification.

When designing some machine learning system for EEG segments, one of the challenges involves extracting the features that are most important to the classification. *Deep learning* is the subfield of machine learning dedicated to algorithms that not only learn how to classify an object, but how to extract the correct features for the classification problem. One of the most exiting developments within deep learning in recent years is the great success in applying Convolutional Neural Networks (CNNs) in the field of image classification[19]. In this thesis, CNNs are applied to the problem of EEG classification by treating two-dimensional representations of the signal as images, and attempting to classify those.

The following research questions (RQs) are explored:

1. Can CNNs be used to classify time-frequency representations of signals?
2. Are CNNs applied on time-frequency representations of EEG viable? (i.e. do they outperform random guessing)
3. If so, which representation of the signal yield the best results?
4. Is this method competitive in comparison to established methods?

RQ1 is explored by looking at a simpler case than classifying EEG: generating synthetic time-frequency representations of three different classes and attempting to fit a CNN classifier to this

data. To answer the other research questions, the same method is applied to real EEG data, data set IVa from BCI Competition III. This data is first processed using Python to create time-frequency representations, which are then used as training data for four different architectures of CNNs. The results are compared to previous work in attempting to classify the same data.

This thesis consists of four parts. Part I presents relevant background theory. Part II presents the practical solutions for the work done for this thesis. In part III the results of this work is presented and discussed. Finally, part IV provides a summary the results, conclusions in regards to the research questions above and a discussion on possible further work.

Part I

Theory and background literature

Chapter 2

BCI and EEG

The goal of this thesis is to develop a new method of classifying electroencephalography (EEG) segments, with a view towards Brain Computer Interface (BCI) applications. As such, some background knowledge on BCI and EEG is necessary.

2.1 Brain-Computer Interfaces

The way most devices are operated is that the brain sends signals to the muscles and the muscles operate the device. The goal of BCI research is to develop some way of bypassing the motor system entirely and control devices using only direct measurements of neural activity. A BCI system must have some method of acquiring signals that contain information about the activity of the brain, it must analyze these signals to translate them into command inputs for the device and finally transmit these commands to the device.

Data acquisition methods are divided into invasive and noninvasive methods. Invasive methods involve surgically implanting sensors into the brain of the subject, whereas noninvasive methods involve some detachable sensors that are placed on the subjects head. Noninvasive methods have the obvious advantage of safety and reversibility: most subjects would not risk brain surgery for simple research or basic applications. Invasive BCI does have the advantage that the signals gathered are typically of a significantly higher quality. Semi-invasive BCI is something

of a compromise: it involves implanting sensors within the skull, but they are placed on top of, rather than within, the brain and therefore carry significantly lower clinical risk.

The actual signals that are aquired vary, but EEG is by far the most common type of signal. Other neuroimaging techniques that may be used include magnetoencephalography (MEG) and electrocorticography (ECoG), but these are not used in this thesis.

Once the signals have been acquired, some data processing algorithm must be applied to translate the raw signals into commands that can be understood by the device. Usually, the data is preprocessed in some way to extract a number of features, such as statistical momenta. A machine learning algorithm is then applied to the extracted features so that the BCI system learns to map a particular signal to a particular command.

Finally, the BCI system requires some way of communicating these commands to the output device. As any wired or wireless method of communication between the system and the output device can be used, this is generally not considered a core part BCI research, but it is a necessary consideration in any practical BCI application.

2.2 History

While not the first to study electrical activity in the brain, EEG research is generally seen as beginning with Hans Berger, a German professor of psychiatry active in the first half of the twentieth century. In his seminal 1929 paper[4], he introduced a method of studying the brain that involved placing electrodes on the scalp of the subject and measuring and amplifying the electrical field. He named the resulting plots *electroencephalograms*. The first known EEG recording, by Berger, is shown in figure 2.1.

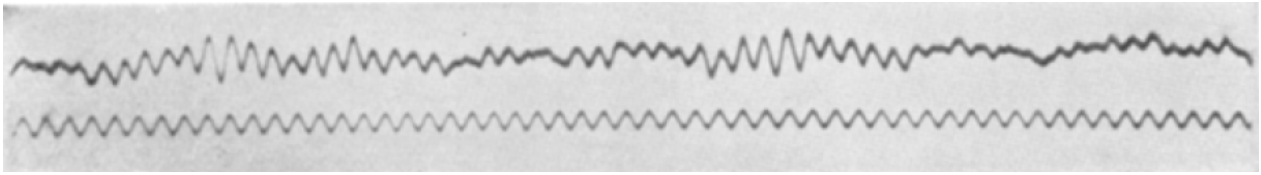

Figure 2.1: The first human EEG recording obtained by Hans Berger in 1924. The upper tracing is EEG, and the lower is a 10 Hz timing signal.

While Bergers paper made little immediate impact, similar work by Edgar Douglas Adrian[1] and others pushed EEG to the forefront of neuroscience. The technology was quickly applied to medical problems such as the detection of seizures[41].

The term BCI was first coined by J. J. Vidal in a 1973 paper[59], as a description of a system that uses EEG for the control of objects. The first practical BCI system was created by Vidal in 1977: EEG control of a cursor-like object on a computer screen[58].

Historically, EEG equipment has been expensive and cumbersome, and as such unviable for commercial applications. Recent advances in hardware has lead to commercially available EEG equipment. Thanks to this development, there is renwed interest in BCI research in areas outside of biomedical applications[14].

2.3 Electromagnetic Fields in the Brain

The reason EEG is able to capture neural activity lies in how the brain processes information. The brain consists of a complex network of around 86 billion neurons[22]. The way these neurons receive, process and transmit information is through electrical signals.

Each neuron is a cell rather like all other cells in the human body. The crucial difference is that, in addition to the normal organelles like mitochondria and the cell nucleus, neurons connect to one another through connections to other neurons. Each neuron has a number of dendrites, from which electrical signals arrive, and an axon, where the electrical signals leave the neuron. An illustration of a neuron is shown in figure 2.2.

The signals sent and received by neurons follow a particular pattern. At rest, there is a negative electrical potential across the cell membrane of the neuron, known as the *resting potential*. As the neuron is stimulated by electrical signals from connected neurons, the membrane potential increases. Should the membrane potential increase above a certain threshold, the membrane potential will very rapidly increase to a significantly higher value, and then just as rapidly decrease and stabilize at the resting potential. This process is known as the *action potential*, and is illustrated in figure 2.3.

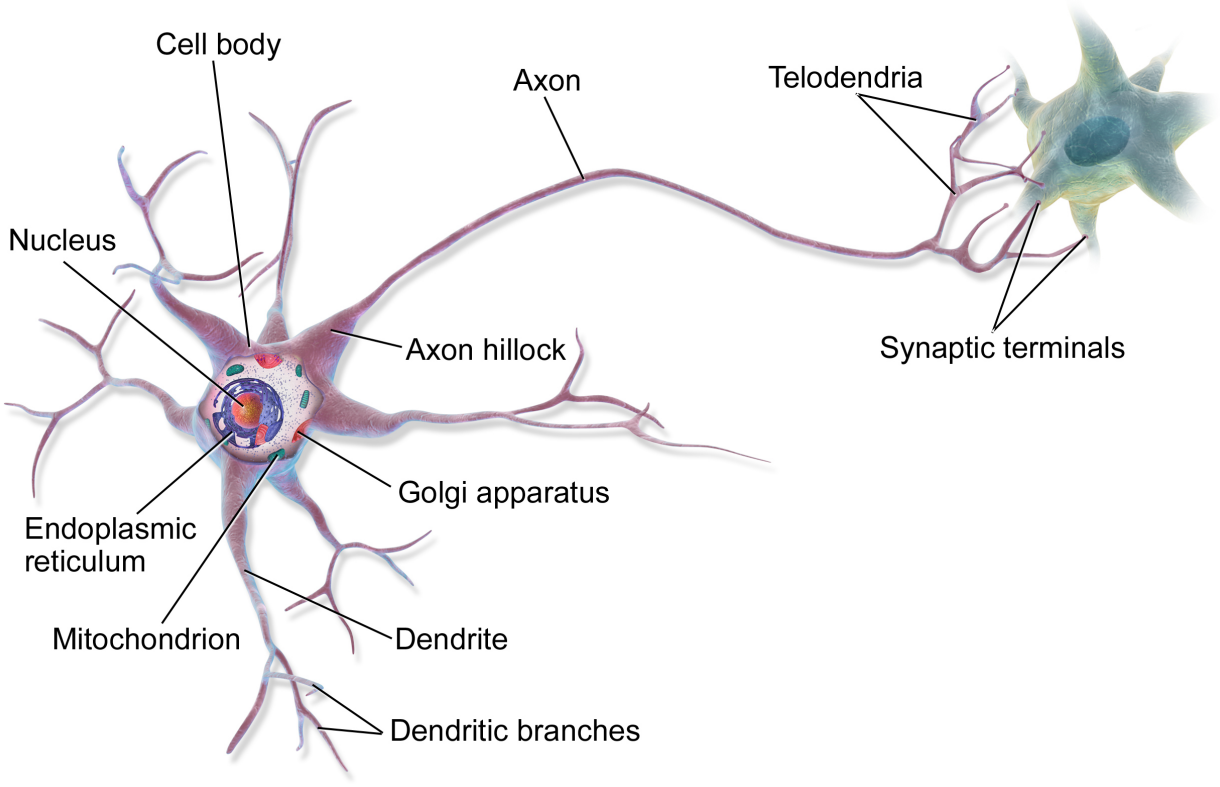

Figure 2.2: Example of neuron

Once the action potential fires in a neuron, it is propagated through the axon to connected neurons, potentially causing an action potential to fire in those neurons as well. This is how information in biological neural networks is propagated and processed. An interesting observation is that the action potential is an all-or-nothing response: either the action potential fires or no information is sent across the axon. Thus, information in the brain is represented in the *frequency* of action potentials, rather than the strength of electrical signals.

An additional source of electrical activity in the brain are the Postsynaptic Potentials (PSPs). These are changes in the membrane potential caused by molecules binding to receptors on the neuron. Depending on the type of receptor the molecule binds to, it may cause the resting membrane potential to increase (depolarize), in the case of Excitatory Postsynaptic Potentials (EPSPs), or decrease (hyperpolarize), in the case of Inhibitory Postsynaptic Potentials (IPSPs). In the case of an EPSP the membrane potential is moved closer to the action potential threshold, and the chance of firing is increased. An IPSP has the opposite effect, decreasing the chance of the action potential firing. This is how neurotransmitters, hormones, psychoactive substances,

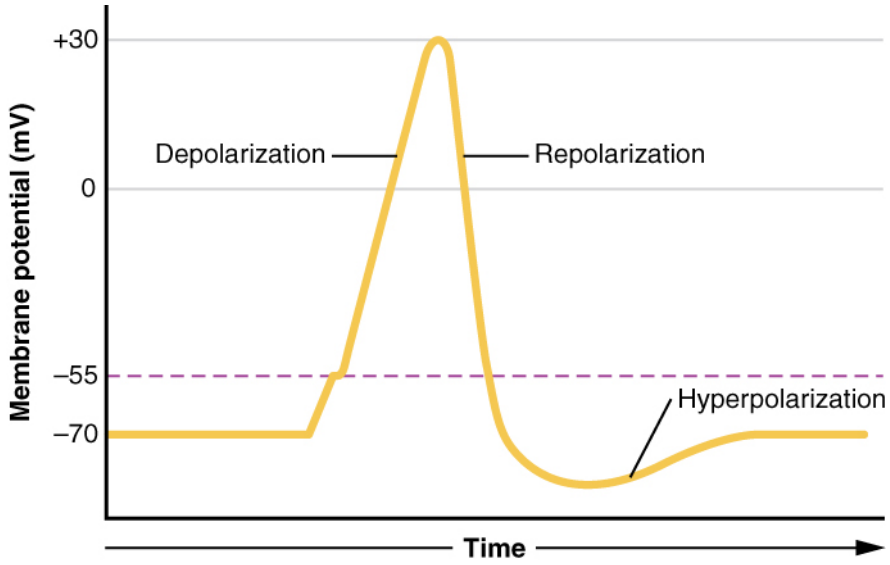

Figure 2.3: Illustration of the action potential[57]

etc. affect the nervous system[46].

The sum of the electrial activity of a large group of neurons has a measurable effect on the electromagnetic field surrounding the head. Not every neuron contributes equally to the field which is measured by EEG. Those neurons that contribute to EEG are primarily those that form so-called *open fields*[9]. Open fields are generated by a type of neuron known as *pyramidal neurons* in the cerebral cortex. When these neurons fire synchronously, they generate coherent electric and magnetic fields rather like those of a *dipole*, such as the one illustrated in figure 2.4.

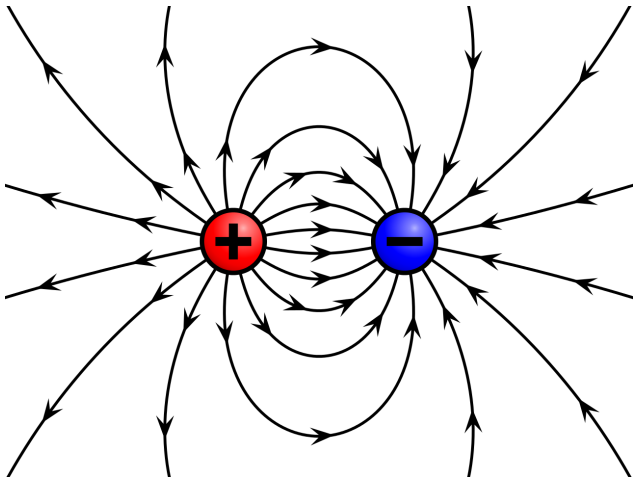

Figure 2.4: Dipole. Image from Wikipedia[62]

These fields can be measured. Measuring the magnetic fields gives rise to MEG, measuring the

electrical field using subdermal implants gives rise to ECoG. Finally, by placing electrodes on the scalp we measure EEG.

2.4 Scalp EEG

Scalp EEG involves placing a set of electrodes on the scalp of the subject and measuring the electric fields generated by the brain. The number of electrodes can vary from a single one to several hundred. Dry electrodes can be placed directly on the scalp, while wet electrodes are connected to the scalp with a highly conductive gel. The measured signal is compared to a specific reference voltage. A common way of defining a reference voltage is to connect an electrode to each earlobe and use their common mode as a reference.

While the number of electrodes vary, they are generally laid out in a standardized manner. The most common standard is the *international 10—20 system*. These number signify that the distance between the electrodes is either 10% or 20% of the front-back or right-left distance of the skull. This layout is illustrated in 2.5. Also commonly used is the *10—10 system*, illustrated in figure 2.6. These systems also provide a nomenclature for describing sensor locations. Each of the letters specifies a particular lobe of the cerebral cortex: prefrontal (Fp), frontal (F), temporal (T), parietal (P) and occipital (O). "C" does not specify a lobe, but stands for "Central". The sensors are also numbered. "z" stands for zero and specifies the electrodes along the midline of the skull. The remaining sensors are numbered, with odd numbers to the left and even numbers to the right of the midline. The "A"s specify the ears, as these are commonly used for reference electrodes.

2.5 Neural Oscillations

There remains the question of how to interpret EEG signals. It has been observed that electrical activity in the brain generally consists of a number of distinct modes at different frequencies. Because of this, EEG signals are often separated into frequency bands labeled using Greek letters. The different brain rhythms are illustrated in table 2.1

Brain rythm	Frequency range
Delta (δ)	< 4 Hz
Theta (θ)	4-7 Hz
Alpha (α)	8-15 Hz
Beta (β)	12-30 Hz
Gamma (γ)	> 30 Hz

Table 2.1: The most common brain rythms

The activities of these different waves are known to correlate with a variety of phenomena. The beta waves are for instance known to correlate with feelings of stress[3].

2.5.1 The Mu Waves

Another type of rythm that is of particular interest to BCI research are mu (μ) waves, with a frequency range of 7.5-12.5 Hz. While this range overlaps with that of the alpha waves, alpha waves are mostly found near the visual cortex. The mu waves, on the other hand, are found near the motor cortex, located slightly anterior to the center of the brain. These rhythms are most prominent when the body is at rest, and are suppressed when some motor activity is performed or, crucially, when some motor activity is imagined. As imagined motor activity is a common control signal in BCI applications, this rhythm is important within this field.

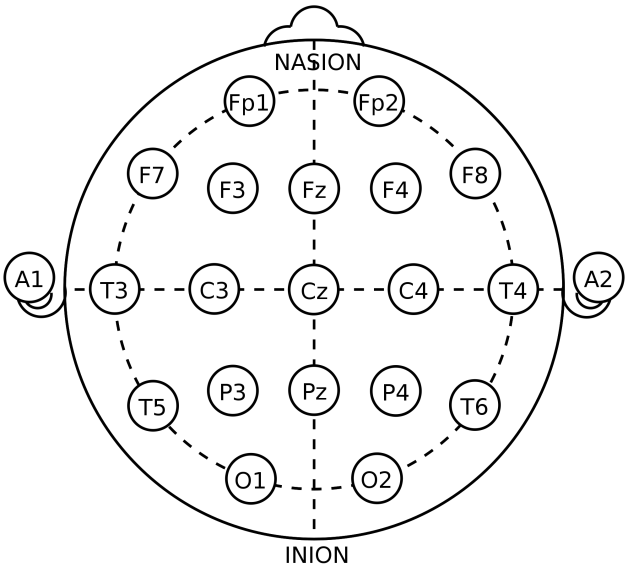

Figure 2.5: 10—20 system[60]

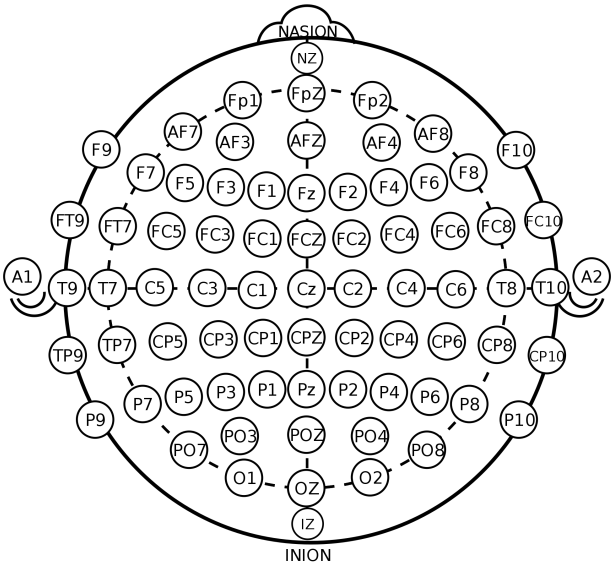

Figure 2.6: 10—10 system[61]

Chapter 3

Signal Processing

There are several methods of representing signals as two-dimensional images. This chapter explains the Short-time Fourier Transform (STFT), Continuous Wavelet Transform (CWT) and Hilbert-Huang Transform (HHT), the three methods used in this thesis.

3.1 The Fourier Transform

One of the most fundamental quantities when analyzing signals is the frequency. For a simple periodic signal, such as a sine wave, with period ΔT, the frequency can be defined as in equation 3.1

$$f = \frac{1}{\Delta T} \tag{3.1}$$

For most interesting signals, however, the period can't be unambiguously defined. Consider the signal defined in equation 3.2.

$$x(t) = 0.7\sin(2\pi 3t) + 1.2\sin(2\pi 7t) + 1.0\sin(2\pi 11t) \tag{3.2}$$

This signal, subject to white noise, is plotted in figure 3.1.

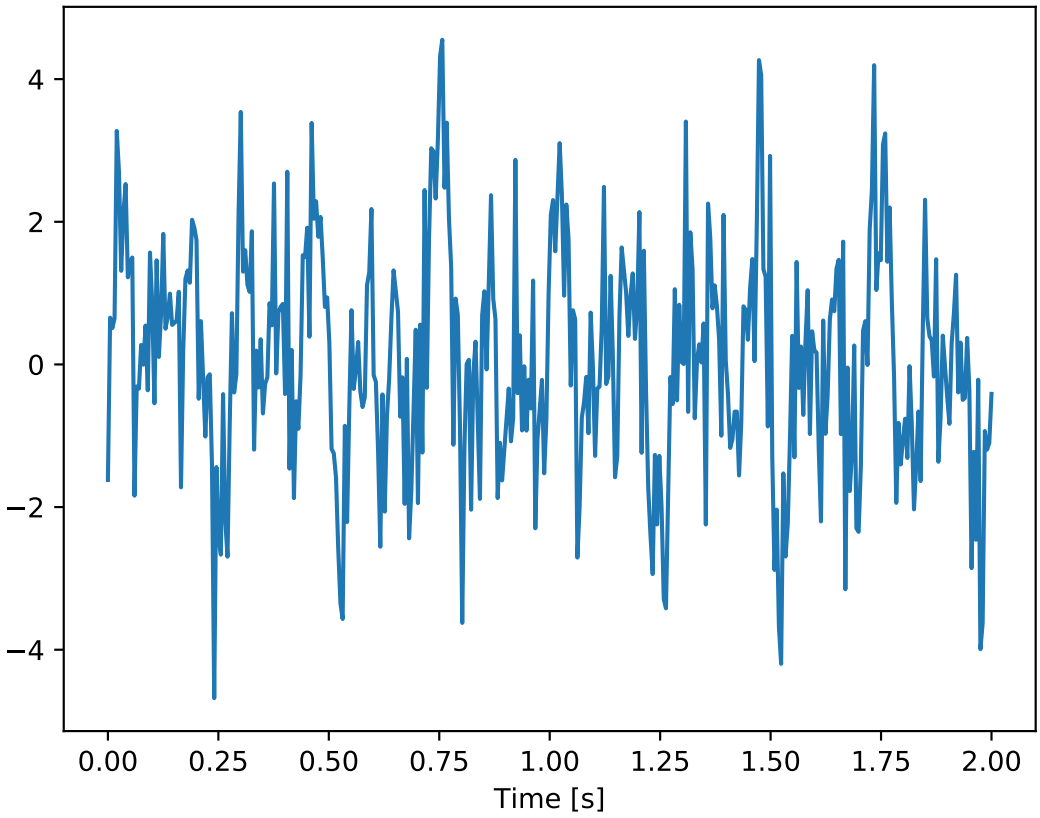

Figure 3.1: Example of time series signal

Here, no single frequency can be defined. For the analysis of such signals, we consider the *frequency spectrum* of the signal. The frequency spectrum of a signal is found using the *Fourier transform* which, for a signal $x(t)$, is defined as in equation 3.3.

$$\mathscr{F}\{x(t)\} = \hat{x}(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} \mathrm{d}t \tag{3.3}$$

The Fourier transform may be interpreted as a decomposition of the signal into an infinite set of sine waves. The value $\hat{x}(f)$ is a complex value whose absolute value indicates the "amount" of the signal witch is of frequency f [45].

$|\hat{x}(f)|^2$ is known as the Power Spectral Density (PSD) of the signal. The PSD of the signal in figure 3.1 is shown in figure 3.2.

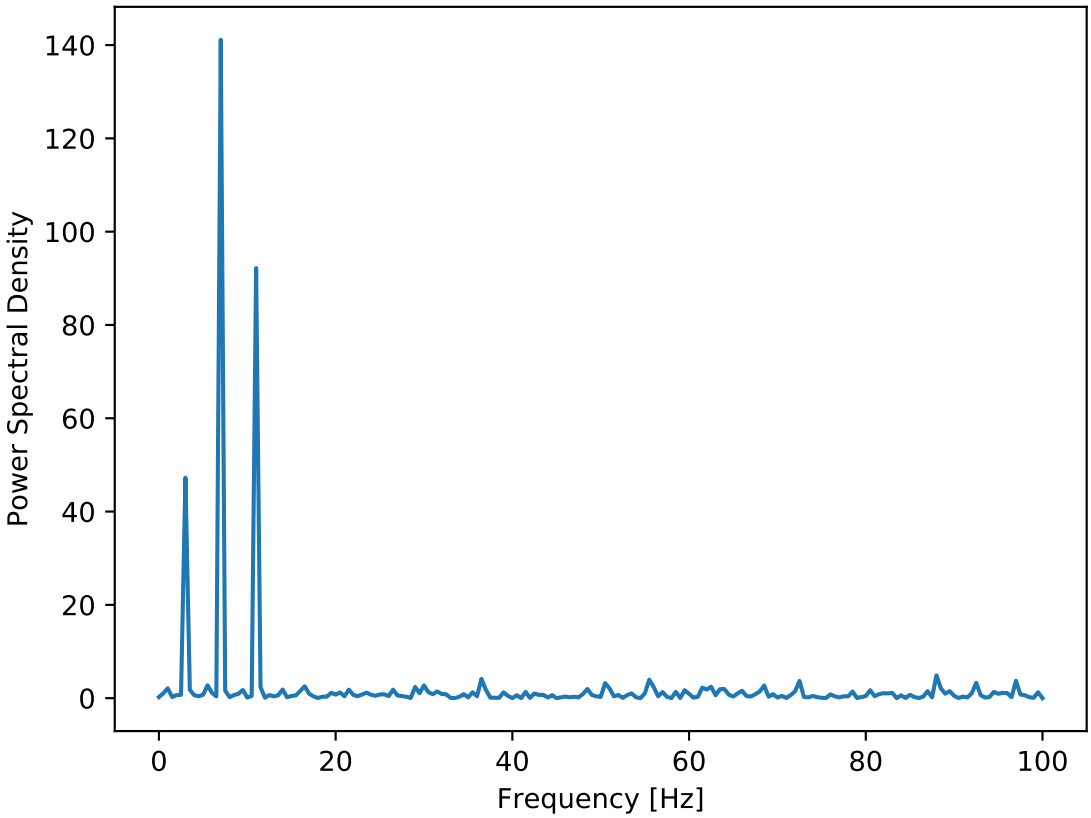

Figure 3.2: PSD of signal in figure 3.1

In this figure three peaks can be identified. The smallest at 3 Hz, the second tallest at 11 Hz and the tallest at 7 Hz. This is exactly as expected given equation 3.2.

3.2 Time-Frequency Analysis

While the Fourier transform is useful for analyzing a wide variety of signals, many real-world processes, including brain activity, are non-stationary. For signals measuring such processes, the frequency and amplitude may change in time. As the Fourier transform is an integral over all time, however, this change can't be captured. As an example, consider the signals shown in figure 3.3. These signals are known as *chirps*, and their frequency change linearly from 1 to 20 (upper) and 20 to 1 (lower). When we look at the power spectra of these signals however, they

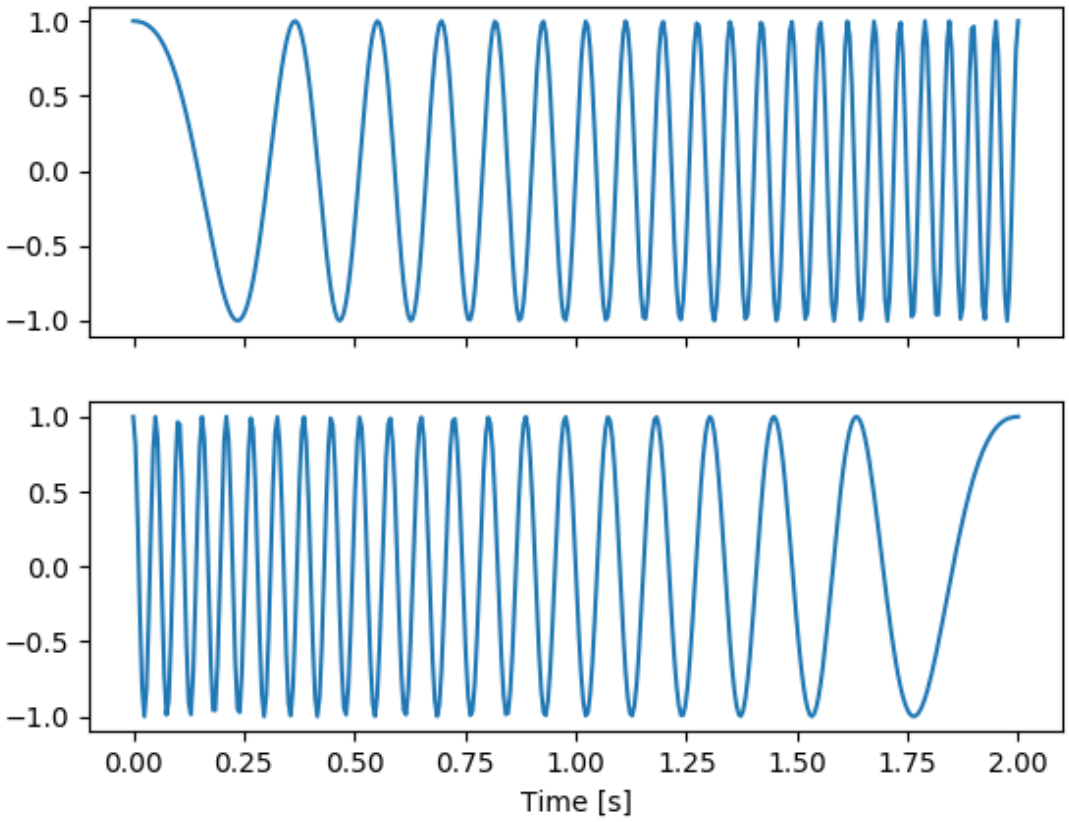

Figure 3.3: Up-chirp and down-chirp

are indistinguishable (see figure 3.4).

3.2.1 Short Time Fourier Transform

Other methods are needed to analyze non-stationary signals. One such method is known as the STFT. This method involves multiplying the signal with a *window function*, which is centered on some value and non-zero for only a small interval surrounding this value. The Fourier transform is then applied to the result of this multiplication. The resulting PSD is then taken to be the frequency spectrum at the time on which the window in centered. By shifting the window along the time axis, we can see how the power at each frequency changes in time. This gives us a two-dimensional representation of the signal, where the signal is divided into "bins" of time and frequency, where each bin indicates the power density at that particular time and frequency.

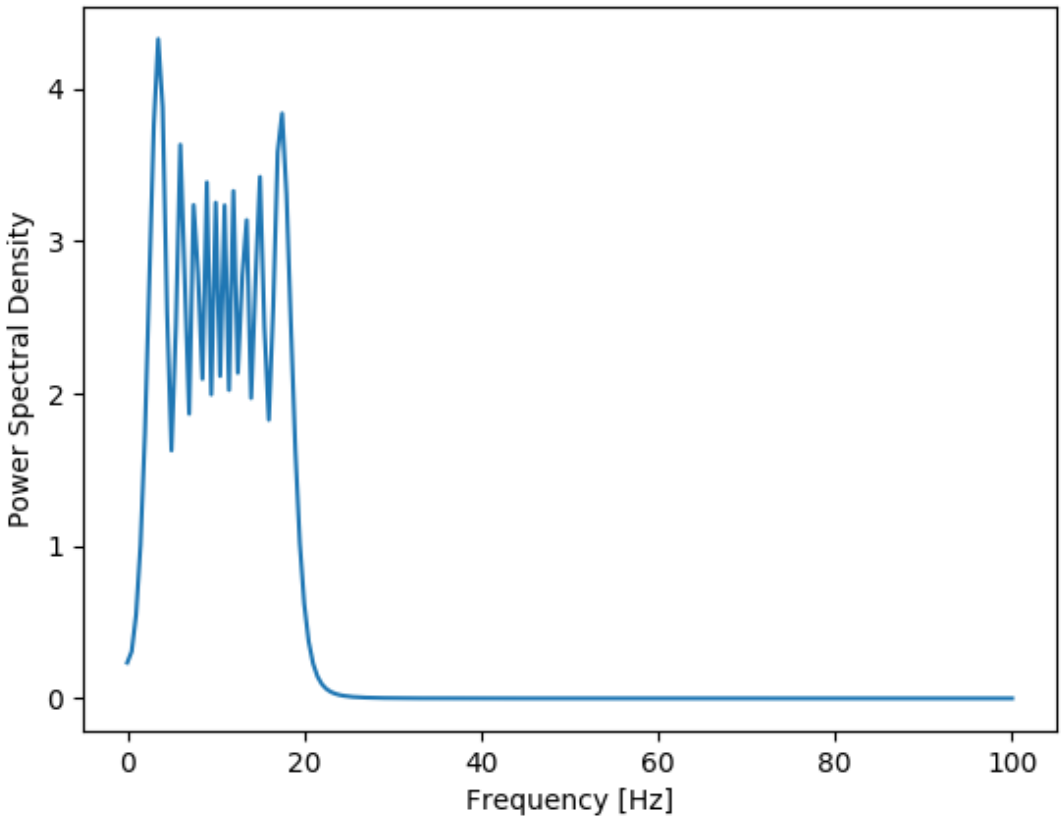

Figure 3.4: PSD of both signals shown in figure 3.3

Such a representation is known as a *spectrogram*.

The signal in figure 3.5 is a up-chirp from 1 to 20 Hz added to a 40 Hz sine wave. It's spectrogram is known in figure 3.6.

3.2.2 Continuous Wavelet Transform

STFT uses a particular resolution for time and frequency by considering the Fourier transform of a given time interval. This, however, fails to take advantage of the fact that different frequencies become apparent at different scales.

To illustrate this, consider the signal in figure 3.7, which plots a signal consisting of a low frequency and a high frequency component. In the upper figure, it can be observed that the low

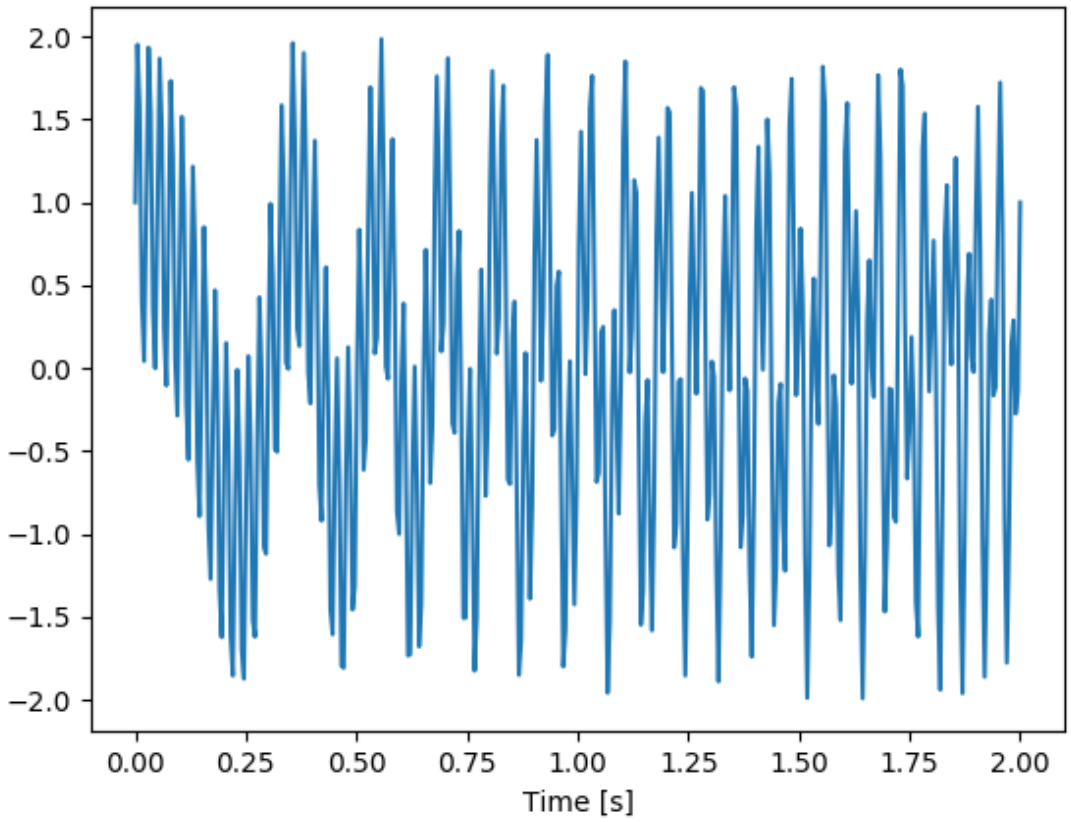

Figure 3.5: Example of non-stationary signal

frequency component oscillates at 4 Hz. This 4 Hz signal is not visible on the "zoomed-in" lower figure. The high frequency component, however, is still visible. By noting that the signal oscillates four times in one millisecond, it's clear that this is a 40 Hz wave.

The CWT takes advantage of this fact. The method involves decomposing the signal into a set of *wavelets*, which is a class of functions that can be described as "brief oscillations". Typically, one decides on some *mother wavelet* $\psi(t)$, such as the one in figure 3.8, and derives the other wavelets by scaling and translating the mother wavelet. The CWT of a signal $x(t)$ is, for a given scale a and time translation b, defined as

$$X_w(a,b) = \frac{1}{|a|^{1/2}} \int_{-\infty}^{\infty} x(t)\bar{\psi}\left(\frac{x-b}{a}\right) \mathrm{d}t \tag{3.4}$$

From this, we get a two dimensional representation of the data with scale on one axis and time

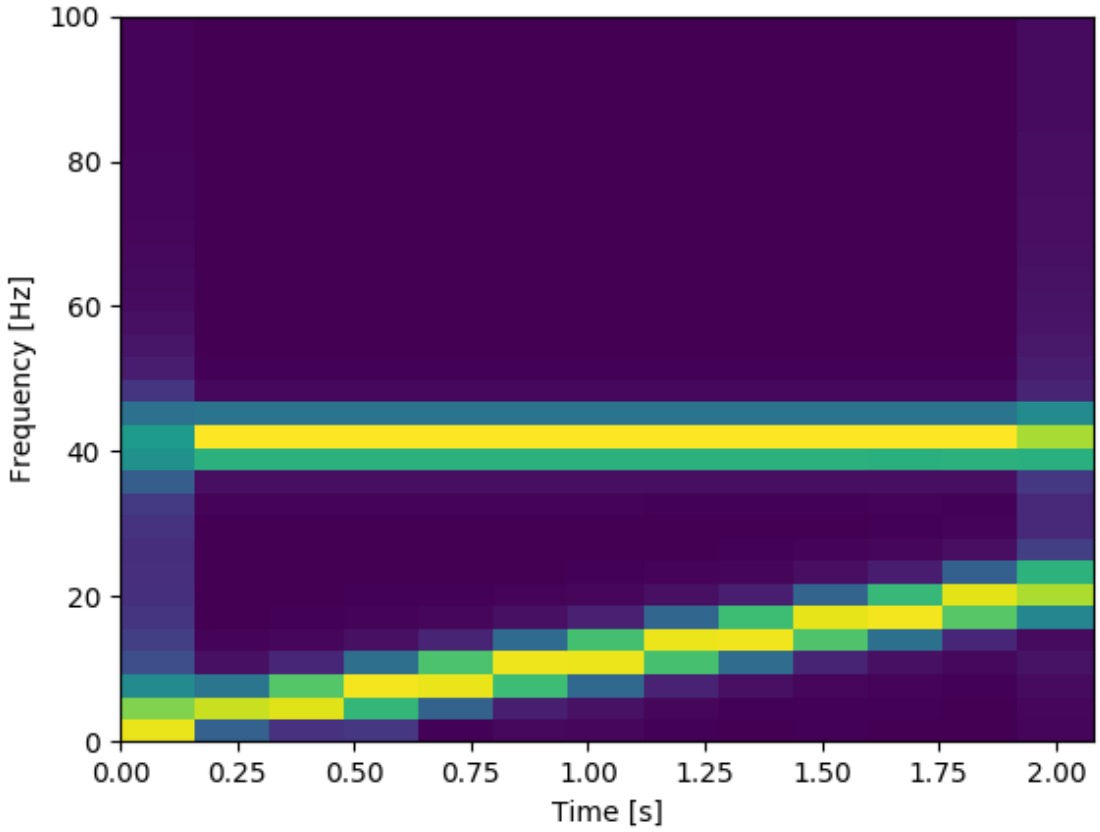

Figure 3.6: Spectrogram of signal in figure 3.5

on the other. This type of representation is known as a *scalogram*. Figure 3.9 shows the CWT of the signal in figure 3.5.

3.2.3 Hilbert-Huang Transform

For any integral method, there is a fundamental relationship between resolution in time and frequency[16]. This is an important consideration when deciding how large to make the "bins" in the spectrogram. If, at one extreme, a single bin covering *all* data points is used, we're left with the ordinary Fourier transform and lose all temporal information. Conversely, the other extreme would be using one bin for each data point. Clearly, a single data point by itself contains no information regarding the frequencies of the signal.

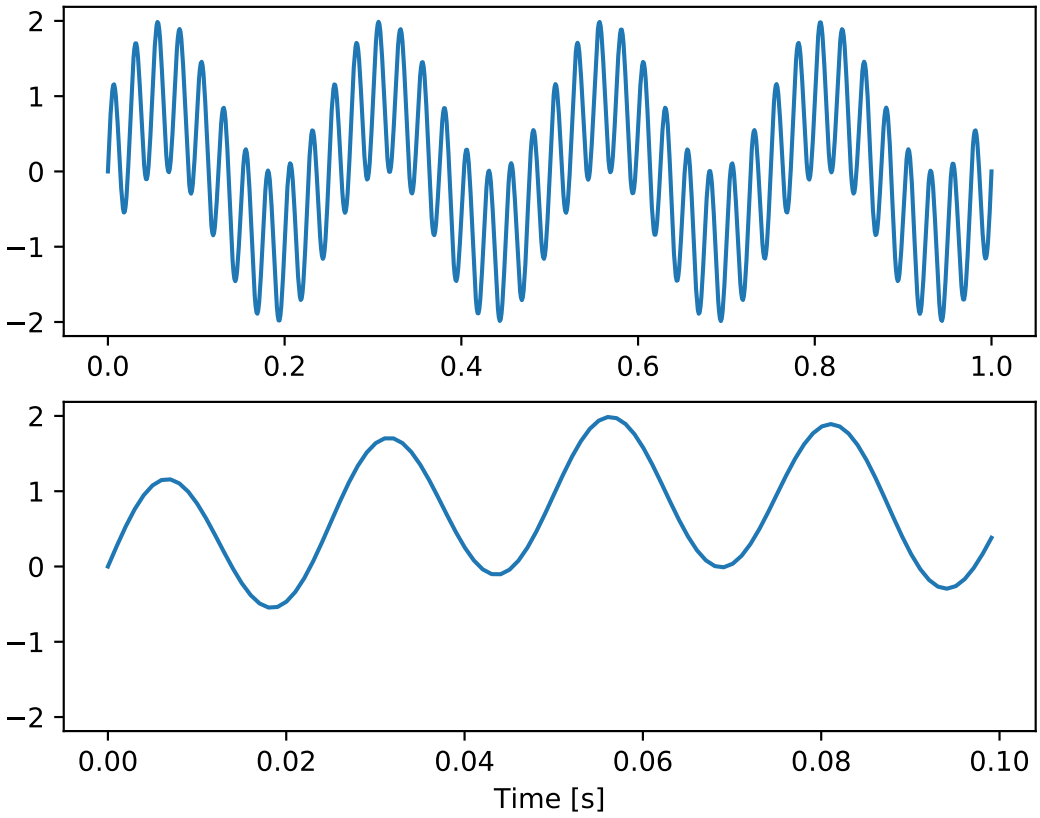

Figure 3.7: The same signal at two time-scales

Increasing the size of the bins improves frequency resolution, but lowers temporal resolution, and vice versa. This trade-off is made formal by the Heisenberg-Gabor inequality, shown in equation 3.5, which relates time resolution Δt and frequency resolution $\Delta\omega$.

$$\Delta t \Delta\omega \geq \frac{1}{2} \tag{3.5}$$

CWT tries to mitigate this by varying the size of the bins, but it is still subject to this limitation. Another strategy for overcoming the constraints of the Heisenberg-Gabor inequality is to use a different conception of the frequency of a signal, known as the Instantaneous Frequency (IF). IF assigns a frequency to each instant of time and, while controversial, has been successfully used in a wide variety of applications[5].

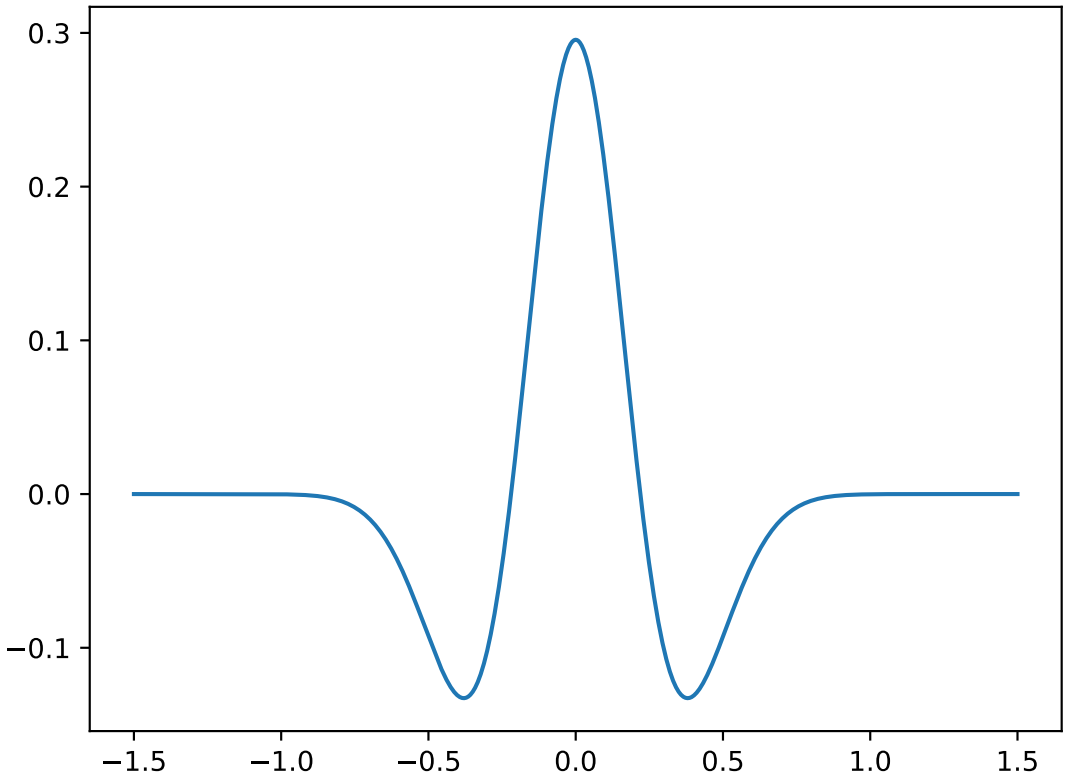

Figure 3.8: The Ricker (Mexican hat) wavelet

To define the IF of a signal $x(t)$, the Hilbert transform defined in equation 3.6 is used.

$$\mathscr{H}\{x(t)\} = \hat{x}(t) = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{x(\tau)}{t-\tau}\mathrm{d}\tau \tag{3.6}$$

Using the Hilbert transform, the *analytic representation* of the signal can be defined as in equation 3.7. This complex-valued signal is similar to the original, but the imaginary part cancels out all negative frequency components of the original signal. This is not a problem, as the negative frequency components carry superfluous information[6].

$$z(t) = x(t) + j\hat{x}(t) \tag{3.7}$$

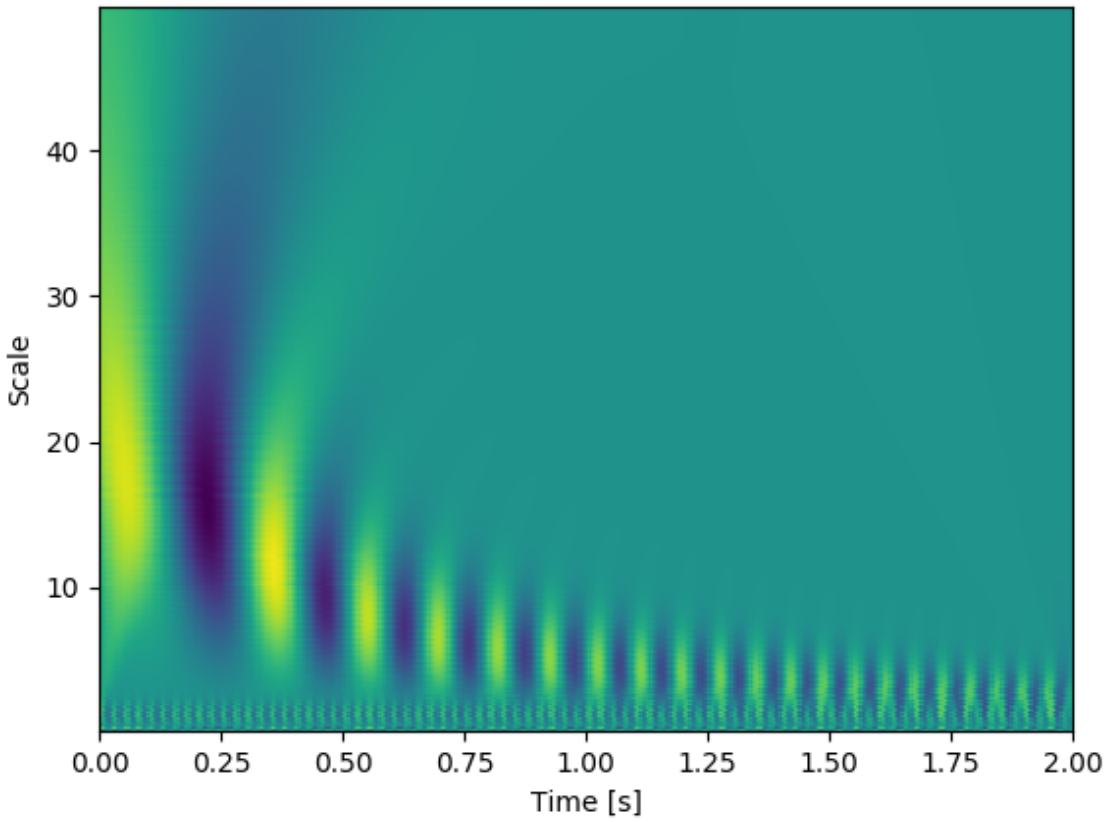

Figure 3.9: CWT of the signal in figure 3.5

Being complex, we can represent any analytic signal in polar form, see 3.8.

$$\begin{aligned} A(t) &= \sqrt{|x(t)|^2 + |\hat{x}(t)|^2} \\ \theta(t) &= \arctan \frac{\hat{x}(t)}{x(t)} \\ z(t) &= A(t)e^{j\theta(t)} \end{aligned} \tag{3.8}$$

$A(t)$ and $\theta(t)$ is known as the instantaneous amplitude and instantaneous phase, respectively. The instantaneous frequency is defined in equation 3.9.

$$\omega(t) = \frac{d\theta(t)}{dt} \tag{3.9}$$

As an example of instantaneous frequency and amplitude, consider the signal in figure 3.10. The instantaneous frequency and amplitude of this signal is known in figure 3.11.

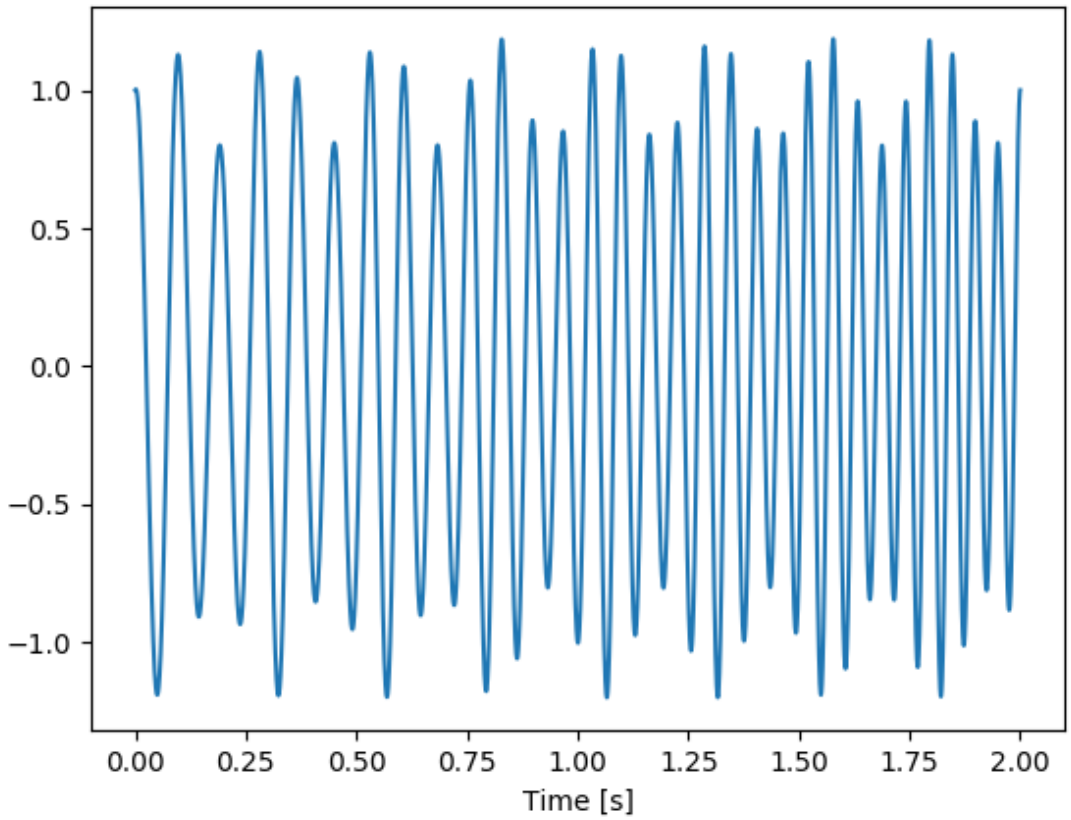

Figure 3.10: Example of signal with varying amplitude and frequency

Unfortunately, not all signals are amendable to instantaneous frequency. Consider, for instance, the signal in figure 3.5. As shown by the spectrogram in figure 3.6, two distinct frequencies are found at each time. As such, it is not clear what one might expect the instantaneous frequency to be, and attempting to apply the definition of IF directly leads to the nonsensical results shown in figure 3.12.

The HHT is a method which creates a kind of spectrogram based on instantaneous amplitude and frequency. It is the combination of the methods Empirical Mode Decomposition (EMD) and Hilbert Spectral Analysis (HSA).

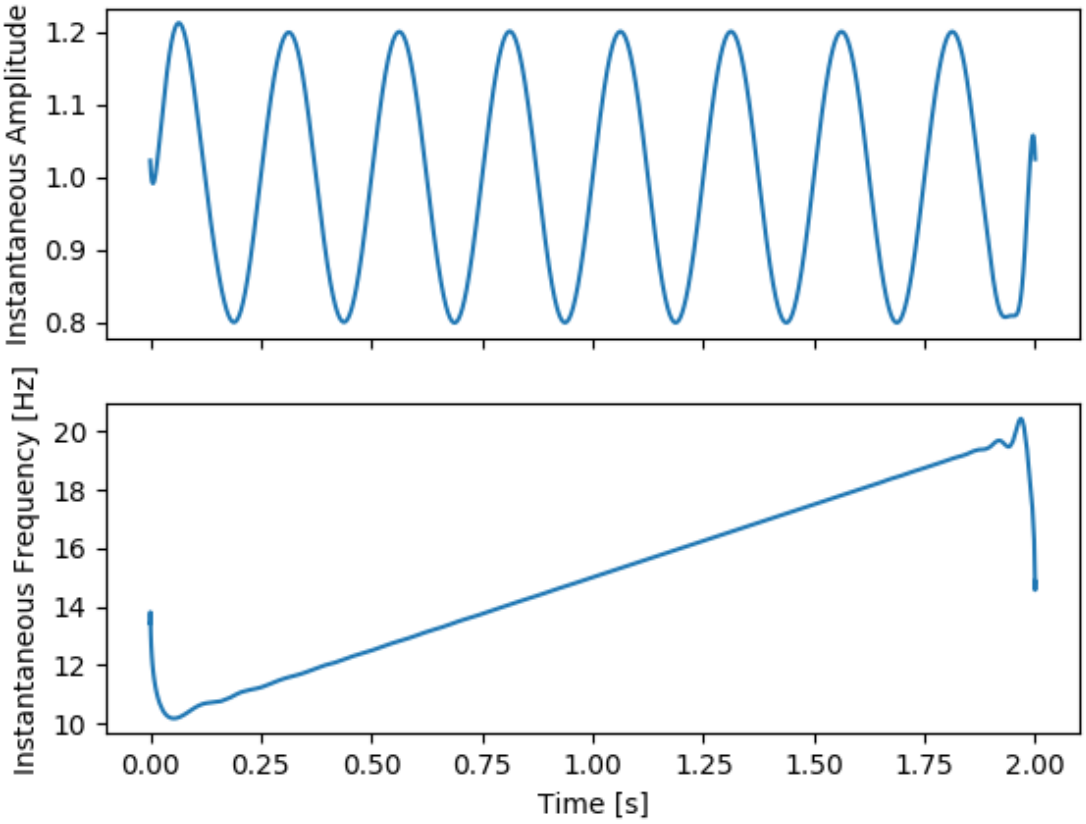

Figure 3.11: Instantaneous amplitude and frequency of signal in figure 3.10

Empirical Mode Decomposition

There is no formal definition of which signals have a well-defined IF. It is intuitively clear, however, that signals such as the one in figure 3.5, consisting of multiple components of different frequencies, are unsuited. Based on this, Huang et al. propose a method for decomposing a signal into a set of so-called Intrinsic Mode Functions (IMFs), each with a well-defined instantaneous amplitude and frequency[27]. An IMF is a function satisfying the following criteria:

1. In the whole data set, the number of extrema and the number of zero crossings must either equal or differ at most by one.

2. At any point, the mean value of the envelope defined by the local maxima and the envelope defined by the local minima is zero.

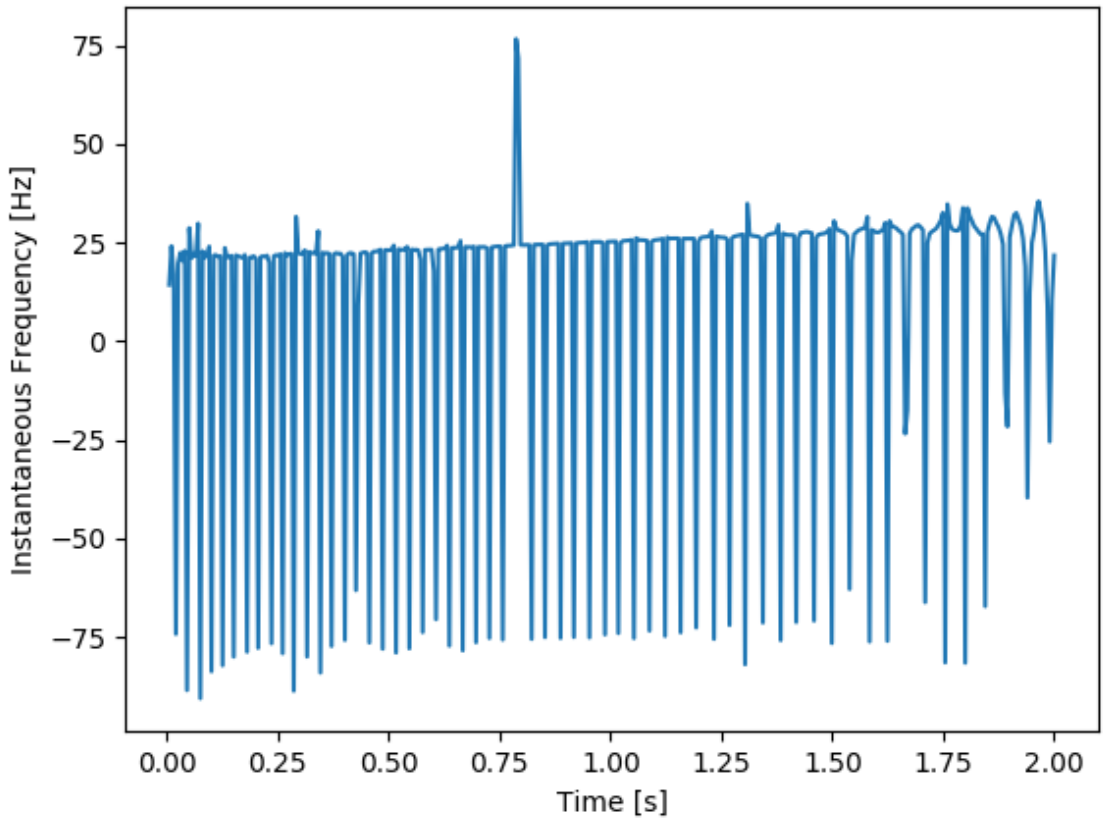

Figure 3.12: Result of attempting to calculate the IF of the signal in figure 3.5

The method of decomposing a signal into IMFs is known as EMD. EMD consists of finding successive oscillation modes using a method known as *sifting*. Consider a signal $x(t)$. The sifting process is defined as follows, with $h_0(t) = x(t)$

1. Define an upper and lower envelope $e_{up}(t)$ and $e_{low}(t)$ as a cubic spline of the maxima and minima of $h_i(t)$, respectively
2. Define the local mean $m_i(t) = 0.5 \cdot (e_{up}(t) + e_{low}(t))$
3. Let $h_{i+1}(t) = h_i(t) - m_i(t)$
4. If the stopping criteria are met, let $h_{i+1}(t)$ be the IMF. Otherwise, the sifting process is repeated with $h_{i+1}(t)$ as input

A number of stopping criteria have been proposed. This is further dicussed in chapter 6. Addi-

tionally, a cubic spline interpolation of the maxima will not envelop the signal near the boundaries. If not handled in some way, this will cause the sifting process to fail. A number of countermeasures have been proposed. This too is further discussed in chapter 6. One step of the sifting process is illlustrated in figure 3.13.

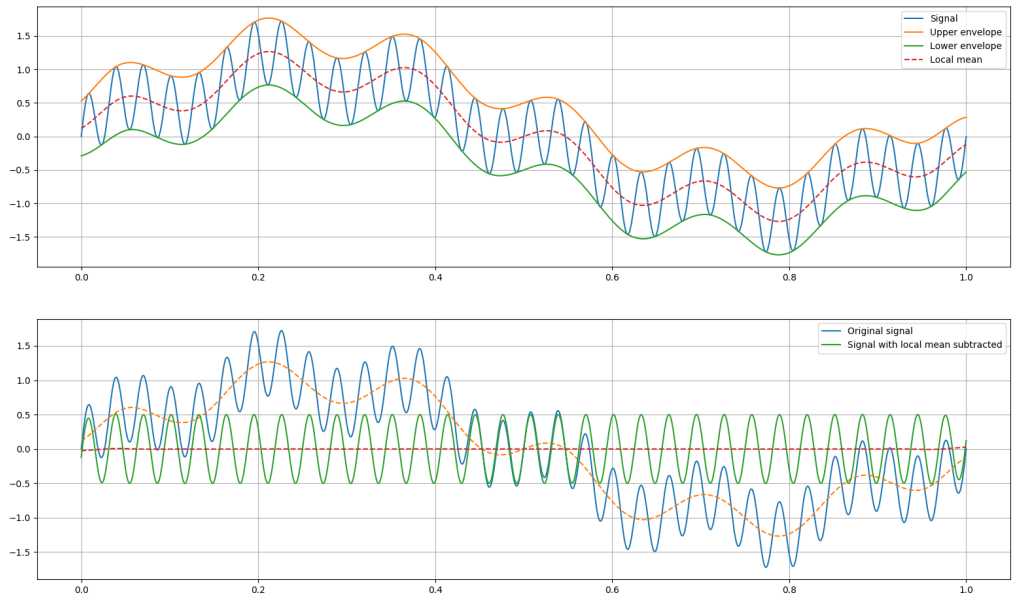

Figure 3.13: A single step of the sifting process

The EMD algorithm can now be defined.

1. Find the first IMF, $c_1(t)$, by performing the sifting algorithm on $x(t)$
2. Subtract the first IMF and define the *residue* $r_1(t) = x(t) - c_1(t)$
3. Find the second IMF, $c_2(t)$ by performing the sifting algorithm on $r_1(t)$ and the residue by computing $r_2(t) = r_1(t) - c_2(t)$
4. Repeat the process until the residue $r_N(t)$ is a monotonic function

This algorithm returns a set of N IMFs, as well as the final residue $r_N(t)$, also known as the *trend*. Figure 3.14 shows the EMD of the signal in figure 3.5.

Hilbert Spectral Analysis

HSA is a method of analyzing the result of the EMD. For each IMF $c_j(t)$ we find the instantaneous amplitude and frequency, $a_j(t)$ and $\omega_j(t)$ respectively. We then define the Hilbert spectrum of the IMF as in 3.10

$$H_j(\omega, t) = \begin{cases} a_j(t), & \text{if } \omega = \omega_j(t) \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

The Hilbert spectrum of the signal is defined as

$$H(\omega, t) = \sum_{j=1}^{N} H_j(\omega, t) \tag{3.11}$$

Though not used in this paper, one can also study the *cumulative Hilbert spectrum*, which integrates the Hilbert spectrum over time to produce a kind of PSD using instantaneous frequency rather than the Fourier transform.

Figure 3.15 shows a flowchart of the HHT algorithm.

Figure 3.16 shows a highly non-stationary signal consisting of two components: a chirp going from 40 to 70 Hz whose amplitude oscillates between 1 and 2, and a frequency modulated signal whose frequency oscillates around 20 Hz and whose amplitude linearly increases from 1 to 2. The Hilbert spectrum of this signal is shown in 3.17.

Mode Mixing

A common problem when employing EMD is mode mixing. This happens when two modes are too close to one another in frequency, and is difficult to counteract in the general case. Consider, for example, the *beat effect*. This is a situation where the superposition of two constant-amplitude modes of similar frequency appear identical with a single amplitude-modulated signal. In these cases the decomposition is ambiguous: either interpretation (two constant-amplitude modes or one amplitude modulated mode) may be considered valid. This is further explored in

Rilling & Flandarin (2008)[49], where the conditions under which mode mixing of this type occurs is explored.

Another situation where mode mixing may occur is in the presence of intermittency, a common feature of turbulent or chaotic processes. This is a situation where small high frequency signals appear intermittently, such as in figure 3.18. When this happens, the signal will contain some extrema that belong to the intermittent signal and some that belong to the underlying mode. The resulting IMF will therefore contain two disparate modes[25].

A number of techniques have been proposed to handle mode mixing. Ensemble Empirical Mode Decomposition (EEMD) is a proposed method which involves performing the sifting process on an ensemble consisting of the signal with different white noise signals added. The average of the ensemble is then computed as the final EMDs[64]. While this method does deal with mode mixing, EEMD is *very* slow.

An alternative method of separating modes that are close in frequency involves the use of a masking signal. A masking signal of a particular frequency and amplitude is determined, and EMD is performed with this signal added, and again with the signal subtracted. Intuitively, the masking signal "pushes" the intrinsic modes apart so that they can be seperated by EMD. The IMFs can then be averaged, and as the masking signal is added to one and subtracted from the other, it should cancel out during the averaging process. The primary challenge with this method is determining the frequency and amplitude of the masking signal. A method for determining these values has been proposed by Fosso and Molinas (2017)[15].

3.3 Comparison of Time-Frequency Transformations

The three methods introduced in this section are all valid methods of creating time-frequency representations of signals, each with their own advantages and disadvantages.

The STFT is an intuitive extension of the well-known Fourier transform to non-stationary cases. As such, it has an extensive theoretical and empirical backing. The primary disadvantage of this method compared to the CWT, however, is that it is limited by the Heisenberg-Gabor in-

equality. CWT uses size-adjusted segments to overcome this difficulty. On the other hand, the spectrograms produced by STFT are generally easier to interpret than the scalograms produced by CWT, as can be seen by comparing figure 3.6 with figure 3.9. This is due to known issues caused by interference terms, border distortion and energy leakage[53].

The HHT is quite different from both of these transforms. Both the STFT and the CWT decompose the signal into a fixed basis: sine waves of given frequencies in the case of the STFT and wavelets at different scales in the case of the CWT. As such, they are not adaptive to the data which they are applied to, but rather assume that all signals can be decomposed in the same way.

The HHT, on the other hand, does not assume a basis. The method decomposes the signal in a fully data-driven way, thus making it more adaptive to the given data. Another advantage of this method is the fact that the frequency calculation is based on differentiation rather than integration, and as such it is not constrained by the Heisenberg-Gabor inequality.

One notable difference between the HHT and the two other transforms is that HHT is defined in an algorithmic and empirical, rather than rigorously mathematical, way. A problem with this is that it becomes quite difficult to examine the properties of the transform, such as when, how and why it fails. The mathematical theory surrounding the Fourier transform provides a framework for rigorously examining the properties of the transform and reasoning about the choices made, such as choice of window function or the size of segments. With the HHT, on the other hand, choices such as stopping criteria, how to handle border effects and how to define the envelopes are made in a somewhat ad hoc manner based on empirical studies and intuition.

Finally, the time complexity of each algorithm should be considered. Bouchikhi et al. (2011)[7] compared matlab implementations of the three algorithms on wind turbine data. Their results are shown in table 3.1. Their results indicate that the STFT is faster than the HHT, which in turn is significantly faster than the CWT. It should be noted, however, that this is a particular implementation on a particular problem. Computational time for the problem examined in this thesis can be found in chapter 8.

Transform	Average Computational Time
STFT	0.2s
CWT	25.2s
HHT	1.57s

Table 3.1: Comparison of the computational time of the three transforms studied in this thesis, by Bouchikhi et al.[7]

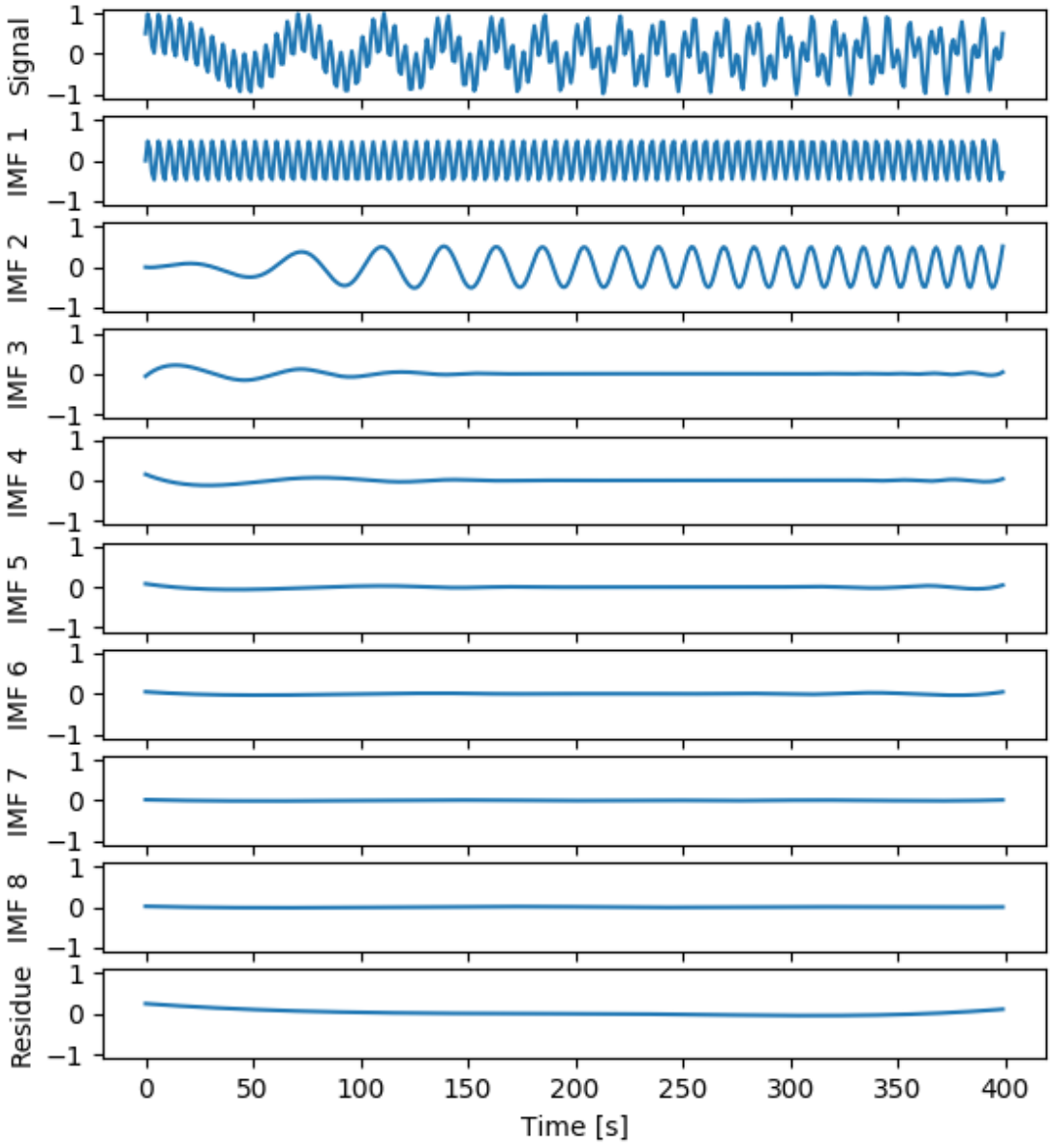

Figure 3.14: Example of EMD

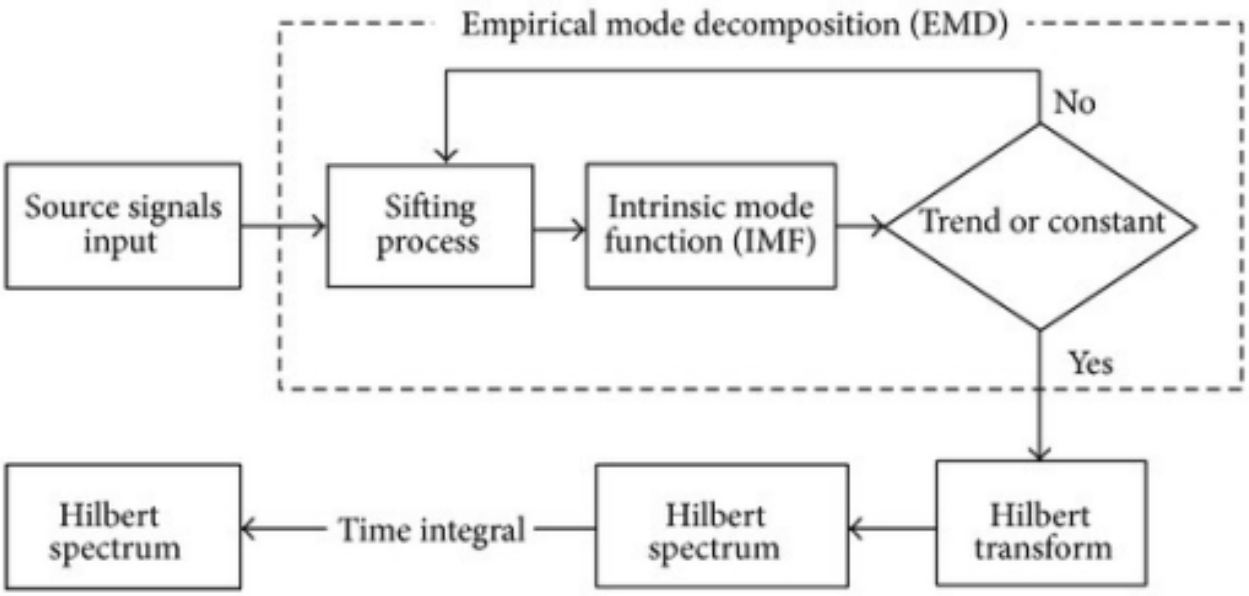

Figure 3.15: Flowchart of the Hilbert-Huang transform[8]

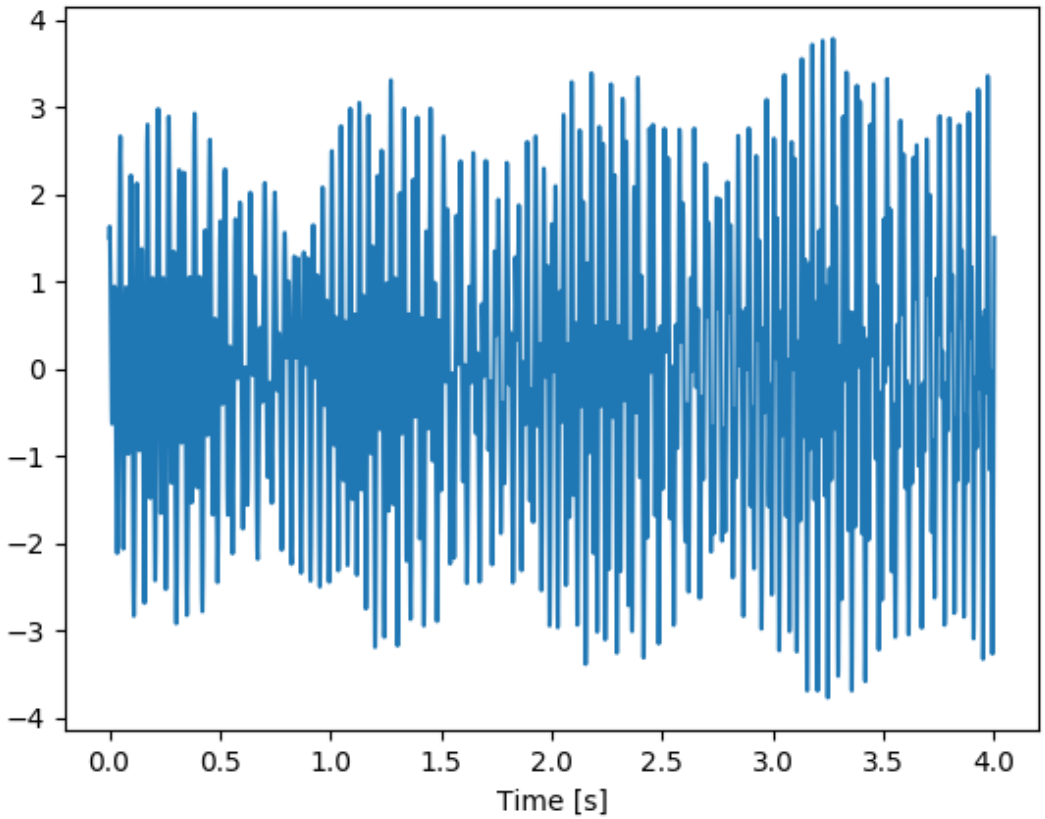

Figure 3.16: Example of highly non-stationary signal

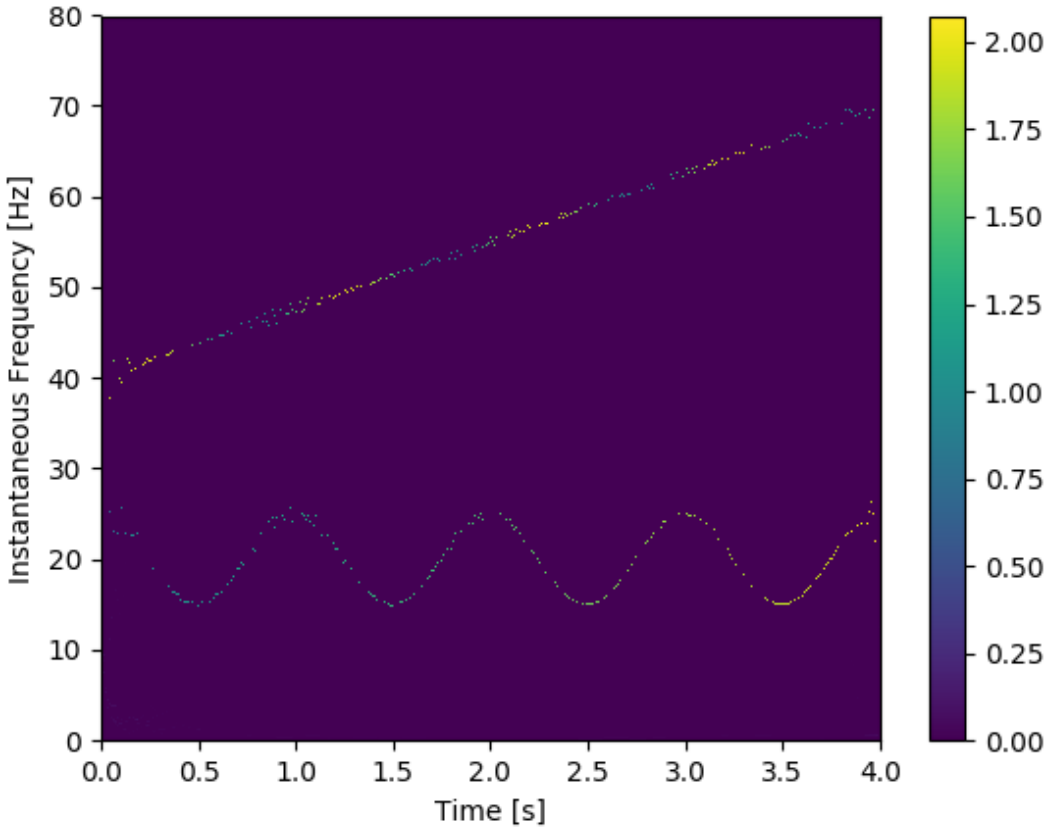

Figure 3.17: Hilbert spectrum of signal in figure 3.16

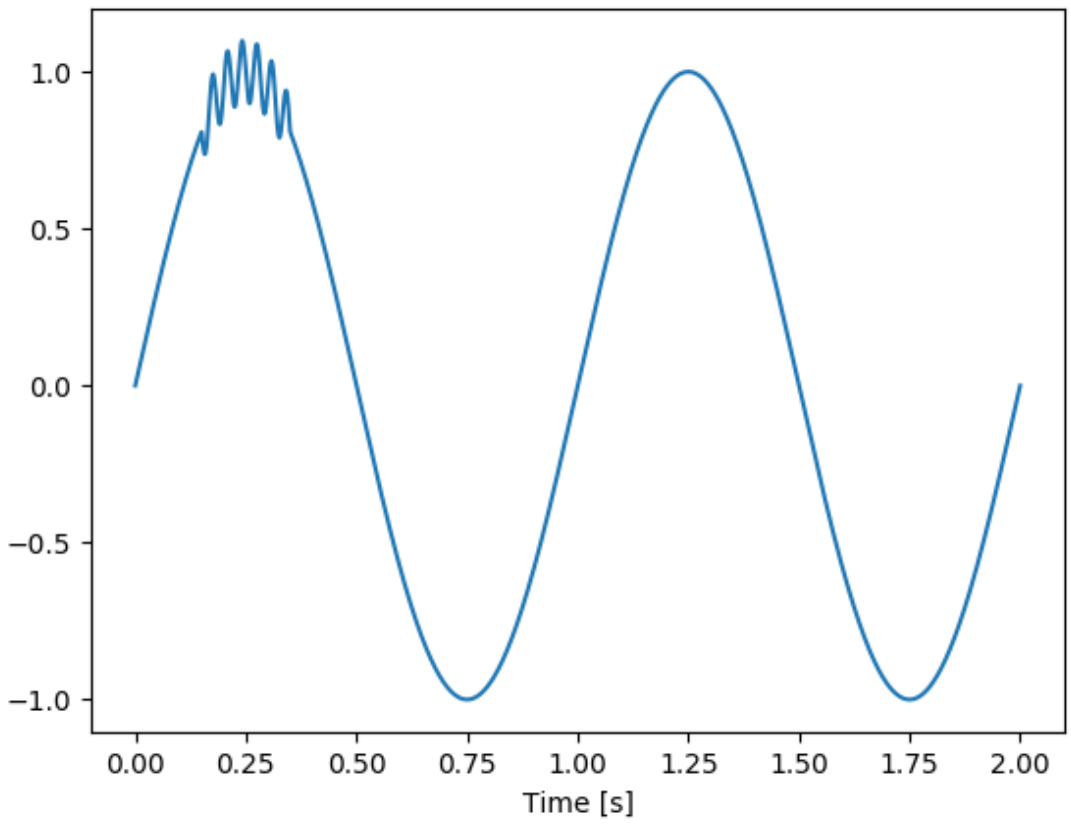

Figure 3.18: 30 Hz signal appearing intermittently over a 1 Hz signal

Chapter 4

Machine Learning

Among the greatest challenges in the development of practical Brain Computer Interface (BCI) is the task of interpreting neuroimaging data. Given an electroencephalography (EEG) recording, we need to know what the subject was thinking during the recording to send the correct command to the output device. Explicitly programming a rule for discriminating between EEG recordings is infeasible given the enormous complexity of the brain, so some other method is required. Machine learning is a field of research that concern techniques and algorithms that allow computers to "learn" how to solve specific problems, rather than having the solution explicitly programmed.

This chapter is divided into three sections. In the first section the necessary fundamentals of machine learning in general are introduced. The second section introduces Artificial Neural Networks (ANNs), the machine learning system used in this thesis. The final section is devoted to Convolutional Neural Networks (CNNs), a recently popularized variation on the ANN that is particularly suited to image classification tasks.

4.1 Machine Learning Fundamentals

An often quoted definition of machine learning was given by Tom M. Mitchell: "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."[43].

Using the problem in this thesis as an example, the tasks T involve mapping EEG segments to specific mental activities. The performance measure P is the accuracy with which the program match the EEG segments to the correct mental activities and the experience E is an exposure to the data set, which contains EEG segments labeled as corresponding to a particular mental activity.

4.1.1 Problem Definition

Most machine learning problems involves finding some function $f(x)$ which maps objects x to some intended output. The objects are usually represented as *feature vectors*, which are vectors whose components capture the relevant features of the object. To do this, the program is trained using a data set $X = \{x^1, x^2, ..., x^N\}$ of feature vectors. If a set of known labels $Y = \{y^1, y^2, ..., y^N\}$ for the objects in X is also provided, the problem is a *supervised learning* task. Otherwise, it is an *unsupervised learning* task. As the task examined in this thesis is a supervised learning task, the rest of this chapter assumes that Y is provided.

To find the function f, generally known as the *objective function*, it must be given some structure. Usually, a class of functions $f(x;\theta)$ parameterized by a set of parameters θ is decided upon. The problem is then reduced to finding the optimal values for θ, given the data available.

As an example, consider the problem of line-fitting. Given a set of points, the goal is to fit a line to the points. Here, the set X is the x-values of the points, the set Y is the set of y-values of the points and the goal is to find the line $y = \alpha x + \beta$ that best fit the points. The parameters to be decided are $\theta = [\alpha, \beta]$. Figure 4.1 illustrates a set of points and three different lines with different values for α and β. While most machine learning problems are obviously more complex than line-fitting, it does illustrate the problem of a program learning from data by fitting parameters.

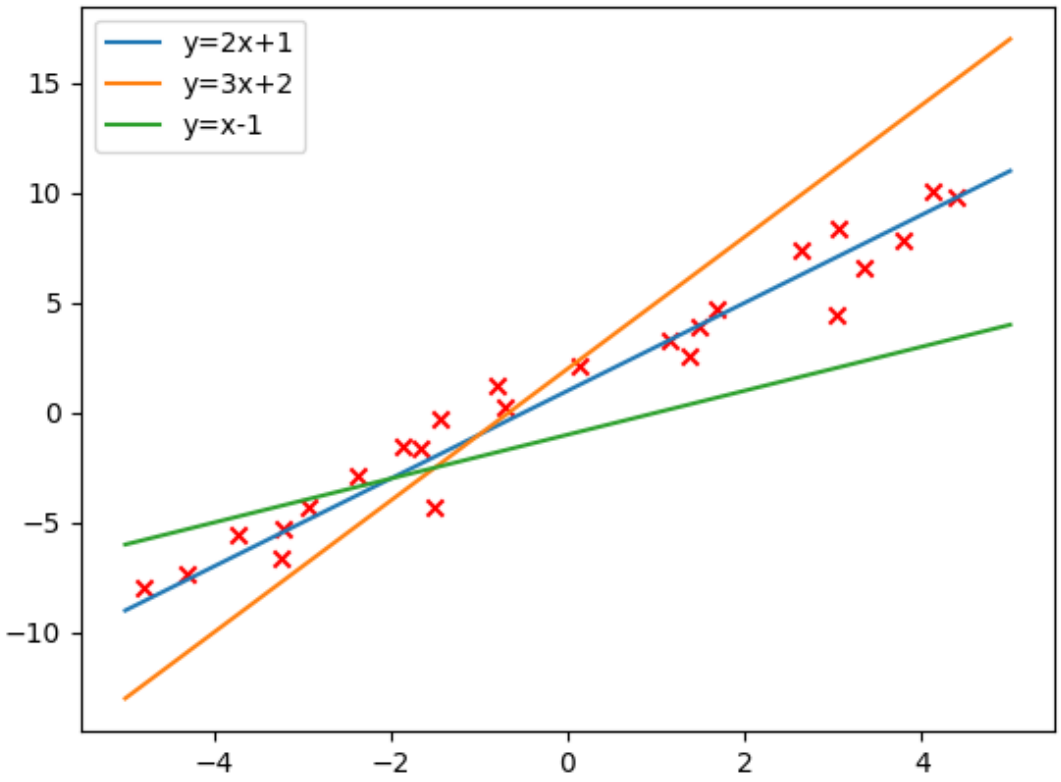

Figure 4.1: Illustration of line fitting problem. The three different lines are three proposals of functions to fit to the parameters

4.1.2 Machine Learning as Optimization

There remains the question of how to find the optimal parameters given a data set. The way this tends to be accomplished is by minimizing some *loss function* (sometimes called a *cost function*) $\mathscr{L}(X,Y,\theta)$. The loss function is in essence a mathematical formalization of the performance measure P from the definition of machine learning. Any number of functions can be used, as long as they are positive-definite and in some sense captures the "badness" of the objective function given the training data and the parameters. The more dissimilar the classifications $\hat{y}^i = f(x^i;\theta)$ is from y^i, the higher the value of the loss function should be. A good example of a

loss function is the Mean Squared Loss (MSL), shown in equation 4.1.

$$\mathscr{L}(X,Y,\theta) = \sum_{i=1}^{N} ||\hat{y}^i - y^i||^2 \tag{4.1}$$

The further $\hat{y}^i$ is from y^i for each i, the higher the value of $\mathscr{L}$, down to a minimum of 0 if they are equal. Thus, this is a valid loss function.

For some simple problems, such as the line-fitting problem described above, the minimum of the loss function can be calculated exactly. In most cases this is impossible or at least infeasible. In these cases, one must turn to the techniques of minimizing functions studied in the field of mathematical optimization.

From this field there are a variety of techniques that can be used. The perhaps best known optimization technique, applicable to differentiable loss functions, is *gradient descent*. The intuition behind this algorithm is that given some function $f(\mathbf{x})$, the direction in which one must move the vector $\mathbf{x}$ in order to decrease the function the most is given by $-\nabla f(\mathbf{x})$ [63]. By calculating the gradient $\nabla_\theta \mathscr{L}$, the parameters can be updated using the rule

$$\theta_{n+1} = \theta_n - \alpha \nabla_\theta \mathscr{L}(X,Y,\theta_n) \tag{4.2}$$

Where α is a small positive value known as the *learning rate*. The lower the learning rate is set, the more slowly the loss function converges. If set too high the step taken might be too large, as the gradient is only the direction of steepest descent close to θ_n. The learning rate is an example of a *hyperparameters*. These parameters are not learned, but set by the programmer prior to training.

4.1.3 Generalization and Validation

The goal of machine learning is to find an objective function that correctly classifies not only those objects that are found in the training set, but which generalize to new objects of the same type. The situation where the objective function works very well on the training data, but fails

to generalize to novel data is known as *overfitting*.

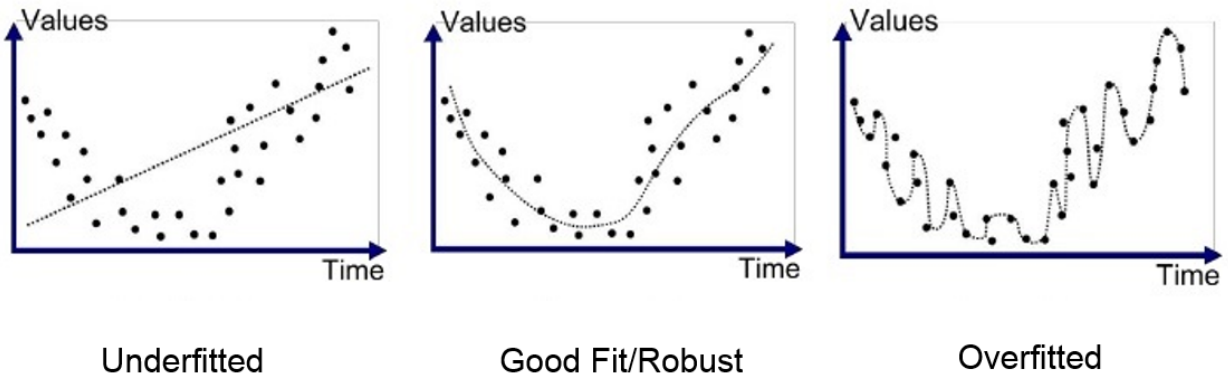

Figure 4.2: Example of underfitting, correctly fitting and overfitting to the data

Consider the three situations in figure 4.2: in the leftmost picture, the objective function, a straight line, does not fit the data very well at all. This is known as underfitting. In the rightmost picture, the objective function fits the training samples extremely well, passing through almost all the samples. It is very unlikely, however, that this objective function is a realistic model of the underlying distribution from which the training data was sampled. Given a random sampling, one would have to be exceptionally lucky to find exactly the samples that fit this model. The picture in the middle is probably a better fit. While the objective function runs through fewer points than in the rightmost picture, it follows the general trend of the data quite well, while remaining fairly realistic.

Overfitting tends to happen either when a complex model is aggressively fitted to a data set that is too small in regards to the complexity of the model. As an overfitted model will fail when put into practice, it is imperative that this is discovered during training. This is commonly done by splitting up the data set into a training set and a validation set. Before training is begun some of the objects, for example 20%, are set aside. The remaining objects are then used in the training process. In the beginning, the objective function probably underfits. As the training process continues, the objective function will fit the training data better. Once all training is done, the objective function is applied to the objects in the validation set. If the accuracy with which the function classifies the validation data is significantly lower than with the testing data, the training process has overfitted.

4.1.4 Feature Engineering and Deep Learning

One of the challenges when designing a machine learning system is how to represent the objects. Designing a lower dimensional feature vector based on the objects is known as feature extraction or *feature engineering*. Here, quantities which are considered important are carefully selected by domain experts.

The problem with feature engineering is that it generally requires a lot of knowledge regarding the problem, and even for experts it ca difficult to know which quantities will be important prior to the classification process. *Deep learning* methods are those methods which not only learn how to classify the data, but based on the raw data also learns the best *representation* of the data for the classification problem at hand. This is sometimes also referred to as *representation learning*.

4.2 Artificial Neural Networks

ANNs are supervised machine learning systems that are, as their name implies, inspired by the workings of the biological neural networks that make up the human nervous system. They are one of the oldest machine learning methods, and research into them has seen a resurgence in recent years due to their success in areas such as image classification, natural language processing and game AI[18][30][54].

4.2.1 Biological Inspiration

The most prominent theory on how learning happens in the human train is known as *Hebbian learning*. This is a theory which postulates that learning is the result of connections between biological neurons strengthening and weakening with use, often summarized as "neurons that fire together, wire together"[21]. Inspired by this, ANNs consist of artificial neurons that imitate the functioning of biological neurons. The workings of biological neurons is described in chapter 2: the neuron receives an input in the form of an electrical signal from other neurons or

sensory cells and if the sum of these inputs is sufficiently strong, the neuron fires its own action potential. The neuron is connected to previous neurons, and the weight of these connections are the learnable parameters of the ANN.

4.2.2 The Perceptron

Inspired by the workings of the biological neuron and the theory of Hebbian learning. The perceptron is a binary classifier: it classifies an input as either 1 or 0. The inputs to the perceptron are the values of the feature vector. There is a weighted connection between each input and the perceptron, modeling the dendrites of a biological neuron. The weighted inputs are summed up and a bias, representing the threshold of activation, is added. This weighted and biased sum is then passed through an *activation function* σ, and the output is used for classification. The activation function in the case of the simple perceptron model is typically the Heaviside step function, which is 1 for positive inputs and 0 for negative inputs. This models the action potential of a biological neuron. The perceptron is illustrated in figure 4.3. If we consider the weights

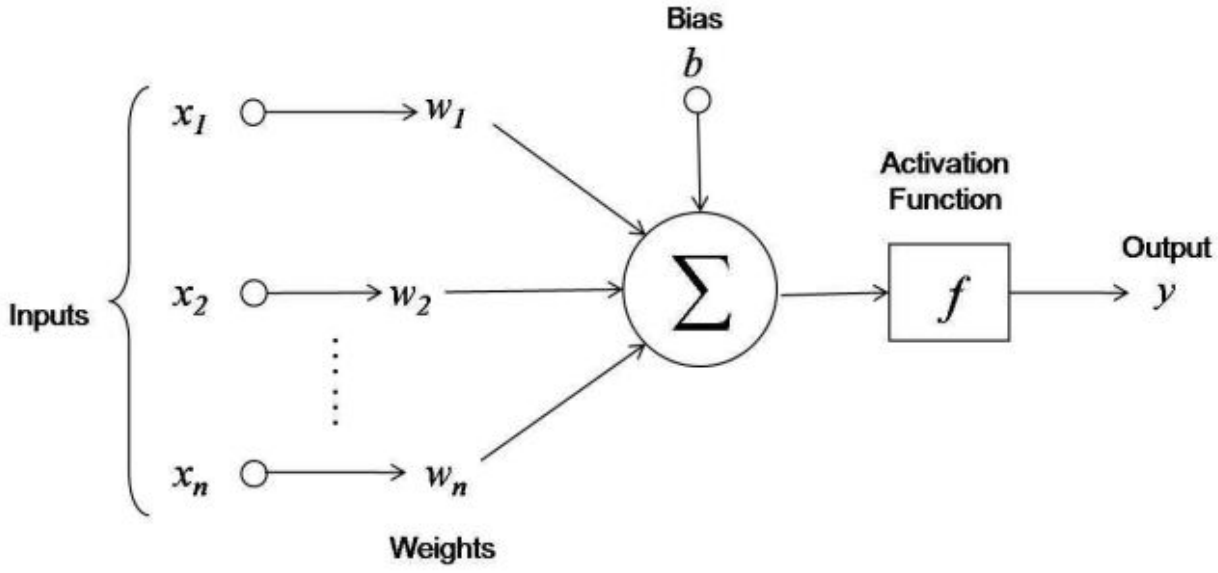

Figure 4.3: The perceptron mode of the neuron

as a vector $\mathbf{w}$, we can define perceptron by the function

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b) \tag{4.3}$$

The weights are the parameters θ. The training algorithm for the perceptron is

1. For each feature vector $\mathbf{x}^j$, calculate output $\hat{y}^j(t) = \sigma(\mathbf{w}^T(t)\mathbf{x}^j + b$
2. Update the weights as $w_i(t+1) = w_i(t) + \alpha(y^j - \hat{y}^j(t))x_i^j$

Where α is the learning rate.

Though limited to binary classification, multiclass classification can be achieved by using a layer of several perceptrons. As each perceptron can output one "bit" of information, at least $\log_2 N$ perceptrons are needed for N-class classification.

While initially promising, early AI researchers quickly discovered the limits of the perceptron: what the perceptron learns is essentially a hyperplane separating the input space. A simple example where the perceptron fails is the XOR (exclusive or) function. This is a function which takes two bits as input and returns 1 if one of the bits, but *not* both, are 1, and returns 0 otherwise. See table 4.1.

X	Y
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

Table 4.1: Inputs and outputs for the XOR function

There is no way for the perceptron to learn this function. The perceptron can only work on *linearly separable* data.

4.2.3 Multi-layer Neural Networks

The significant limitations of the perceptron model lead to a decline in research into neural networks. In the 1980s, however, interest in neural networks picked up as it was realized that these

limitations could be overcome using more advanced networks of artificial neurons. A general multi-layer ANN consists of a number of artifical neurons stacked in layers, with each neuron being connected to all the neurons in the previous layer. The first layer, the *input layer*, is the feature vector. The final layer is known as the *output layer*. The layers between the input and output layer are known as *hidden layers*. A simple neural network is illustrated in figure 4.4.

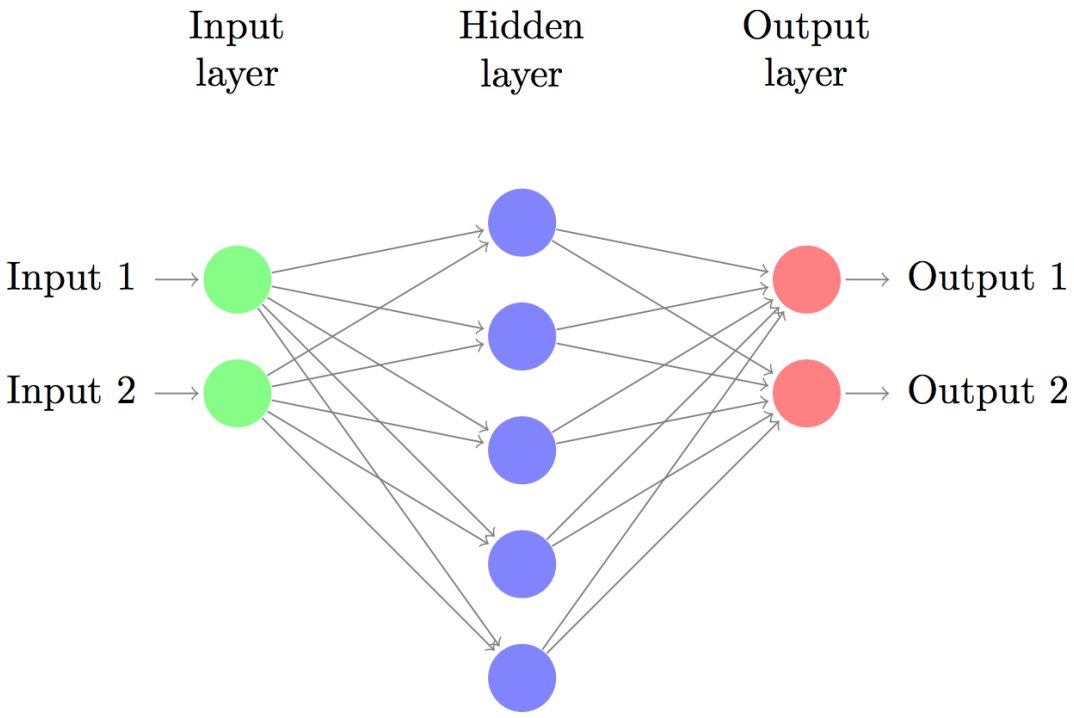

Figure 4.4: Simple ANN with three layers

The hidden layers allow the ANN to overcome the limitations of the simple perceptron model: each of the layers performs a kind of nonlinear transformation of the input, with the aim of making it linearly separable at the final layer. It has in fact been proven that ANNs are capable of approximating *any* continuous function on compact subsets of $\mathbb{R}^n$ [24] (this does not imply, however, that they can easily *learn* any such function).

The Backpropagation Algorithm

One of the reasons why the transition from single-layer perceptrons to multi-layer ANNs was took as much time as it did was the fact that the learning algorithm described in section 4.2.2 can't be used to update the weights and biases of the hidden layers. The algorithm used to train ANNs is known as *backpropagation*.

The backpropagation algorithm is based on gradient descent, and requires the objective function to be differentiable. Because of this, the Heaviside step function used as an activation function in the perceptron model must be replaced. The sigmoid function is a continuously differentiable function which approximates the Heaviside step function. It is illustrated in figure 4.5.

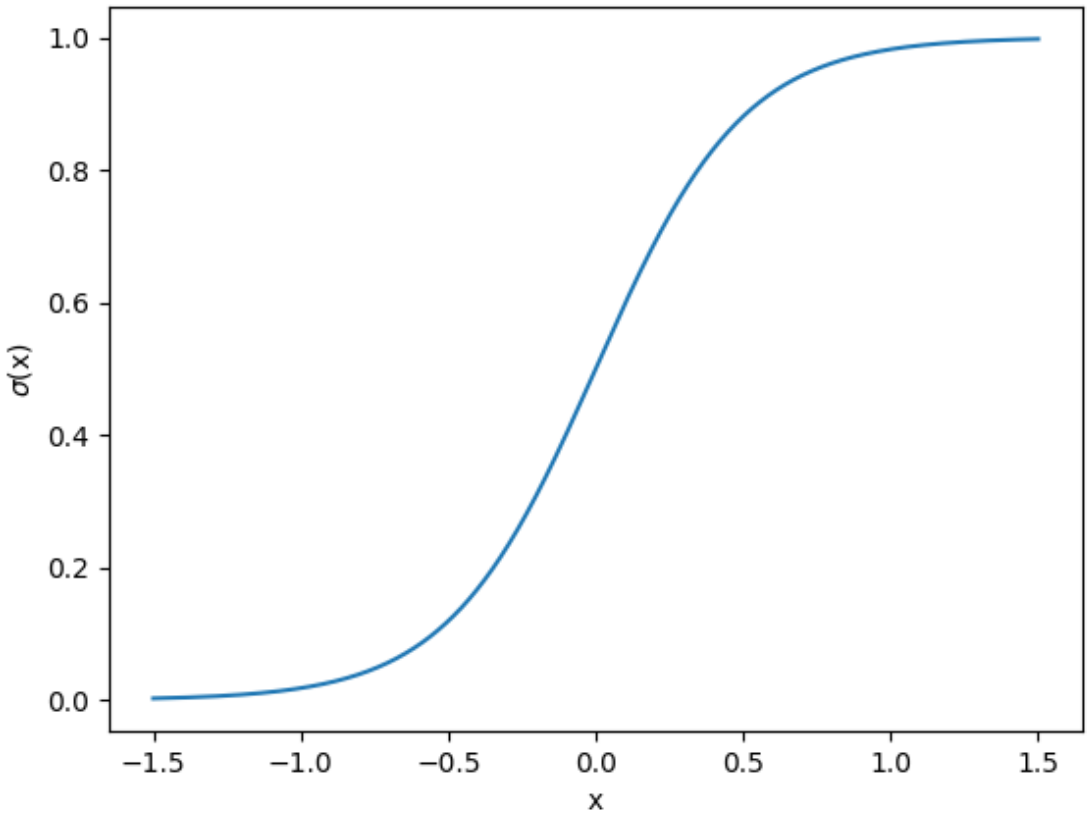

Figure 4.5: Sigmoid function $\sigma(x) = \frac{1}{1+e^{-4x}}$

Equation 4.4 is the function used to propagate the activation forward through the network. Here we denote the activation of the jth neuron of the lth layer as a_j^l, the weight between the kth neuron of layer $l-1$ and the jth neuron of layer l as w_{jk}^l, the bias of the jth neuron in layer l as b_j^l. σ is the activation function.

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \tag{4.4}$$

We note that the activation of each layer can be expressed as a vector $\mathbf{a}^l$, and the weights and biases can be expressed as a matrix $\mathbf{W}^l$ and vector $\mathbf{b}^l$. We can then express the above feedforward activation as a vector equation, with element-wise application of the activation function.

$$\mathbf{a}^l = \sigma\left(\mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l\right) \tag{4.5}$$

We assume we have some differentiable loss function $\mathscr{L}(X,Y,\theta)$, such as MSL. The backpropation algorithm involves applying the chain rule to the gradient of the loss function to find it's derivative with respect to each weight and bias. These are then updated using the gradient descent rule. The steps of the algorithm are shown below. Proofs are omitted in this thesis, though they involve a fairly straightforward application of the chain rule of differentiation. The output layer is denoted as L, and the intermediate values $z_j^l = \sum_k w_{jk}^l + b_j^l$ are defined for convenience. Let $\mathscr{L}^i$ denote the loss associated with the classification of object i.

1. Perform a feedforward classification of x^1
2. In the output later, compute the error $\delta_j^L = \frac{\partial \mathscr{L}^1}{\partial z_j^L} = \frac{\partial \mathscr{L}^i}{\partial a_j^L}\sigma'(z_j^L)$
3. For layers $L-1$ to 1, define the error term as $\delta_j^l = \frac{\partial \mathscr{L}^1}{\partial z_j^l} = \sigma'(a_j^l)\sum_k w_{jk}^{l+1}\delta_k^{l+1}$
4. Compute the derivatives $\frac{\partial \mathscr{L}^1}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$ and $\frac{\partial \mathscr{L}^1}{\partial b_j^l} = \delta_j^l$
5. Repeat the above procedure to calculate the derivatives of $\mathscr{L}^{2..N}$
6. Calculate the final derivatives $\frac{\partial \mathscr{L}}{\partial w_{jk}^l} = \frac{1}{N}\sum_{n=1}^N \frac{\partial \mathscr{L}^n}{\partial w_{jk}^l}$ and $\frac{\partial \mathscr{L}}{\partial b_j^l} = \frac{1}{N}\sum_{n=1}^N \frac{\partial \mathscr{L}^n}{\partial b_j^l}$
7. Update the weights and biases with $\Delta w_{jk}^l = -\alpha\frac{\partial \mathscr{L}}{\partial w_{jk}^l}$ and $\Delta b_j^l = -\alpha\frac{\partial \mathscr{L}}{\partial b_j^l}$

The above procedure is performed a number of times, or *epochs*, for a complete training session. The optimal number of epochs is often problem-dependant.

The method described above, where the parameters are updated only after all the training samples have been processed, is known as full-batch backpropagation. In Stochastic Gradient Descent (SGD), the parameters are updated at the end of each training sample, using the derivative of C^i. This method may cause erratic behaviour in the optimizer, but in practice it often performs just as well or better than full batch backpropgation[38]. A compromise between these two methods is known as *mini-batch backpropagation*. Here, the training data is divided up into smaller batches of a given size and the parameters are updated at the end of training with each batch.

4.3 Deep Neural Networks

In the context of deep neural networks, "deep" has a double meaning. In neural networks, "depth" is a measure of how many layers the network has. The second meaning is in the context of deep learning: machine learning algorithms which learn representations of data, rather than just how to classify them.

The usual interpretation of the hidden layers in an ANN is that they transform the data in some way which is beneficial to classification. Thus, by tuning the weights and biases of the hidden layer the network learns how to transform the input into a better representation. By stacking layers, more complex transformations can be made.

Deep neural networks were for a long time considered too difficult to train to be useful. But in 2006, Geoffrey Hinton introduced a fast training method for a type of neural network known as a deep belief network, which revitalized research into deep networks[23].

While the CNN was first described by LeCun et al. in 1998[39]. The ongoing deep learning revolution began when a CNN-based solution by Krizhenvsky et al.[35] won the ImageNet Challenge by a considerable margin. The ImageNet competition is an annual competition where research teams evaluate algorithms and compete to achieve the highest score on image classification

tasks on a labeled image dataset known as "ImageNet". Krizhenvsky et al. lowered the state-of-the-art error rate from 26.1% to 15.3%.

4.3.1 Convolution in Neural Networks

The ANNs described in section 4.2.3 utilize fully-connected layers, where every neuron is connected to every neuron in the preceding layer. For one, this is not in line with the structure of biological neural networks, where most neurons are connected to only a few neurons. Secondly and perhaps more importantly, the large number of connections lead to a large number of parameters. This is problematic due to what is known as the *curse of dimensionality*, which refers to a number of problems that arise as the number of parameters (or dimensionality of the parameter space) is increased[13]. In particular, the amount of data required for training tends to increase exponentially with the number of parameters. Historically, this has limited the depth of ANNs.

CNNs overcomes this constraint by introducing *convolution*. Given two matrices $\mathbf{I}$ and $\mathbf{K}$, the convolution operator is defined as

$$(\mathbf{I} * \mathbf{K})(i, j) = \sum_m \sum_n \mathbf{I}(i+m)(j+n)\mathbf{K}(m, n) \tag{4.6}$$

Typically, $\mathbf{I}$ is an image or some other large matrix whereas $\mathbf{K}$ will be a significantly smaller matrix known as the *kernel* or *filter*. Convolution can be interpreted as a sliding the kernel along windows of $\mathbf{I}$ and calculating the dot product of the kernel and the window. Technically, this operation is cross-correlation. For proper convolution, the kernel must be flipped along its horizontal and vertical axis before the dot product is calculated. The main difference between them is that convolution is commutative, whereas cross-correlation is not. Because this property tends to be unimportant in machine learning, the convention in machine learning literature and software is to call both of these operations convolution[36].

Each neuron in a convolutional layer has a kernel associated with it, and it is the values of this kernel that are the trainable parameters. The input to each neuron is the dot product of this

kernel and its receptive field in the preceding layer. As with fully connected layers, a bias is added to the input and it is passed through an activation function to produce some output.

In a fully-connected layer there is little structure to the neurons, though we tend to think of them as stacked in a column. In convolutional layers, by contrast, the shape of the layer is very important. Typically, they are stacked in a three-dimensional volume. Each neuron is connected only to a subset of the neurons in the preceding layer known as the neurons *receptive field*. See figure 4.6 for an illustration.

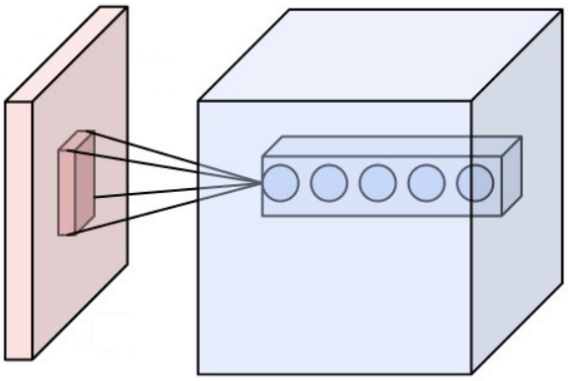

Figure 4.6: Two convolutional layers. The dark section of the red layer is the receptive field of the neuron in the blue layer.

Those neurons that are stacked in the depth dimension share a receptive field in the input volume, and all neurons in a depth slice (or *activation map*) share the same kernel. Adjacent neurons in the activation map have adjacent centers of their receptive fields in the input layer. This way, the output can be viewed either as the output of an ordinary neural network layer where each neuron is connected only to a subset of the previous layer and parameters are shared between neurons or as the output of convolving the previous layer with the weights of the neurons in each depth slice.

Consider as an example a convolutional layer connected to an input volume consisting of $32x32x3$ RGB images (the depth dimension is the color channel). If the layer contains five $3x3$ kernels, the layer will consist of $32x32x5 = 5120$ neurons. The output will be the results of convolving each kernel with the input layer. As the values of the kernels are the inputs, there are only $3x3x5 = 45$ trainable parameters. If each of the 5120 neurons were connected to each value in

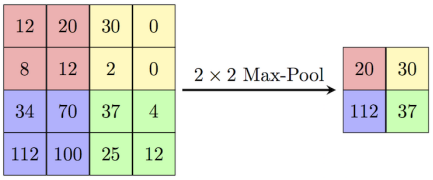

Figure 4.7: Illlustration of max pooling. Image by Cambridge Spark Ltd.

the input volume, as in a fully-connected layer, the number of trainable parameters would be $5120x32x32x3 = 15,728,640$.

An additional hyperparameter in convolutional layers is *stride*. This is the number of units the convolutional kernel slides along the input volume at each step or equivalently the distance between the center of receptive fields of adjacent neurons.

4.3.2 Max Pooling

To further reduce the size of the input volume, some form of *pooling* tends to be applied after the convolution. In this stage, the output volume is downsampled by dividing the output volume into non-overlapping rectangular windows and replacing them with a single value. The most common form of pooling is *max pooling*, where the maximum value in each window is used. This is illustrated in figure 4.7.

In addition to reducing the computational load by reducing the size of the input volume for the next layer, max pooling provides some invariance to small translations of the input volume.

4.3.3 CNN Architecture

In addition to the convolutional and max pooling layers, most CNNs also have at least one fully-connected layer. After a number of convolutional and max pooling layers, the input is transformed and down-sampled to create a representation of the data that can be classified using ordinary fully-connected layers. A typical CNN architecture is illustrated in figure 4.8.

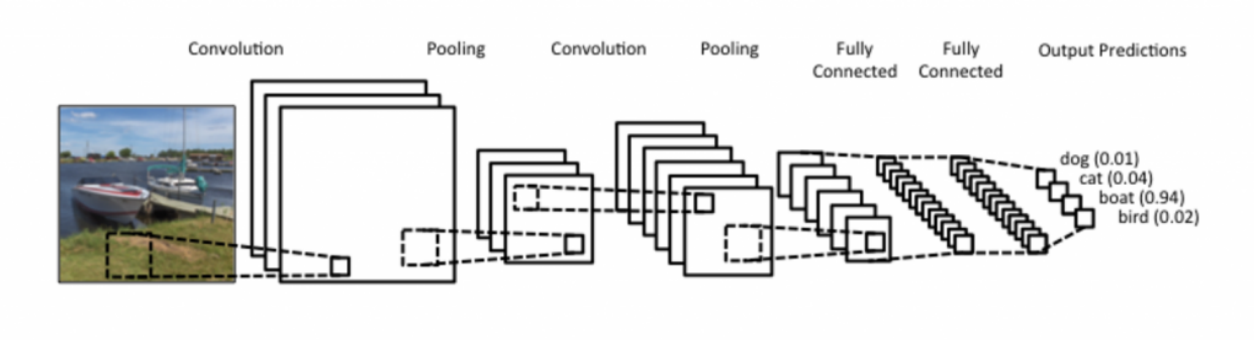

Figure 4.8: Typical CNN architecture for image classification

4.3.4 Interpretation of CNNs

Visual recognition is a task in which humans outperform most computer algorithms. In a matter of milliseconds a person can recognize whether they are looking at a cat or a dog, or identify a face. For this reason, CNNs are heavily inspired by the way the human brain processes visual information[37]. As noted, the biological neurons in the brain are not as densely connected as those in fully-connected neural networks. The structure of convolutional layers are especially inspired by those in the primary visual cortex.

Much of what is known about the visual cortex, also known as visual area 1 (V1) is thanks to the work of David Hubel and Torsten Wiesel. They discovered that the neurons in V1 are arranged in a grid, much like the neurons in a convolutional layer[28]. In addition, they demonstrated that all neurons in V1 are activated by a small region in the visual field. This region is the receptive field of the neuron, which the receptive fields in CNNs are inspired by and named after. As with artificial neurons, biological neurons that are close to one another in V1 have receptive fields that are close to one another in the visual field.

In addition to the biological interpretation of CNNs, they are interesting from the perspective of image processing and deep learning. In image processing, many of the methods of extracting features from images take the form of convolution of the image with a particular kernel. A simple way to perform edge detection is by convolving an image with the so-called Sobel operators. Equation 4.7 shows the Sobel operator G. Figure 4.9 illustrates the effect of convolving an image with the Sobel operator.

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \tag{4.7}$$

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +3 \\ 0 & 0 & 0 \\ -1 & -2 & -3 \end{bmatrix} \tag{4.8}$$

$$\mathbf{G} = \sqrt{G_x^2 + G_y^2} \tag{4.9}$$

Figure 4.9: Image before and after convolution with the Sobel operator

Other kernels can be used to extract other features. The goal of deep learning is to create methods which learn the best representations of the problem for classification, and learning which kernels to filter the input images with can be seen as a solution to this. If the edges in a picture are important for some classification task, the CNN might learn a kernel similar to the Sobel operator. In a classification task where edges are important, the CNN might learn a kernel for edge detection.

The purpose of the hidden layers in neural networks is to learn some way to transform on the data which makes it better suited for classification. By stacking these transformations, even more complex transformations can be learned. Hidden layers have have too many trainable parameters for it to be feasible to stack very many of them. CNNs, however, can stack more layers as the number of transformations is restricted in a way that is inspired both by the way

the brain processes visual information and the way hand-crafted image processing algorithms extract features from data.

Part II

Methods

Chapter 5

Data

5.1 Synthetic Data

The generation of synthetic data is handled by the make_synthetic_data.py script. As the purpose of time-frequency representations is to highlight the spectral properties of the data, it can be assumed that the method proposed in this thesis should be capable of detecting information "hidden" in the frequency and amplitude of the oscillatory modes of the signal.

On the basis of this, a signal lasting two seconds sampled at 100 Hz was created. The signal consists of three modes. Each mode has a frequency normally distributed around 5, 17 and 29 Hz, respectively. For one of these modes, a temporary increase in power occurs to simulate some "event" at this frequency band. The center of this pulse of normally distributed around the middle of the signal. The signal is also subject to white noise.

Figure 5.1 shows an example of a synthetic signal generated for this thesis. This example signal is of the class of signals with an amplitude increase in the highest frequency mode.

A synthetic data set of 10,000 labeled samples was created for this thesis.

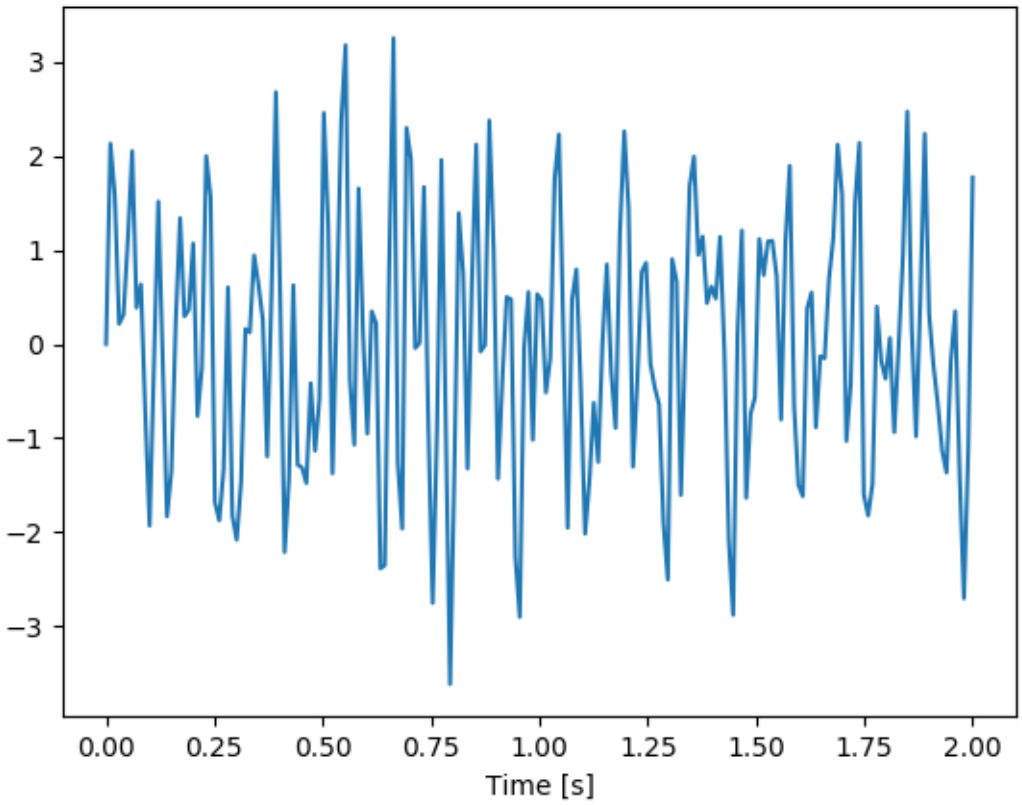

Figure 5.1: Example of synthetic signal. The "event" is in the highest frequency mode.

5.2 EEG Data

In this thesis, the data set IVa from BCI competition III was used[44]. The BCI competitions are competitions organized by Berlin Brain Computer Interface (BBCI) which consists of a number of BCI problems with associated data sets, generally involving the classification of electroencephalography (EEG) data segments. It should be noted that while the problem in the original competition involved training on only a small subset of the data. As the amount of training data provided is too small for what is generally recommended when working with Convolutional Neural Networks (CNNs), this constraint was relaxed.

5.2.1 Experimental Setup

Data set IVa contains a continuous EEG measurements with 118 channels from taken from five healthy subjects. The subjects sat on a comfortable chair with armrests during the recording sessions. They were given visual cues lasting for 3.5 seconds where they were asked to imagine movement either in the right hand or in the left foot. The presentation of the visual cues were intermittent with intervals of lengths between 1.75 and 2.25 seconds.

There were two different types of visual stimulation: either letters appearing behind a fixation cross or a randomly moving target indicating the type of action to be performed. The first type has the advantage of being simpler, and thus probably producing less noise. The noise that is created, however, might be the cause of target-correlated eye movement. The second type probably causes more noise, but this noise will be target-uncorrelated, thus lessening the probability of the machine learning algorithm "learning the noise". Each of the subjects were shown 280 cues, for a total of 1,400 objects in the data set.

5.2.2 Format and Preprocessing

Preprocessing was done in the preprocess.py script. This script uses the Numpy and SciPy libraries, as well as the custom implementation of Hilbert-Huang Transform (HHT) described in

section 6.

The data is provided by the competition organizers in either 100 or 1000 Hz. The 100 Hz version was used in this project, as the higher resolution version would probably lead to unacceptably slow preprocessing and too large input volumes.

The data was presented as either .txt or .mat files. The .txt files were used for the data, as .mat files are not suited for Python. A complete set of labels was only available in .mat format, so these were processed in this format. This script, for each of the five subjects, begins by parsing the text files to create a numpy array of the continuous EEG data. Once this is done, the script iterates through a list of indices indicating the starting point of an activity with a corresponding label. From this, a segment of data is identified and transformed using one of the three methods described in chapter 3. Once all the data is processed, the script saves the processed data as a four dimensional numpy array (the dimensions being number of objects, width, height and channel, respectively) in the .npy format used by numpy.

All the 118 channels were not used, as this would make the objects too large to work with. Instead, all but 8 of the 118 sensors were used. To get a reasonably representative selection, the sensors at Fp1, Fp2, F3, F4, C3, C4, O1 and O2 were used.

Chapter 6

Signal Processing

6.1 Short-time Fourier Transform

For the purpose of this thesis, the implementation of Short-time Fourier Transform (STFT) provided by the SciPy library was used[52]. This implementation takes the signal itself, the sampling rate, the number of samples in each segment and the number of overlapping samples in two consecutive segments.

For the work done for this thesis, the number of samples per segment was set to 32, corresponding to 32ms segments. This was chosen at it is a power of two, which is advantageous for computational reasons as the algorithm uses the Fast Fourier Transform (FFT), and gives a reasonable compromise between time and frequency resolution. For the window function the Hann window, shown in figure 6.1, was used.

The frequency resolution is chosen automatically to conform to the Nyquist frequency and the Heisenberg-Gabor limit. With a time resolution of 32 ms we have a frequency resolution of 3.125 Hz. With a sampling rate of 100 Hz, the Nyquist frequency limits the frequency to 50 Hz. Thus we produce spectrograms of volume 23x17x8, corresponding to time, frequency and channel, respectively.

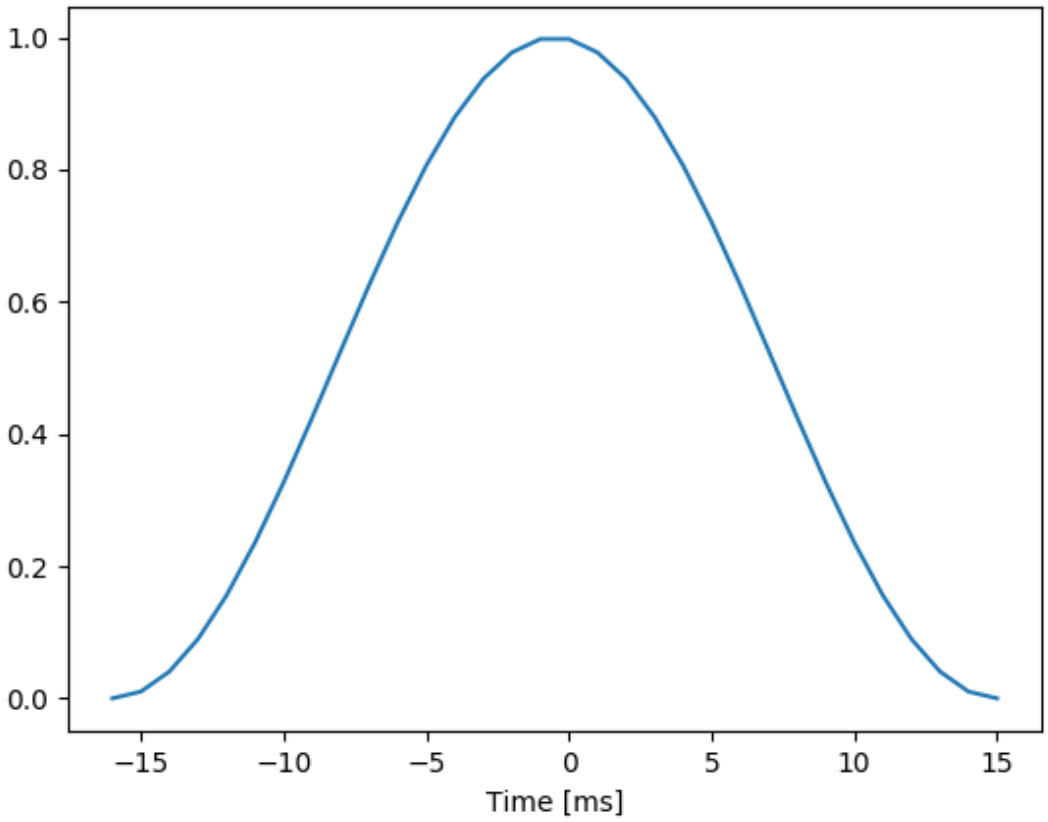

Figure 6.1: The Hann window function

6.2 Continuous Wavelet Transform

The SciPy implementation of Continuous Wavelet Transform (CWT)[51] was used as well. This method takes as input the signal, the mother wavelet to be used and a list of scales in which the signal should be examined.

The mother wavelet used for the work in this thesis is the Ricker wavelet seen in figure 3.8. Scales between 1 and 50 were used, with a resolution of 0.5. The result of this had volume 349x100x8, corresponding to time, scale and channel.

6.3 Hilbert-Huang Transform

An Hilbert-Huang Transform (HHT) library was implemented in Python for the purpose of this project. This section discusses the challenges involved in implementing HHT, and the choices made for this implementation.

The library consists primarily of implementations of Empirical Mode Decomposition (EMD) and Hilbert Spectral Analysis (HSA), as well as functions for visualizing the results of the EMD.

6.3.1 Interpolation

The upper and lower envelopes of a function are not mathematically defined, and must be defined algorithmically.

The envelopes should "hug" the signal as closely as possible by interpolating on the upper and lower envelopes. There are two problems to solve here: computing the extrema and interpolating between them.

The simplest method of computing extrema involves classifying each sample based on their two neighbouring samples; it both are lower, we classify the point as a maximum. If both are greater, we classify the sample as a minimum. The problem with this method is that the precision of the extrema calculations is very important for the algorithm, and for discrete data it is quite likely that the real extrema are located between sampling points. This, however, is a minor concern. The simple method was therefore used.

The second issue is how the extrema should be used to interpolate the envelopes. Huang et al.[27] suggests using *cubic spline* interpolation, which constructs the envelope by connecting cubic polynomials which are piecewise interpolated on small intervals. Experiments by Rilling et al.[50] empirically justifies this method of interpolation over other methods like B-splines. Because of this, cubic interpolation is used to determine the envelopes.

6.3.2 Boundary Effects

When computing the upper and lower envelope, the borders should not be ignored. Consider the signal in figure 6.2.

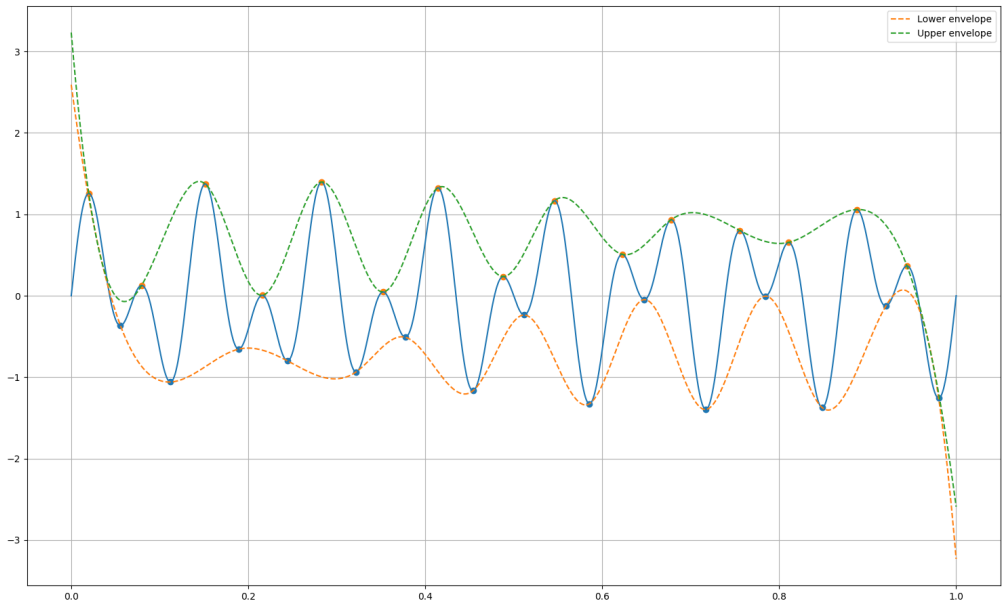

Figure 6.2: Signal with envelopes when boundaries is ignored

Here, the envelopes are quite good for most of the signal, but fail near the boundary. This is due to the fact that the samples between the first (or last) extrema and the boundaries are ignored completely. A number of methods have been proposed to handle this[48][40]. The method used in this implementation is based on linear interpolation of the two first or last extrema.

We consider a line found by interpolation using the two last or two first extrema, and calculate the value of this line at the end (or beginning) of the signal. If this value is higher (for the upper envelope) or lower (for the lower envelope) than the value at the boundary, this point is used in the calculation of the envelope. Otherwise, the boundary point itself is used. See figure 6.3 for an illustration.

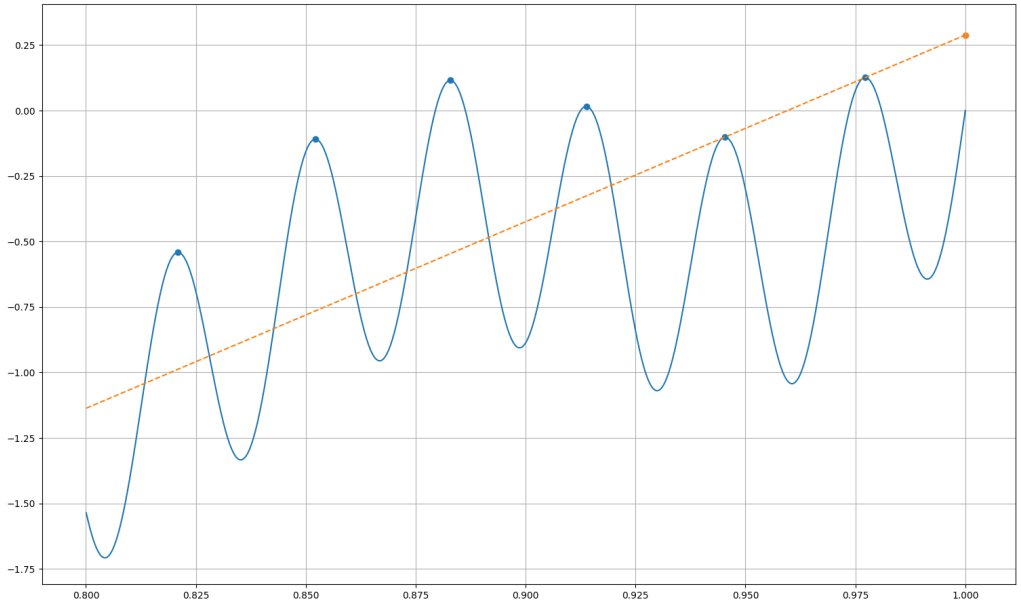

Figure 6.3: Linear interpolation to find the point at the end for the upper envelope. As the end point of the line is higher than the end point of the signal, the point on the line is used together with the maxima to calculate the upper envelope

6.3.3 Stopping Criteria

As mentioned in chapter 4, one of the challenges when implementing the sifting algorithm is deciding when to stop the sifting. The purpose of the sifting process is two-fold: It should remove riding waves (i.e. other, lower frequency, modes of oscillation) from the signal and force the mean to zero. There is, however, an unfortunate side-effect of the sifting process: It evens out amplitude variations of the oscillatory mode which may contain important information about the underlying physical process. This reveals a trade-off. To eliminate the riding waves and to force a local zero, sifting as many times as possible is needed. On the other hand, too many sifting steps will reduce the Intrinsic Mode Function (IMF) to be a constant-amplitude frequency-modulated function, which would obliterate the intrinsic amplitude variations and render the results physically less meaningful[31][26].

A number of alternative stopping criteria have been proposed. The simplest method is to just fix

a constant number of iterations. The obvious advantage of this method is that it is very simple and often gives reasonable results in practice. The obvious disadvantage is it's non-adaptive nature; the optimal number of siftings is not the same for every signal.

The method originally proposed by Huang et al.[27] involves a Cauchy-type convergence for the IMF. If we denote the kth iteration of the sifting process as $h_{1k}(t)$ we compute the standard deviation SD as

$$SD = \sum_{t=0}^{T} \left[\frac{|h_{1(k-1)}(t) - h_{1k}(t)|^2}{h_{1(k-1)}^2(t)} \right] \tag{6.1}$$

And stop the sifting process once SD goes below a given tolerance ϵ (Huang et al. proposes $0.2 \leq \epsilon \leq 0.3$).

The problem with this method is that it is not closely related to the definition of the IMF, and as such does not guarantee that the output is valid.

A method more closely connected with the definition of the IMF was proposed by Huang et al. (1999)[25]. This method involves stopping the sifting process when the number of zero crossings and extrema have remained the same for S number of siftings.

The stopping criterion in the implementation used for this thesis uses the S-number criterion as well as the requirement that the mean is within some tolerance ϵ of the zero mean for all time. For this implementation, $S = 5$ and $\epsilon = 0.001$. In addition, an upper limit of 200 siftings is set, to ensure that the algorithm eventually terminates for all signals.

6.3.4 HSA Implementation

While the HHT in theory produces continuous values for frequency, we do in practice have to divide the frequency spectrum into "bins" of frequency. The *widths* parameter of the "hsa.hht" function defines these bins. This parameter should be a list of values which defines the limits of the bins. In this project, frequencies between 1 Hz and 50 Hz were considered. The width of each bin is 0.5 Hz, meaning that a total of 100 bins were used.

With these parameters, the volume of the Hilbert spectra produces by the preprocessing script is 350x100x8, corresponding to time, frequency and channel, respectively.

Chapter 7

Machine Learning

The machine learning portion of the project was implemented using the Keras library with the Tensorflow library as a backend. Four different Convolutional Neural Network (CNN) architectures were tested. While fairly similar in the overall structure and design, they differ in depth and width. This section outlines the choices made in the design of these networks.

7.1 Architecture

Four different architectures were used for the work presented in this thesis. A design consisting of a number of convolutional layers together with max pooling layers, followed by one or two fully connected layers was used. The variations in architecture involved using either four or eight convolutional layers and either one or two fully connected layers followed by a final classification layer. As an example, the first architecture, consisting of four convolutional layers, two max pooling layers and a single dense layer, is shown in figure 7.1. The remaining architectures are shown in appendix A.

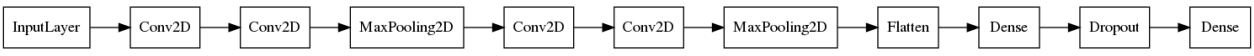

Figure 7.1: Architecture 1

7.2 Activation

Historically, sigmoid functions like the one shown in figure 4.5 has been used as activation functions due to their similarity to the activation of biological neurons. A different activation function that has become popular in recent years is the Rectified Linear Unit (ReLU), introduced by Hahnloser et al. as a biologically plausible activation function[20]. The ReLU function is defined as $f(x) = \max(x, 0)$. A plot of this function for $-1 \leq x \leq 1$ is shown in figure 7.2

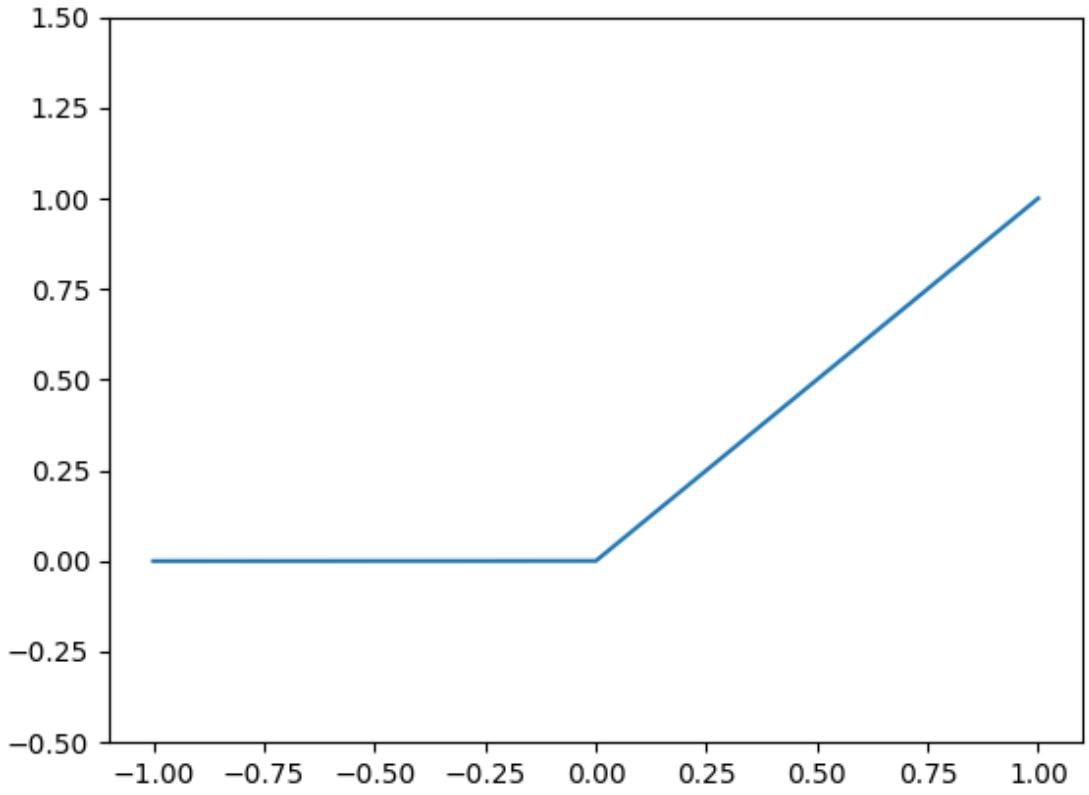

Figure 7.2: The ReLU function

While this function is not differentiable at $x = 0$, this is rarely a problem in practice, as it is continuously differentiable at every other value. The function has been wildly successful in practice and is, as of 2018, the default choice of activation function in neural networks[47]. It is used for both the convolutional and the dense layers for the work in this thesis.

For the classification layer, the *softmax* activation function is used. Assuming a layer has K neurons and input vector (i.e. the weight sum of outputs from the previous layer plus bias) z, this function is defined as in 7.1, for $j = 0, .., K$.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_i}} \tag{7.1}$$

The effect of this function is essentially "squeezing" the input into a vector whose components are between 0 and 1 and which sums to one. Thus, the components of this vector can be viewed as the network assigning a probability estimate of the object belonging to each class.

7.3 Loss Function

The Mean Squared Loss (MSL) function introduced in chapter 4 is intuitive, but is not necessarily the best loss function for neural networks. It is generally best suited for regression problems, where a function which maps to a continuous variable must be fitted as best as possible. Consider the problem of fitting a line to a set of points. Using the MSL as a loss function, the best model will be the one that minimize the distances between the line and the points, even if it *correctly* classifies (i.e. run through) very few or none of the points.

The problem of classifying electroencephalography (EEG) segments, however, is "all-or-nothing" in that a classification is either correct or incorrect. Because of this, the networks in this thesis use Cross Entropy Loss (CEL). Given two discrete probability distributions p and q, the cross-entropy between them is defined as

$$H(p, q) = -\sum_i p(x_i) \log q(x_i) \tag{7.2}$$

Intuitively, if p is some true distribution and q is our assumption of the distribution, cross-entropy measures how wrong this assumption is[10].

If $\mathbf{y}$ is taken to be the one-hot encoded label (i.e. a vector which is one for the component corresponding to the correct class and zero for all other components) and $\hat{\mathbf{y}}$ to be the output of the

softmax-activated last layer, these two vectors are essentially two probability distributions of the classification. As the learning data is labeled, the "true" distribution is one that assigns probability 1 to the correct class and 0 to the others. As mentioned above, the output of a softmax-activated layer is essentially a probability distribution assigning a certain probability estimate of the object belonging to each class.

If class j is the correct classification, $y_j = 1$ and $y_{i \neq j} = 0$, so the CEL with weights and biases θ and output $\hat{\mathbf{y}}$ becomes

$$\mathscr{L}(\theta) = -\sum_{i=1}^{K} y_i \log \hat{y}_i = -\log \hat{y}_j \tag{7.3}$$

In addition to the theoretical justification, empirical studies by Kline and Berardi (2005)[33] has demonstrated the superiority of CEL over MSL.

7.4 Dropout Regularization

As mentioned in chapter 4, overfitting is a common problem in machine learning, particularly neural networks. One method of reducing the effect of overfitting is dropout regularization, a method first introduced by Srivastava et al. in 2014[55].

Dropout regularization involves randomly dropping units, as well as their connections, from the network during each training step. With 50% dropout on a particular layer, each of the units in that layer has a 50% chance of being deactivated at each training step.

The theory behind dropout regularization is that one should be able to counteract overfitting by training networks of different configurations and using their average output for the prediction. This, however, is infeasible for complex architectures.

Dropout allows us to approximate the average of exponentially many networks. With a 50% dropout rate, the network is trained with a slightly different architecture at each step, and the final effect is that the network ends up as an average of all these different architectures. Experimental results of Srivastava et al. demonstrated the promise of this method in counteracting overfitting[55]. It should be noted that dropout is *only* applied during the training stage. For

testing and predictions, all the units are used.

For the convolutional layers, however, the spatial relationships between adjacent nodes are very important and deactivating half the units in a convolutional kernel might cause nonsensical outputs[17]. As such, dropout fails to accurately approximate the average of many networks. For this reason, dropout is only applied on dense layers in the networks utilized for this thesis.

7.5 Optimization

For optimization, the *Adam* optimizer, which was devised by Kingma and Lei Ba in 2004[32], is used. This optimization algorithm is an extension of the usual Stochastic Gradient Descent (SGD) optimizer.

This algorithm works by maintaining a moving estimate of the mean and variance of the gradient of the objective function. As hyperparameters, the algorithm requires a learning rate α and well as decay rates β_1 and β_2, for the mean and variance respectively. Kingma and Lei Ba recommends the default values $\alpha = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. In addition, some initial estimate of the parameters (weights and biases) θ_0 is required. For the networks in this thesis, the weights and biases are randomized at start.

At each step t, the gradient of the objective function is estimated by backpropagation using a given batch size as with SGD. We denote this estimate as g_t. For this thesis, a batch size of 32 samples is used. This is used to update a running estimate of the mean and variance as

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{7.4}$$

$$v_t = \beta_2 v_{t_1} + (1-\beta_2) g_t^2 \tag{7.5}$$

These are then bias-corrected as

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{7.6}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{7.7}$$

Finally, the parameters are updated

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \tag{7.8}$$

The apparent effect of this is that the learning rate is adaptive to the problem at hand, as opposed to traditional SGD where the learning rate is fixed.

7.6 Training and Validation

The most common measure of success in any machine learning problem is accuracy, i.e. how many of the objects in the data set were correctly classified. A machine learning algorithm might achieve a very high accuracy on the training set while being unable to generalize to new data, and for this reason some method of validating the algorithm is necessary.

To accurately ascertain the performance of any machine learning method, some form of validation has to be performed. This is generally done by splitting the data set into a training set, on which the data is trained, and a testing set which is not touched during the training period, but on which the network is tested on after training is done. The data set can be split in any number of ways. It must be large enough to accurately represent the data, but not so large that we are left with too little training data. A common split is using 80% for training and 20% for testing[18].

Because the data set examined in this thesis is fairly small, a train/test split of the data set may leave too little data either for good training or for accurate testing. Because of this, a more advanced set of validation techniques known as *cross-validation* is employed. These methods involve repeatedly training the network on different subsets of the training data. The most common method, and the one used to evaluate the networks in this thesis, is known as *k-fold cross-*

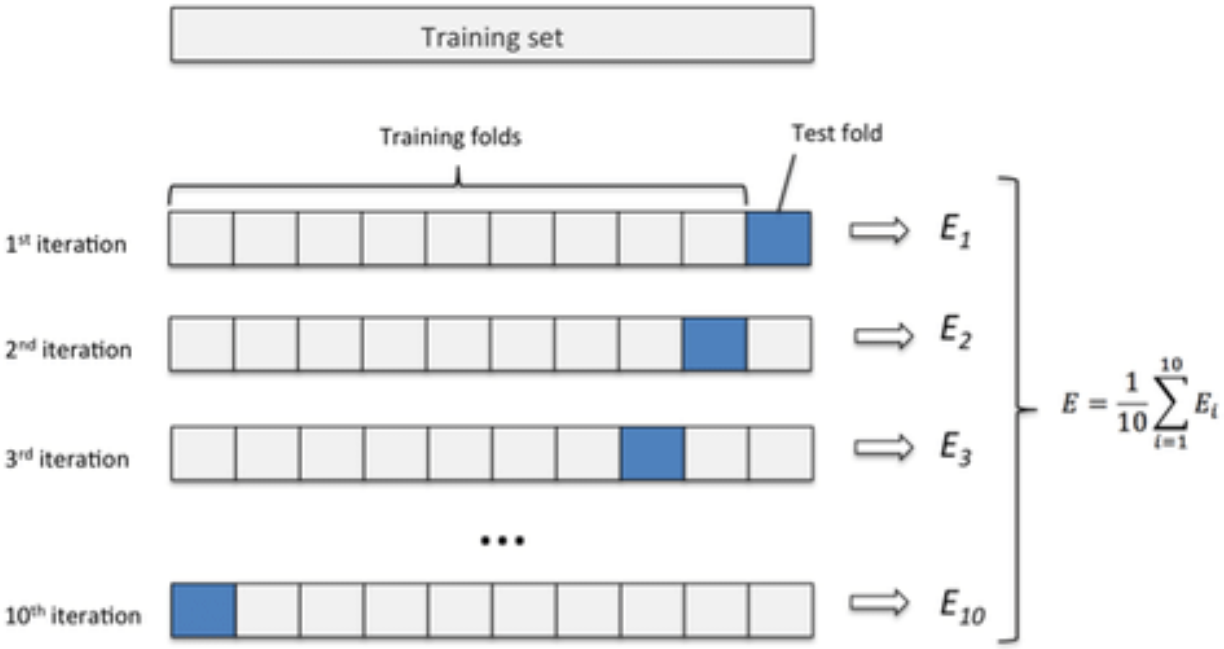

Figure 7.3: Figure illustration k-fold cross-validation with $k = 10$. Each of the k partitions are used as a test set with the remaining $k-1$ partitions used as a training set. Image from PhD thesis by Jaskiret Dhindsa[12]

validation[18]. First, the data set is divided into k different subsets. For each of these subsets, a full training session is performed using the other $k-1$ subsets as the training set. The network is then validated on the subset in question. The final accuracy is taken to be the average of the accuracy on each of the subsets. See figure 7.3.

Stratified k-fold cross-validation is used in this thesis. "Stratified" simply means that each of the k partitions contain roughly the same amount of samples from each class. In choosing a value for k, the advantage of higher values is obvious: retaining more data for training. The disadvantage is that higher values of k significantly increases the time required for training. Additionally, higher values of k is associated with higher variance of the accuracy estimate, as the training sets are smaller and therefore less likely to be representative samples of the data.

In this thesis $k = 5$ is used for the synthetic data, retaining 20% of the data for validation. As the EEG dataset is quite small, $k = 10$ is used.

Part III

Results and Discussion

Chapter 8

Signal Processing

8.1 Synthetic Data

The three time-frequency representations of the signal in figure 5.1 are plotted in figure 8.1-8.3.

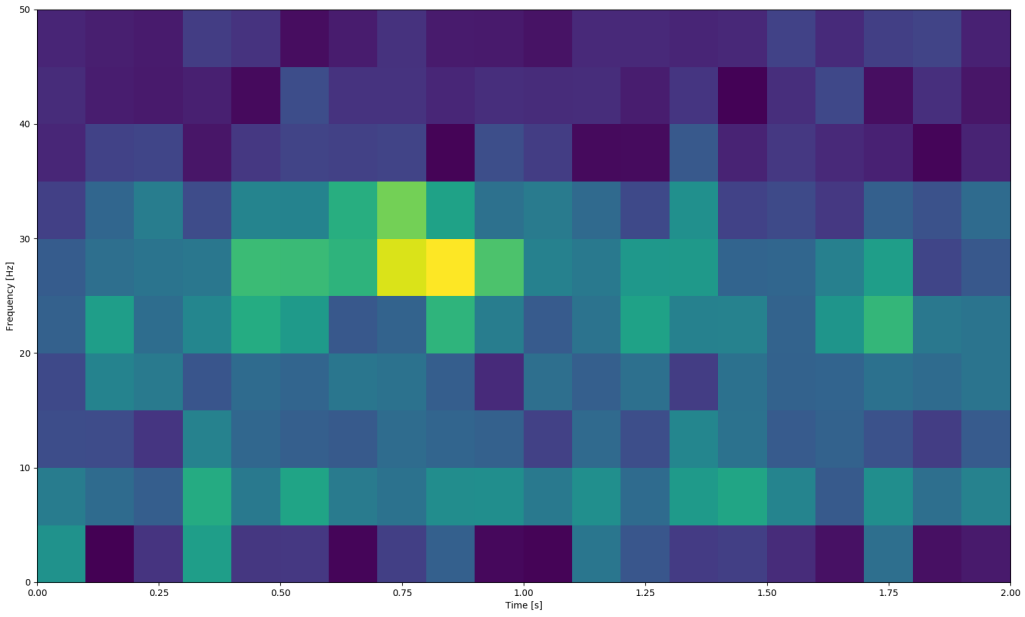

Figure 8.1: Spectrogram of signal in figure 5.1

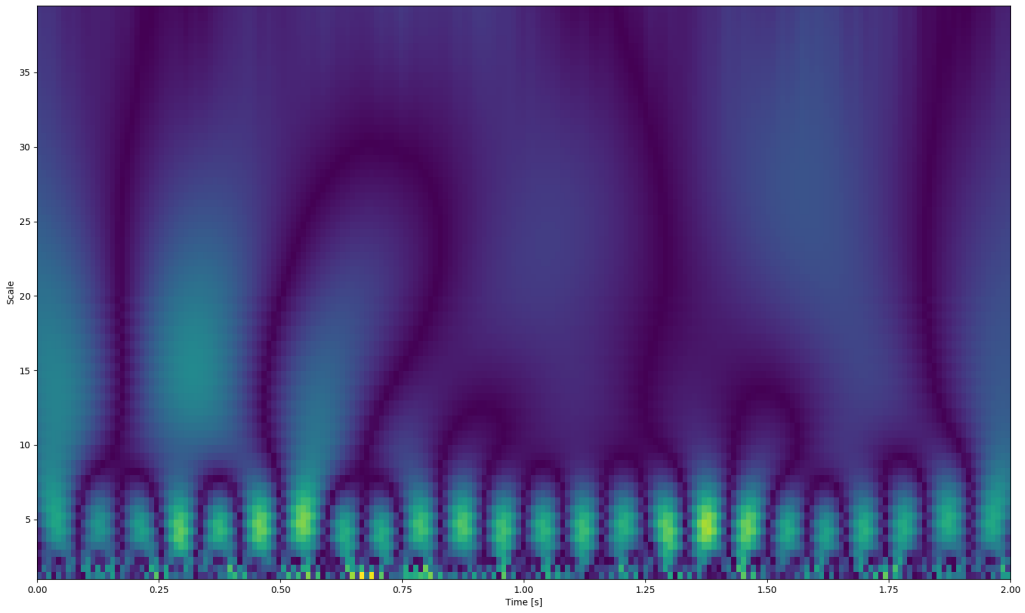

Figure 8.2: Scalogram of signal in figure 5.1

The increase in power near 27 Hz is immediately identifiable in figure 8.1. It is difficult to identify any increase in power in figure 8.2 by visual inspection. This scalogram appears to be distorted by edge effects, as there is an increase in power near the edges that is difficult to explain from the data.

The Hilbert spectrum, unfortunately, does not seem to seperate the three modes cleanly. This may be due to the presence of white noise in the signal, as susceptibility to noise is a well known challenge with HHT[2]. Even so, an increase in instantaneous amplitude is clearly visible near 27 Hz, so the Hilbert spectrum does represent the necessary information quite well.

8.2 Real Data

Average computational time for each of the transforms applied to the real-world data during the preprocessing stage is shown in table 8.1. The program was run on computer with an Intel Core i7-4770 CPU running the Ubuntu 18.04 LTS operating system. The preprocessing script was ran

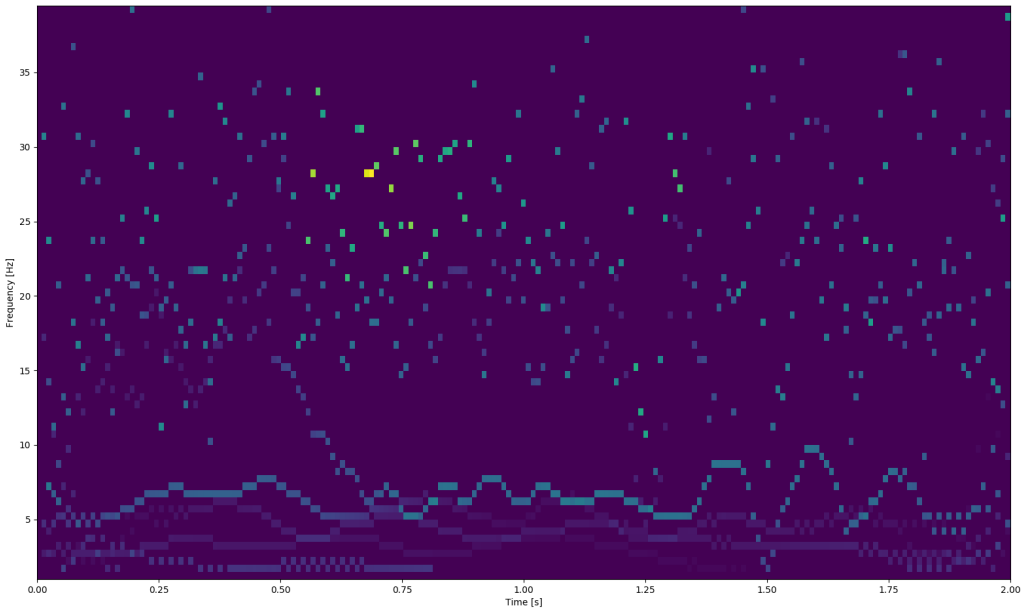

Figure 8.3: Hilbert spectrum of signal in figure 5.1

using Python 3.6.5 with Numpy version 1.14.2 and SciPy version 1.0.1.

The Hilbert-Huang Transform (HHT) is *significantly* slower than the other transforms, in contrast with what one would expect from the results shown in table 3.1. It should be noted that table 8.1 compares an unoptimized custom implementation of the HHT with more mature implementations in the SciPy library, which may explain the discrepancy.

Transform	Average Computational Time
STFT	0.0011 s
CWT	0.0873 s
HHT	4.4928 s

Table 8.1: The computation time of each of the transforms in this thesis applied to 3.5s EEG segments

As an example of the data, an sample recording is shown in figure 8.4. The three representations of this particular segment are shown in figure 8.5-8.7.

In all three representations, a spike in power is visible near the beginning of the segment, between 0.5 and 1.5 seconds on the lower end of the frequency spectrum. Of the three, the scalo-

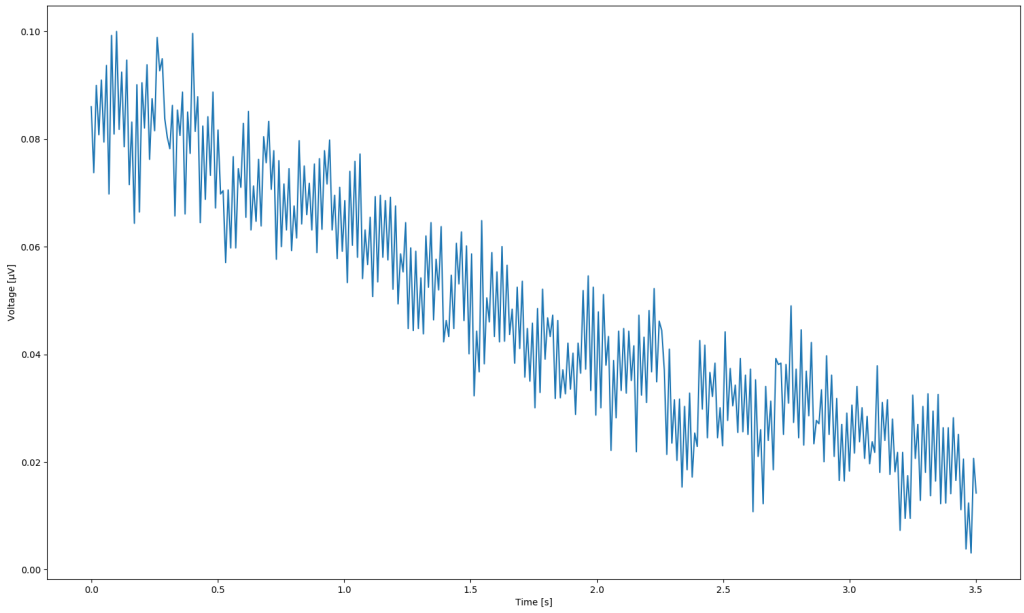

Figure 8.4: Example of an electroencephalography (EEG) recording. This shows the recording at the Fp1 sensor location from cue #141 of subject 'aa'. This segment is labeled as corresponding to mental imagery of right foot movement

gram is again the one that is most difficult to interpret.

Looking at the Hilbert spectrum, the HHT seems to have separated the signal into a series of identifiable modes at the lower end of the frequency spectrum, giving a more detailed picture of the EEG signal than the spectrogram at these frequencies. At higher frequencies the HHT again seems to fail. As with the synthetic data, it is likely that noise interferes with the actual brain waves at higher frequencies, causing the empirical mode decomposition to fail. 8.4 show the Empirical Mode Decomposition (EMD) of the signal and the power spectra of its Intrinsic Mode Functions (IMFs) is shown in figure 8.9.

The power spectra of the IMFs reveal a few things: the first IMF consists of an almost pure 50 Hz tone. The obvious interpretation of this is that this mode corresponds to 50 Hz AC power-line noise. For the remaining IMFs, the power spectra contain multiple peaks and overlapping frequencies. This is worrying, as it is an indicator of mode mixing. Attempts were made to implement the methods suggested in chapter 3 to handle mode mixing, but the Ensemble Empirical

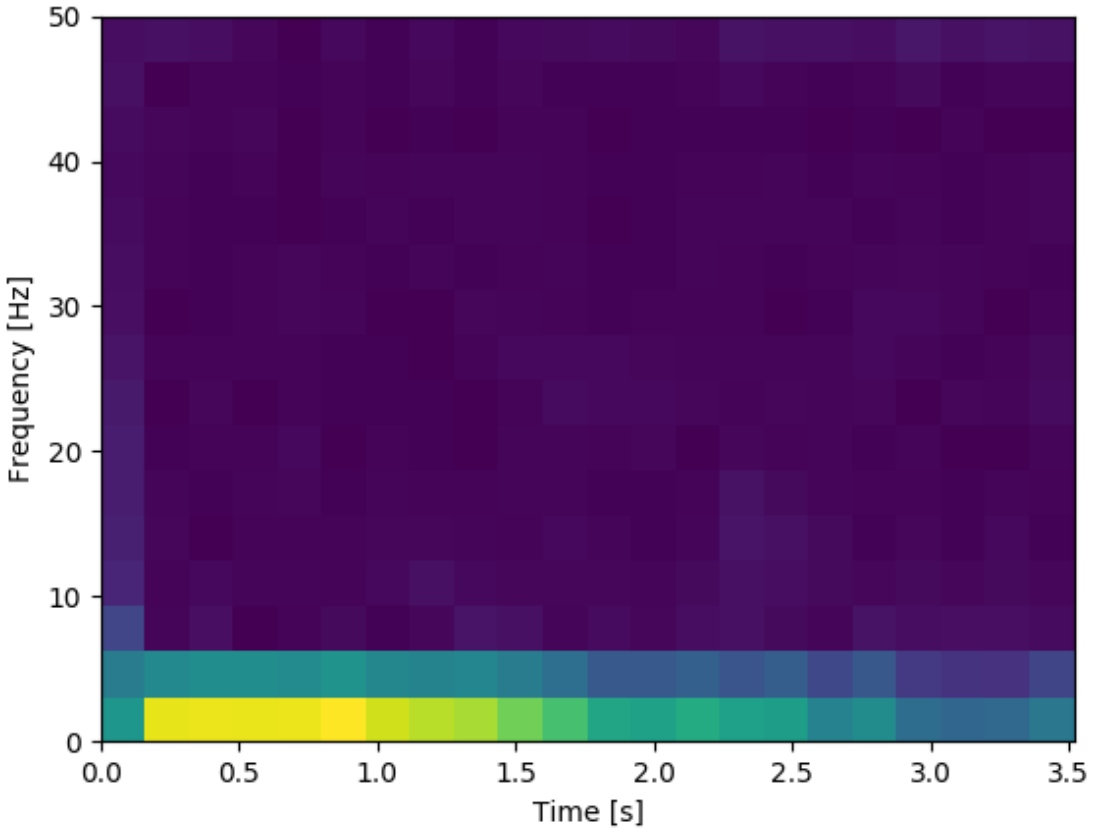

Figure 8.5: Spectrogram of the signal in figure 8.4

Mode Decomposition (EEMD) proved unacceptably slow, and I did not manage to implement the masking signal method in the general case.

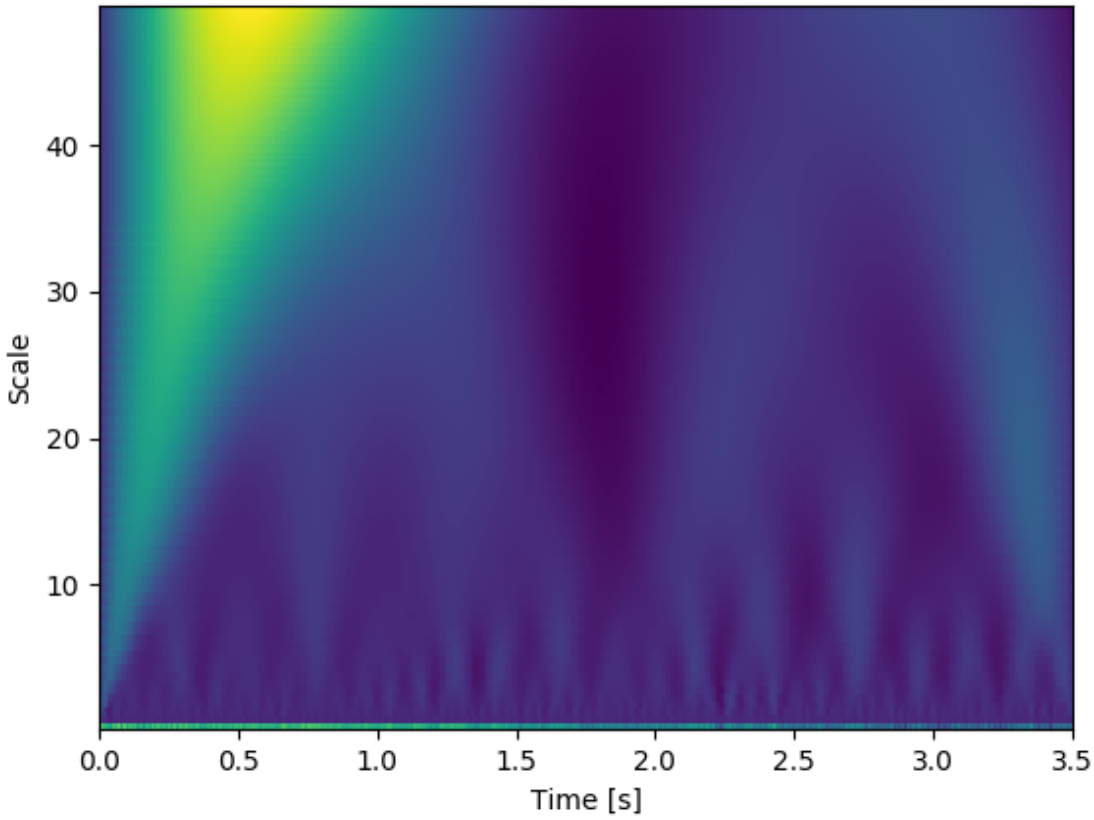

Figure 8.6: Scalogram of the signal in figure 8.4

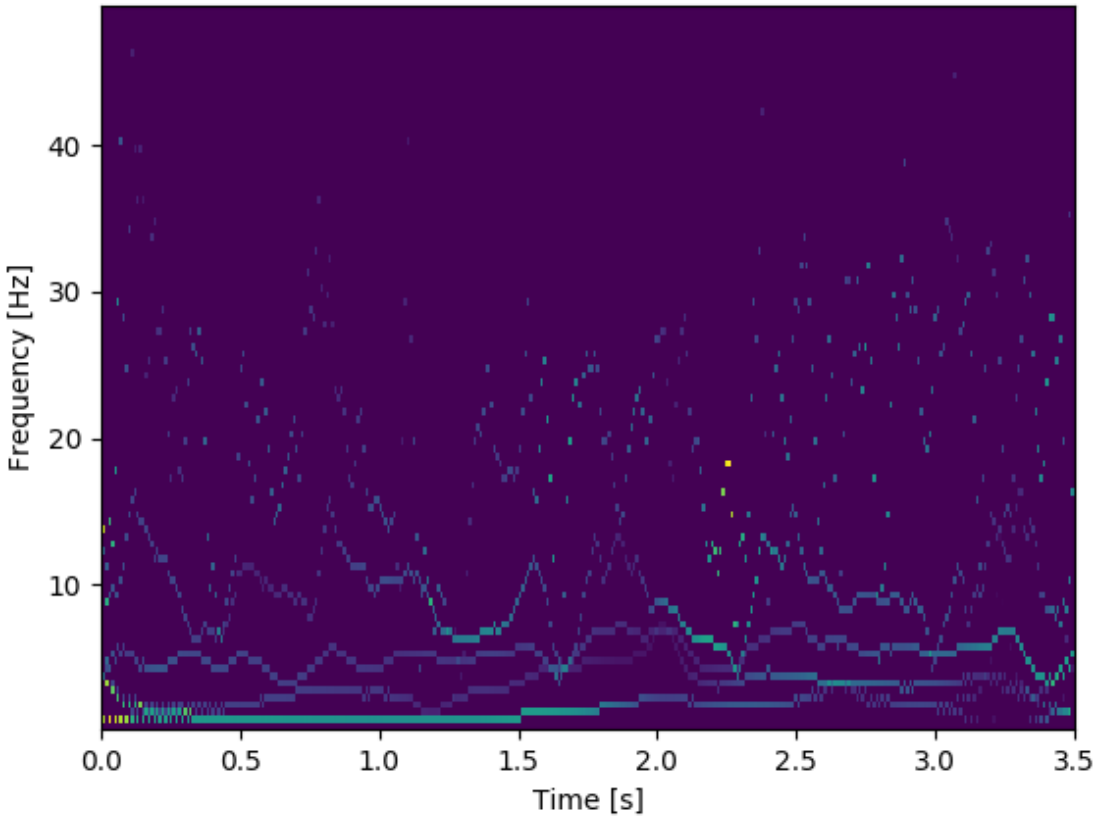

Figure 8.7: Hilbert spectrum of the signal in figure 8.4

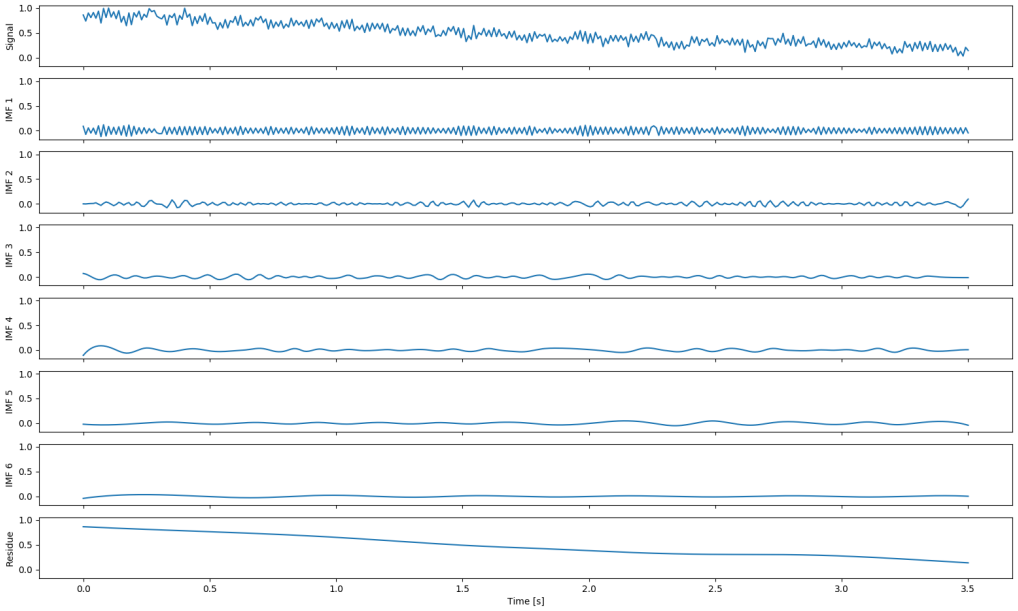

Figure 8.8: EMD of the signal in figure 8.4

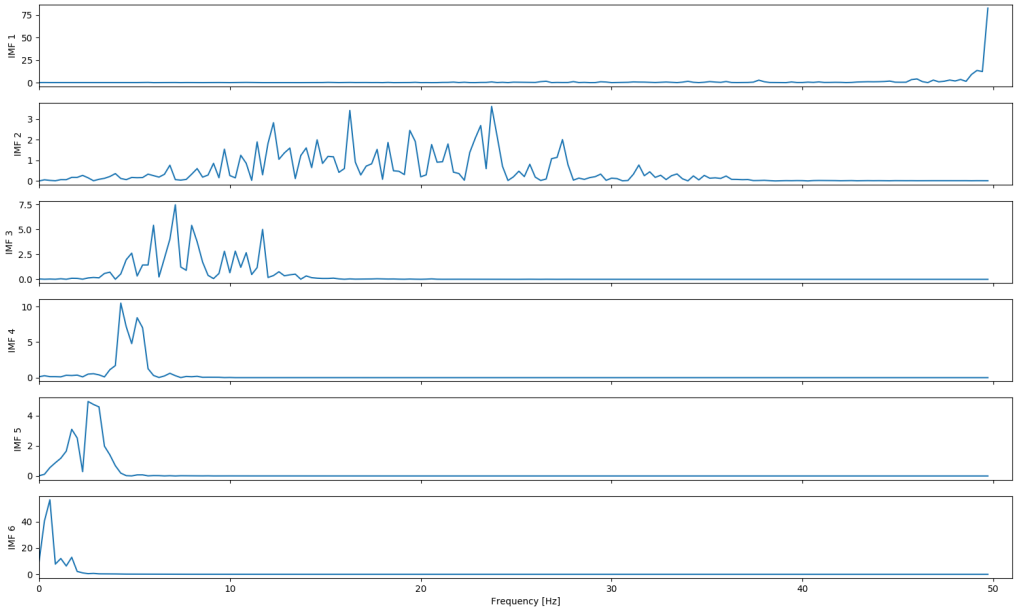

Figure 8.9: Power spectra of the IMFs in figure 8.8

Chapter 9

Training on Synthetic Data

Each combination of time-frequency representation and CNN architecture was evaluated using 10-fold stratified cross-validation. The resulting mean and standard deviations of accuracy are summarized in table 9.1. Each training session used three epochs.

	STFT	CWT	HHT
Architecture 1	**98.30% (+/- 0.36%)**	59.29% (+/- 5.59%)	88.19% (+/- 0.40%)
Architecture 2	98.00% (+/- 0.37%)	48.94% (+/- 1.30%)	87.29% (+/- 0.94%)
Architecture 3	N/A	49.74% (+/- 1.60%)	86.92% (+/- 2.38%)
Architecture 4	N/A	45.32% (+/- 2.23%)	86.27% (+/- 1.03%)

Table 9.1: Comparison of classification accuracy for each architecture and representation

The summary of these results is that all combinations of time-frequency representation and CNN architecture led to classification results better than what one would expect from random guessing (33.33%). Using the spectrogram representations led to noticably better results than using the Hilbert spectrum representations, both of which outperformed the CWT scalograms by a significant margin. The choice of architecture had less of an impact than the choice of representation, though the architectures with fewer convolutional layers tended to outperform those with more.

Both of these results can be explained by noting that the problem itself is quite simple. It is likely that the increased frequency resolution gained from representing the data as scalograms or Hilbert spectra did not improve on the learning process, but rather introduced unnecessary

and noisy additional information. The performance of the classifier on the scalograms are particularly poor, suggesting that scalograms might not be a good representation of signals when using CNNs. The results from chapter 8 also reveal that while the STFT produces a very good representation of the data, the CWT and HHT seem to struggle, which may explain some of the advantage of the spectrogram classification.

Figure 9.1-9.3 show the accuracy of classification on the training data and the test data as a function of samples used for training. To generate these, the dataset was divided into 10 batches, of which 20% was used for testing. Architecture 1 was used, as this had the best performance on all the data sets. There is no evidence of overfitting in any of these graphs.

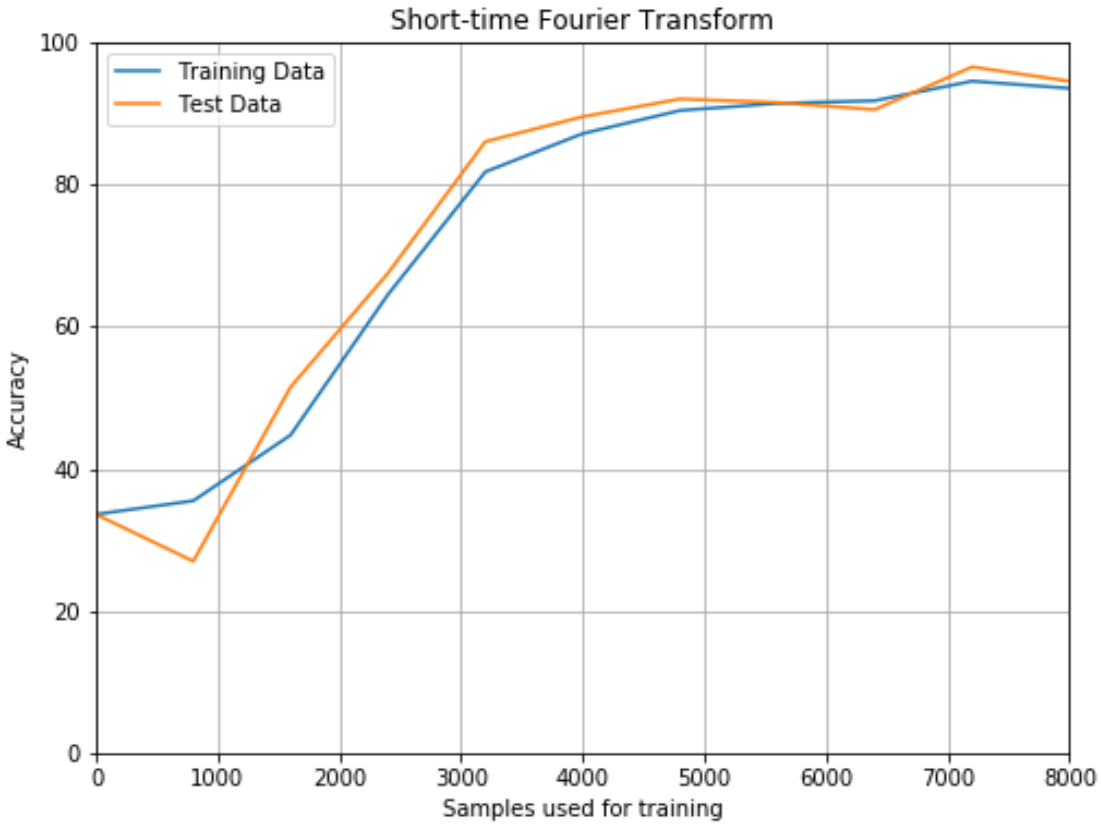

Figure 9.1: Accuracy for synthetic spectrogram classification

The performance of the network on spectrograms and Hilbert spectra were perfectly adequate. The performance with the scalograms, however, were very poor. Because of this, another more

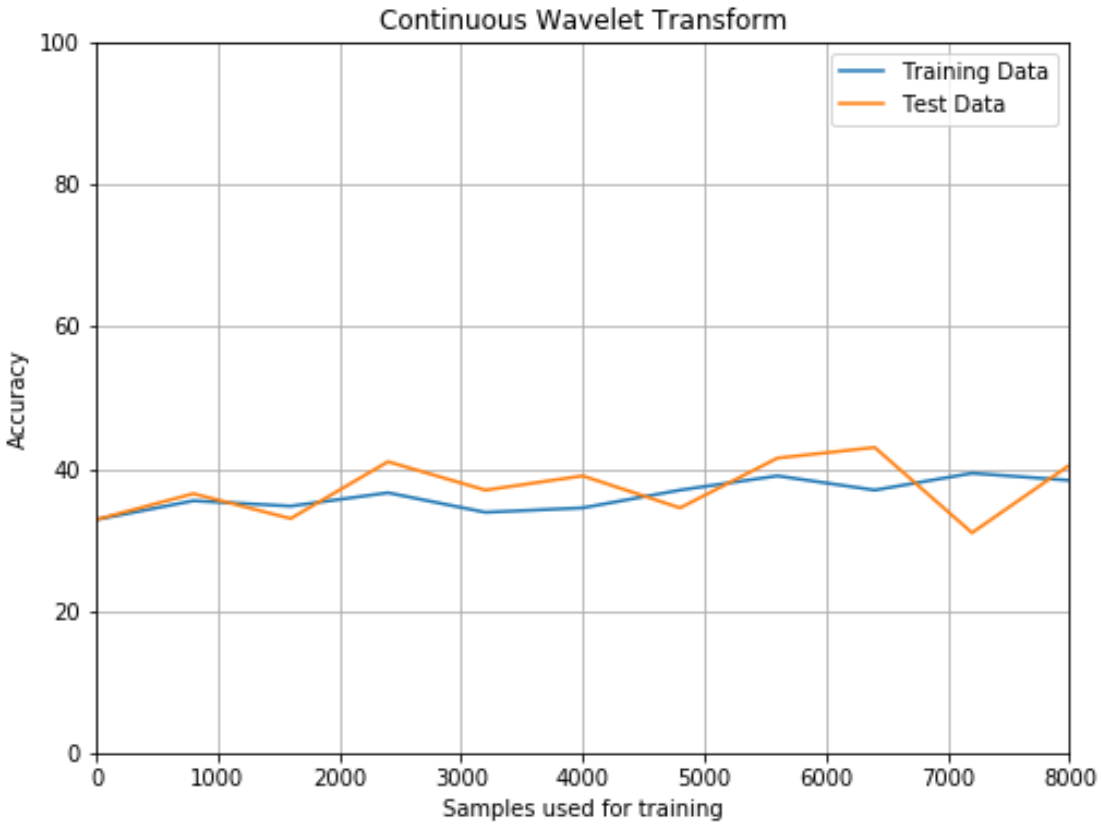

Figure 9.2: Accuracy for synthetic scalogram classification

aggressive training session was recorded. The results are illustrated in figure 9.4. Here, each batch was trained for five epochs. This did significantly improve the results, but at the expense of introducing significant overfitting.

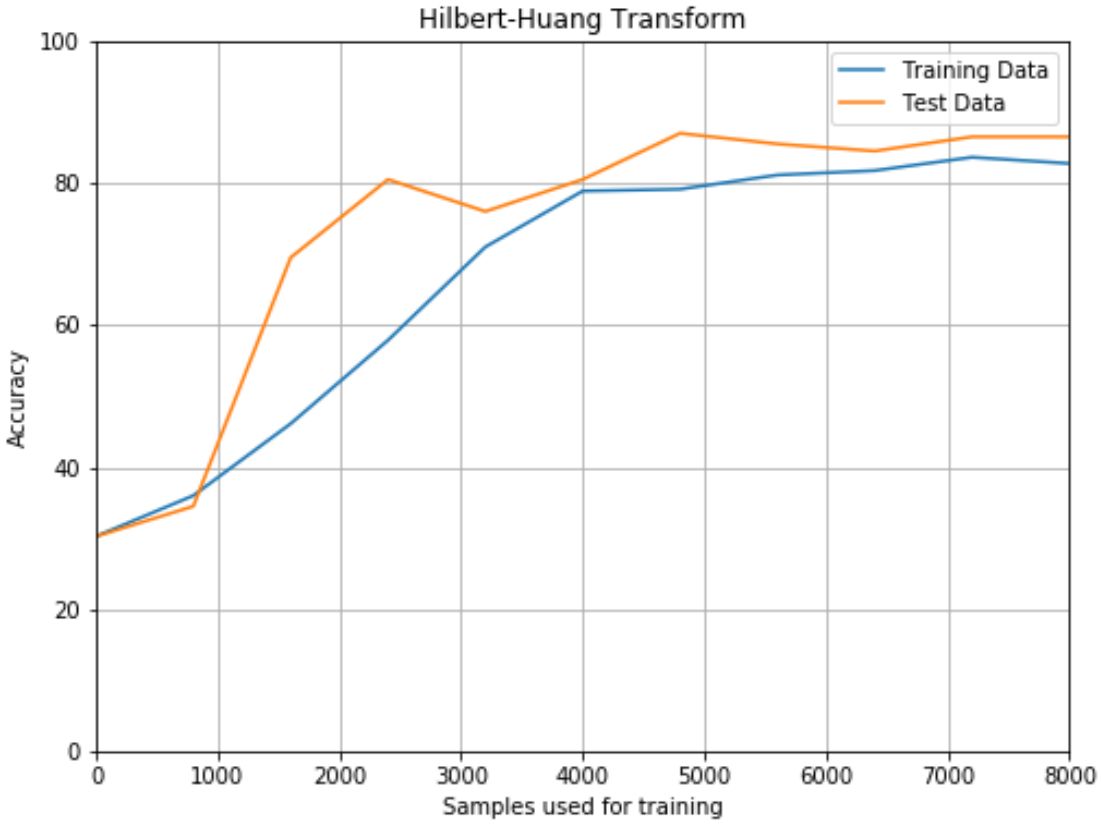

Figure 9.3: Accuracy for synthetic Hilbert spectrum classification

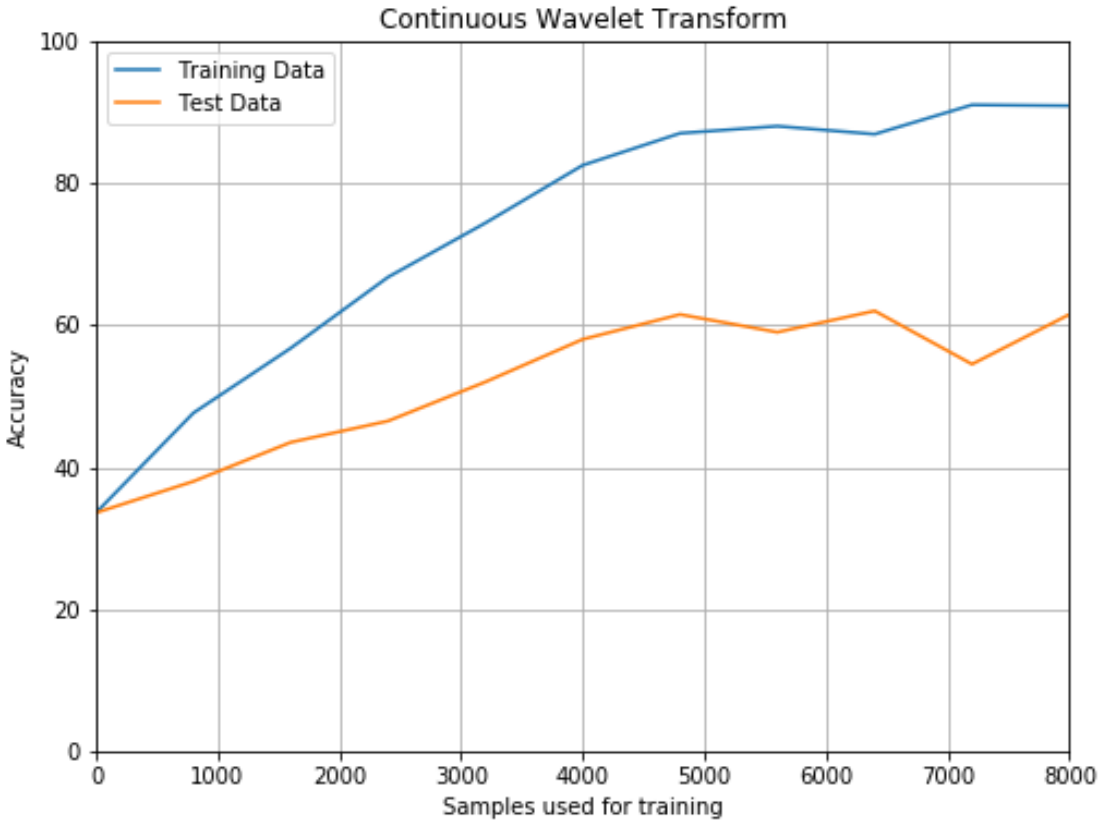

Figure 9.4: Accuracy for synthetic scalogram classification using 5 epochs per batch

Chapter 10

Training on Real Data

For each method, the Convolutional Neural Network (CNN) was trained using 10-fold cross-validation, with 15 epochs. The mean and standard deviation of the accuracy is illustrated in table 10.1.

	STFT	CWT	HHT
Architecture 1	68.00% (+/- 4.77%)	50.00% (+/- 0.00%)	53.00% (+/- 4.48%)
Architecture 2	**72.50% (+/- 3.11%)**	50.00% (+/- 0.00%)	51.79% (+/- 4.21%)
Architecture 3	N/A	55.93% (+/- 3.64%)	58.00% (+/- 5.37%)
Architecture 4	N/A	52.38% (+/- 3.87%)	56.43% (+/- 4.48%)

Table 10.1: Comparison of classification accuracy for each architecture and representation

The results are comparable to those of the synthetic data classification: classification with spectrograms outperforms classification with Hilbert spectra, which in turn outperform classification with scalograms.

Much of the discussion on the results in chapter 9 is also applicable to these results: as discussed in chapter 8, the STFT seems to transform the data better than the CWT or the HHT, both of which produce unwanted distortions or fail to properly represent the data at certain frequencies.

As with the synthethic data, the simplicity and compactness of the spectrograms in comparison to the scalograms and Hilbert spectra is a likely factor in their superiority for this task. The reason why a simpler representation is advantageous might be different, however. The problem of identifying motor imagery in EEG data is without a doubt more challenging than identifying a

pulse in one out of three otherwise stationary modes. For this reason, more might expect the more detailed scalograms or Hilbert spectra to outperform the simple spectrogram representations.

But the simpler spectrograms give rise to simpler machine learning models. Table 10.2 shows a comparison of trainable parameters for each combination of time-frequency representation and architecture. The models using spectrogram representations contain significantly fewer trainable parameters.

	STFT	CWT	HHT
Architecture 1	64,066	8,038,978	8,133,186
Architecture 2	51,650	4,039,106	4,086,210
Architecture 3	N/A	359,106	375,490
Architecture 4	N/A	207,426	215,618

Table 10.2: The number of trainable parameters for each network, given input volume sizes produced by each of the three methods examined in this thesis

As explained in chapter 4, machine learning models are subject to the curse of dimensionality: increasing the complexity of the model significantly increases the amount of data required to fit the more complex model to the problem. Given the limited size of the dataset, it might be easier to fit the smaller number of trainable parameters involved in the smaller and simpler STFT-based models than the much larger number of trainable parameters involved in the larger and more complex CWT- and HHT-based models, even if the more advanced transforms provide more detailed time-frequency representations.

For the scalogram and Hilbert spectrum classification, the deeper architectures outperformed the shallower ones. Two factors that probably influence this is the more complex representations that can be learned using deeper networks and the fact that the increased pooling leads to a significantly smaller number of parameters, see table 10.2.

The effect of using one or two fully-connected layers is inconclusive. For the spectrograms, the network with two fully-connected layers outperformed that with one, whereas with the scalograms and Hilbert spectra, the networks with one fully-connected layer outperformed those with two.

Figure 10.1-10.3 shows a comparison of accuracy for the training data and the validation data

over 30 epochs for each method, using the network architecture on which the method performed best.

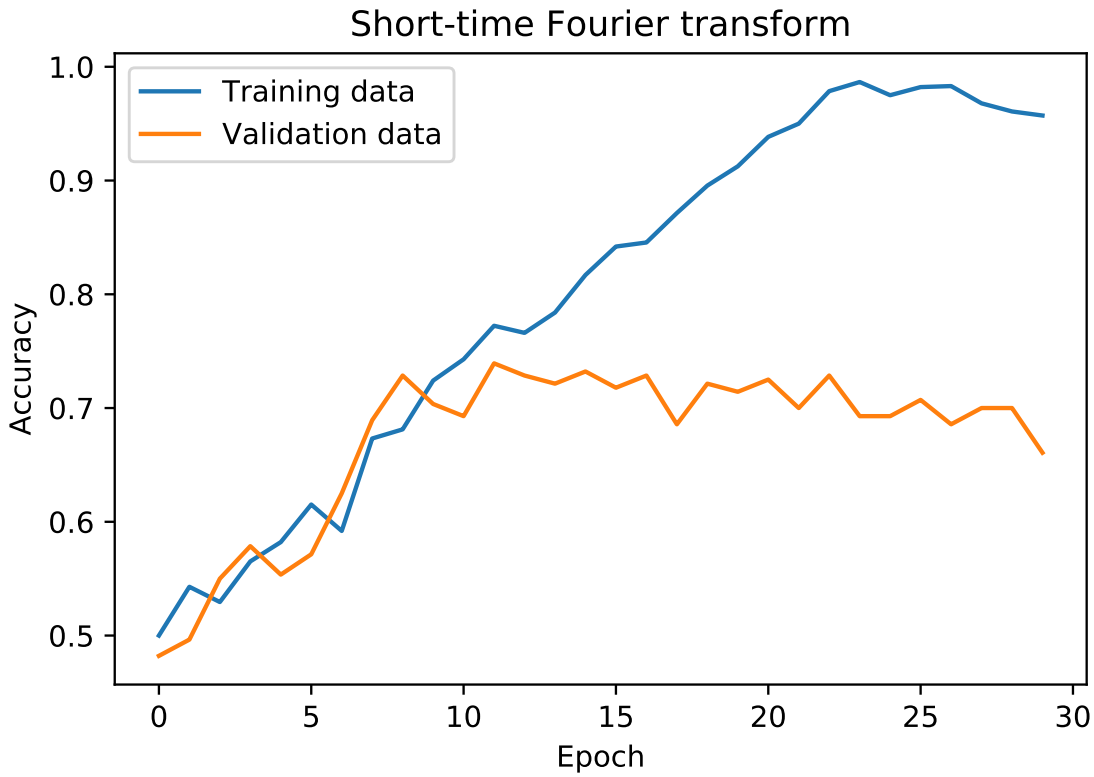

Figure 10.1: Accuracy over 30 epochs with Short-time Fourier Transform (STFT)

In each case, the accuracy for the validation data roughly follows the accuracy for the training data up to between 10 and 15 epochs, at which point the accuracy of the validation data stagnates while the accuracy for the training data continues to improve. This is a typical indication of overfitting. There are two reasons why this may occur.

1. The *quality* of the data is too low. This can happen either because the electroencephalography (EEG) recordings are not good enough or because the transforms produce time-frequency representations which fail to express the necessary information.

2. The *quantity* of data is too low. CNNs are generally applied to very large data sets such as ImageNet, which contains 14 million images[11]. Compared to this, a data set of 1400 images is quite small.

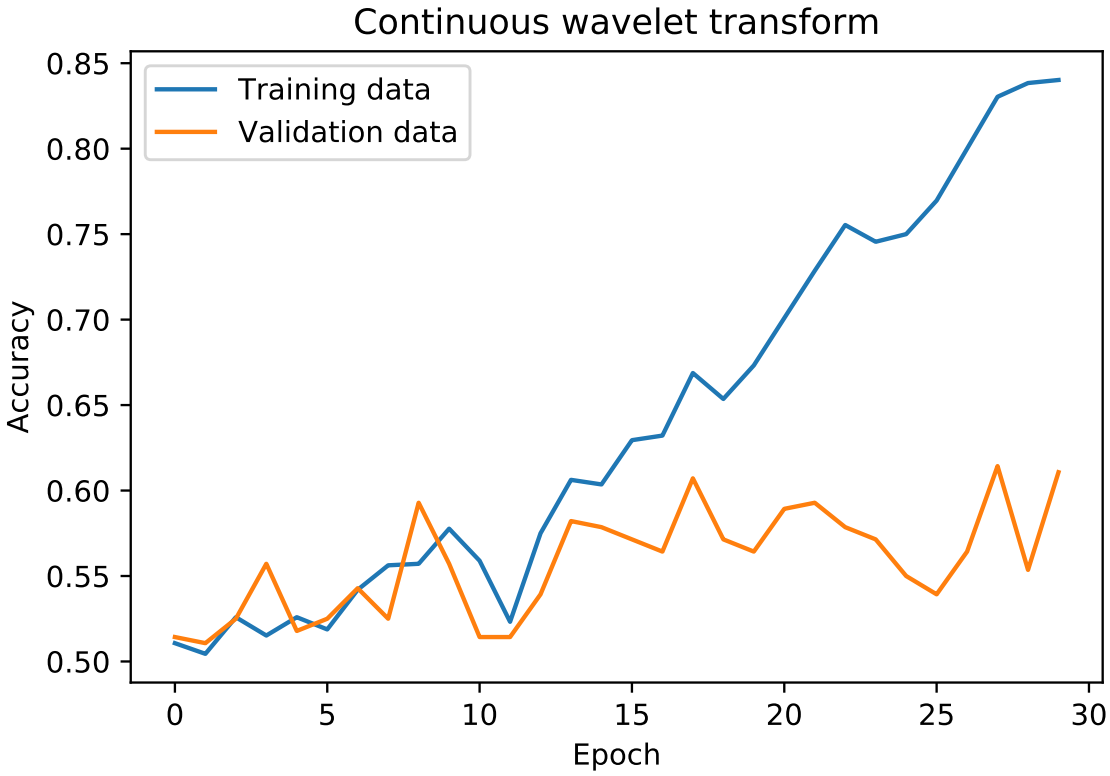

Figure 10.2: Accuracy over 30 epochs with Continuous Wavelet Transform (CWT)

As the EEG recordings were recorded by professionals for the purpose of a competition, they can be expected to be of sufficient quality for classification. Based on the issues discussed in chapter 8, the time-frequency transforms producing invalid representations for classification remains a distinct possibility.

10.1 Using all Channels

Based on the results above, the STFT was selected for further work. The preprocessing was repeated using all sensors, not just the 8 sensors used for the above results. While this leads to unacceptably large input volumes in the case of the CWT and the HHT, the STFT spectrograms are only of size 23x17, so using all the channels is viable. On architecture 1, this led to an accuracy of $66.29\% \pm 11.17\%$ and on architecture 2, this led to an accuracy of $53.14\% \pm 3.60\%$.

Adding the excluded channels to the input did not improve the results. This suggests that the

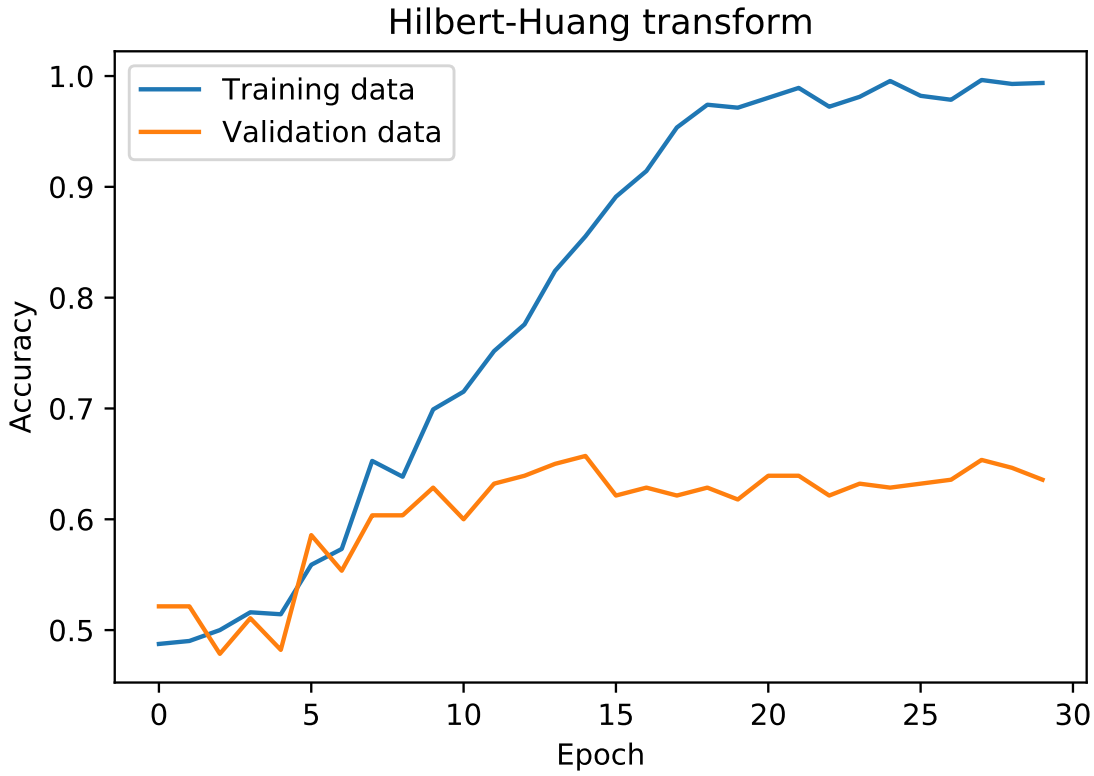

Figure 10.3: Accuracy over 30 epochs with Hilbert-Huang Transform (HHT)

information necessary to discriminate between the two classes was adequately represented by the eight channels used in the original processed data. Another possibility is that the neural network failed to take advantage of the additional information. CNNs train each filter on all channels, and as such the very large number of channels might lead to a lower signal-to-noise ratio, as the training algorithm attempts to fit the filter to all of the channels, each with their own sources of noise, simultaneously.

Part IV

Summary and Conclusions

Chapter 11

Summary

The work done for this thesis involved producing datasets consisting of three different time-frequency representations of both synthetic data and real EEG data: spectrograms produced using the Short-time Fourier Transform (STFT), scalograms produced using the Continuous Wavelet Transform (CWT) and Hilbert spectra produced using the Hilbert-Huang Transform (HHT).

The synthetic data consists of 10,000 signals for which a pulse was "hidden" in one out of three modes oscillating at different frequencies. The classification task was to decide which of the three modes the pulse was to be found in. The EEG data was a publicly available dataset published as part of BCI Competition III. It consists of EEG recordings from five individuals amounting to a total of 1400 recording segments during which the subject was asked to imagine movement in either the right hand or the left foot.

In the preprocessing step, spectrogram were produced using the SciPy implementation of the STFT, scalograms were produced using the SciPy implementation of the CWT and a custom implementation of the HHT was used to produce Hilbert spectra for all the labeled signals in both datasets.

The computational time of the HHT was significantly higher than that of the STFT and the CWT. The scalograms and Hilbert spectra had some issues with representing higher frequencies of the signal, and further analysis revealed mode mixing in the Empirical Mode Decomposition (EMD)

step of the HHT. I was not able to counteract this.

These two-dimensional representations of the signals were then used to train Convolutional Neural Networks (CNNs), of which four different architectures were compared. Only 8 of the in total 118 sensors were used, as the input volumes resulting from using all 118 channels were too large in the case of the scalograms and Hilbert spectra.

The results of testing the network using 10-fold cross-validation can be seen in table 10.1. The highest mean accuracy was achieved using the STFT spectrograms and the network architecture with two fully-connected layer, which achieved a mean classification accuracy of 72.50% and a standard deviation of 3.11%. The method was also tested using input volumes consisting of spectrograms of every channel, but this did not lead to improvements in accuracy.

Chapter 12

Conclusion

Convolutional Neural Networks (CNNs) were able to classify the synthetic data very well when represented as spectrograms and Hilbert spectra, with accuracies of 98.30% and 88.19%, respectively. Representing the data as scalograms led to considerably worse results, though the classifier still managed to outperform random guessing with a classification accuracy of 59.29%. Thus, the answer to **RQ1** is that it is indeed possible to classify signals by training CNNs on their time-frequency representations.

These results where echoed when the CNNs were trained on real EEG data: for all of the time-frequency representations, the classifier managed to outperform what would be expected from random guessing, i.e. 50%. This answers **RQ2** in the affirmative.

Among the different representations, the spectrogram was the clear winner in both the synthetic and real datasets, achiving an accuracy of 72.50% with a standard deviation of 3.11% on the real data. This answers **RQ3**. While it is difficult to conclusively state the reason for this, a likely factor is the fact that the Short-time Fourier Transform (STFT) produces simpler and more compact representations of the data. It was also demonstrated that using spectrograms of a representative sample of the sensors proved just as good as using spectrograms of all of them.

To answer **RQ4**, it's necessary to look at previous attempts to classify the data set. While, as mentioned, the actual competition for which the data was collected differed in that only a smaller subset of the data was available for training, it can still make for an interesting comparison.

The winner of this competition was Yijun Wang from Tsinghua university, who achieved a total classification accuracy of 94.17%[29]. As he was working on only a subset of the training data available for this thesis, this result is clearly superior to that achieved here. The answer to **RQ4** is therefore that, in its present state, the method proposed in this thesis is *not* competitive with established methods.

12.1 Recommendations for Future Work

An area with room for improvement on the work done for this thesis is the implementation of the Hilbert-Huang Transform (HHT). I did not manage to deal with mode mixing in a satisfactory way, and doing so might improve the results. In addition, decreasing the computational time would be necessary for a practical Brain Computer Interface (BCI) implementation based on the HHT.

Beyond the architecture of the network, many of the decisions made in the design of the CNNs used for this work was based on a combination of informal preliminary research and standard recommendations. A more thorough study on tweaking various hyperparameters might lead to a classifier more suited for this kind of problem.

One problem with the results in this thesis is the small size of the data set used. Historically, neural networks only revealed their advantage in regard to other machine learning algorithms when they were applied to very large datasets. As such, it is difficult to conclude whether or not CNNs are applicable in classifying electroencephalography (EEG) data without a comparison with other methods on larger, more complex datasets.

Finally, deep learning has proven to be very useful in the area of *transfer learning*[42]. This is an area of machine learning where a classifier trained on a large data set is further refined for a domain-specific task by training on a smaller data set related to that task. The CNN-based DeepFace facial recognition system, developed by a research group at Facebook[56], was initially trained on a very large data set. Once trained, however, DeepFace can learn to reliably identify a new person using only a few or even one image.

A very useful feature for a BCI application would be the ability to quickly "calibrate" to a user. One method of implementing this could be to train a classifier on a very large dataset to learn how to classify EEG signals as corresponding to certain mental activities, and then further specializing this classifier by training it on a much smaller dataset recorded from the user.

Appendix A

Network Architecture

Figure A.1-A.4 show the four neural network architectures used in the work done for this report.

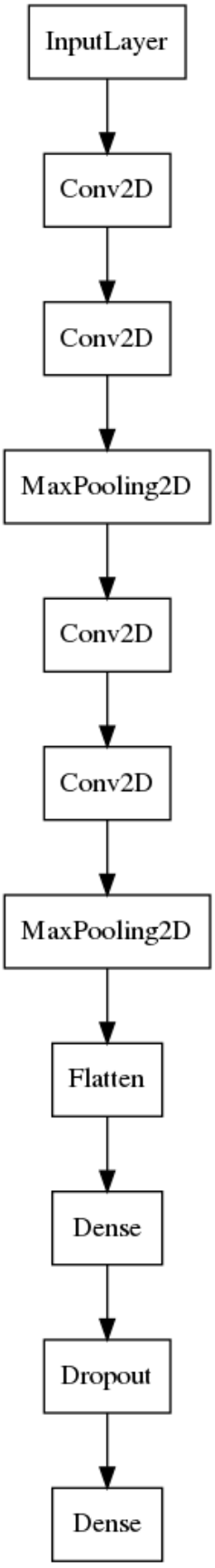

Figure A.1: Architecture 1

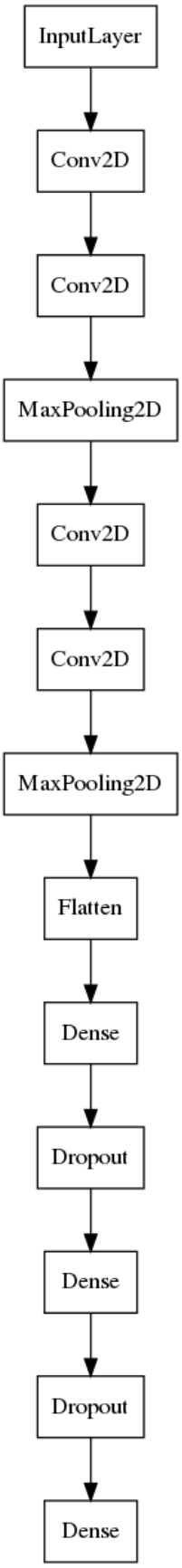

Figure A.2: Architecture 2

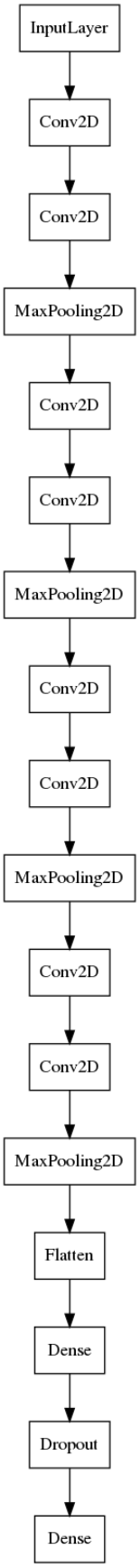

Figure A.3: Architecture 3

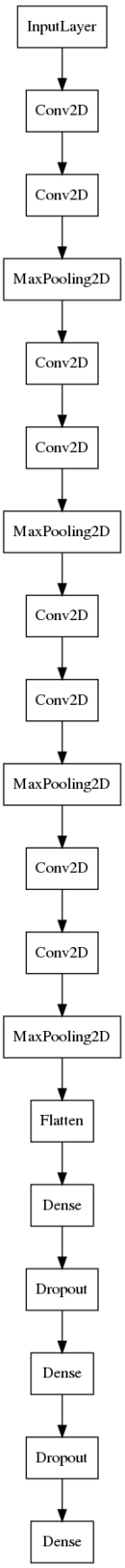

Figure A.4: Architecture 4

Appendix B

Glossary

Glossary

Motor Imagery A mental process in which the subject simulates (i.e. imagines) a particular motor activity. 2

Neuroimaging The science and technology involved in reading brain activity. 2

Non-stationarity A stationary process is a stochastic process for which the probability distribution describing the process remains the same when shifted in time. The frequency spectrum of a signal resulting from the measurement of a nonstationary signal is not, in general, constant in time. 3, 16

Acronyms

ANN Artificial Neural Network. x, 36, 41, 42, 44, 45, 47, 48

BBCI Berlin Brain Computer Interface. 57

BCI Brain Computer Interface. 2, 6–8, 12, 36, 95, 96

CEL Cross Entropy Loss. 67, 68

CNN Convolutional Neural Network. iv, x, 3, 4, 36, 47, 48, 50–52, 57, 65, 86, 88, 90, 93–95

CWT Continuous Wavelet Transform. iv, v, ix, xi, 3, 14, 19–21, 23, 29, 30, 60, 89, 92

ECoG electrocorticography. 7, 11

EEG electroencephalography. xi, 2–4, 6–8, 10, 11, 36, 37, 57, 58, 67, 76, 88, 95

EEMD Ensemble Empirical Mode Decomposition. 29, 76

EMD Empirical Mode Decomposition. x, xi, 24, 26–29, 32, 61, 76, 80, 92

EPSP Excitatory Postsynaptic Potential. 9

FFT Fast Fourier Transform. 59

HHT Hilbert-Huang Transform. iv, v, xi, 3, 14, 24, 28, 30, 57, 61, 64, 75, 89, 90, 92, 93, 95

HSA Hilbert Spectral Analysis. 24, 28, 61, 64

IF Instantaneous Frequency. ix, 21, 22, 24–26

IMF Intrinsic Mode Function. xi, 25–29, 63, 64, 76, 80

IPSP Inhibitory Postsynaptic Potential. 9

MEG magnetoencephalography. 7, 10

MSL Mean Squared Loss. 39, 46, 67, 68

PSD Power Spectral Density. ix, 15–18, 28

PSP Postsynaptic Potential. 9

ReLU Rectified Linear Unit. 66

RQ research question. 3

SGD Stochastic Gradient Descent. 47, 69, 70

STFT Short-time Fourier Transform. iv, v, xi, 3, 14, 17, 18, 29, 30, 59, 88, 89, 92–94

V1 visual area 1. 51

Bibliography

[1] Edgar Douglas Adrian and Brian HC Matthews. “The Berger rhythm: potential changes from the occipital lobes in man”. In: *Brain* 57.4 (1934), pp. 355–385.

[2] Chunxiao Bao et al. “Time-varying system identification using a newly improved HHT algorithm”. In: *Computers & Structures* 87.23-24 (2009), pp. 1611–1623.

[3] Jochen Baumeister et al. “Influence of phosphatidylserine on cognitive performance and cortical activity after induced stress”. In: *Nutritional neuroscience* 11.3 (2008), pp. 103–110.

[4] Hans Berger. “Über das elektrenkephalogramm des menschen”. In: *Archiv für psychiatrie und nervenkrankheiten* 87.1 (1929), pp. 527–570.

[5] Boualem Boashash. “Estimating and interpreting the instantaneous frequency of a signal. II. Algorithms and applications”. In: *Proceedings of the IEEE* 80.4 (1992), pp. 540–568.

[6] Boualem Boashash. “Time-Frequency and Instantaneous Frequency Concepts”. In: *Time-Frequency Signal Analysis and Processing (Second Edition)*. Academic Press, 2016, pp. 31–63.

[7] El H Bouchikhi et al. “A comparative study of time-frequency representations for fault detection in wind turbine”. In: *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*. IEEE. 2011, pp. 3584–3589.

[8] Ming-Shu Chen and Bernard C Jiang. “Resistance training exercise program for intervention to enhance gait function in elderly chronically ill patients: multivariate multiscale entropy for center of pressure signal analysis”. In: *Computational and mathematical methods in medicine* 2014 (2014).

[9] Fernando Lopes Da Silva. “EEG: origin and measurement”. In: *EEg-fMRI*. Springer, 2009, pp. 19–38.

[10] Pieter-Tjerk De Boer et al. “A tutorial on the cross-entropy method”. In: *Annals of operations research* 134.1 (2005), pp. 19–67.

[11] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[12] Jaskiret Dhindsa. “Generalized Methods for User-Centered Brain-Computer Interfacing”. In: (2017).

[13] David L Donoho et al. “High-dimensional data analysis: The curses and blessings of dimensionality”. In: *AMS math challenges lecture* 1.2000 (2000), p. 32.

[14] Jan van Erp, Fabien Lotte, and Michael Tangermann. “Brain-computer interfaces: beyond medical applications”. In: *Computer* 45.4 (2012), pp. 26–34.

[15] Olav B Fosso and Marta Molinas. “Method for Mode Mixing Separation in Empirical Mode Decomposition”. In: *arXiv preprint arXiv:1709.05547* (2017).

[16] Dennis Gabor. “Theory of communication. Part 1: The analysis of information”. In: *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering* 93.26 (1946), pp. 429–441.

[17] Yarin Gal and Zoubin Ghahramani. “Bayesian convolutional neural networks with Bernoulli approximate variational inference”. In: *arXiv preprint arXiv:1506.02158* (2015).

[18] Ian Goodfellow, Y Bengio, and A Courville. “Machine Learning Basics”. In: *Deep Learning* (2016), pp. 94–159.

[19] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT press Cambridge, 2016, pp. 11–28.

[20] Richard HR Hahnloser et al. “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit”. In: *Nature* 405.6789 (2000), p. 947.

[21] Donald Olding Hebb. “Distinctive features of learning in the higher animal”. In: *Brain mechanisms and learning* (1961), pp. 37–46.

[22] Suzana Herculano-Houzel. “The human brain in numbers: a linearly scaled-up primate brain”. In: *Frontiers in human neuroscience* 3 (2009), p. 31.

[23] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[24] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257.

[25] Norden E Huang, Zheng Shen, and Steven R Long. “A new view of nonlinear water waves: the Hilbert spectrum”. In: *Annual review of fluid mechanics* 31.1 (1999), pp. 417–457.

[26] Norden E Huang et al. “A confidence limit for the empirical mode decomposition and Hilbert spectral analysis”. In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*. Vol. 459. 2037. The Royal Society. 2003, pp. 2317–2345.

[27] Norden E Huang et al. “The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis”. In: *Proceedings of the Royal Society of London A: mathematical, physical and engineering sciences*. Vol. 454. 1971. The Royal Society. 1998, pp. 903–995.

[28] David H Hubel and Torsten N Wiesel. “Receptive fields of single neurones in the cat’s striate cortex”. In: *The Journal of physiology* 148.3 (1959), pp. 574–591.

[29] Berlin Brain-Computer Interface. *BCI Competition III Final Results*. 2005. URL: http://www.bbci.de/competition/iii/results/ (visited on 05/25/2018).

[30] Aditya Jain, Gandhar Kulkarni, and Vraj Shah. “Natural language processing”. In: *International Journal of Computer Sciences and Engineering* 6.1 (2018).

[31] Cheng Junsheng, Yu Dejie, and Yang Yu. “Research on the intrinsic mode function (IMF) criterion in EMD method”. In: *Mechanical systems and signal processing* 20.4 (2006), pp. 817–824.

[32] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[33] Douglas M Kline and Victor L Berardi. “Revisiting squared-error and cross-entropy functions for training neural network classifiers”. In: *Neural Computing & Applications* 14.4 (2005), pp. 310–318.

[34] Wlodzimierz Klonowski. “Everything you wanted to ask about EEG but were afraid to get the right answer”. In: *Nonlinear Biomedical Physics* 3.1 (2009), p. 2.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “The convolution operator”. In: *Deep Learning*. 2015, pp. 331–334.

[37] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “The neuroscientific basis for convolutional networks”. In: *Deep Learning*. 2015, pp. 364–371.

[38] Yann LeCun et al. “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[39] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[40] Da-Chao Lin et al. “Elimination of end effects in empirical mode decomposition by mirror image coupled with support vector regression”. In: *Mechanical Systems and Signal Processing* 31 (2012), pp. 13–28.

[41] Alfred Lee Loomis, E Newton Harvey, and Garret Hobart. “Electrical potentials of the human brain.” In: *Journal of experimental Psychology* 19.3 (1936), p. 249.

[42] Grégoire Mesnil et al. “Unsupervised and transfer learning challenge: a deep learning approach”. In: *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning workshop-Volume 27*. JMLR. org. 2011, pp. 97–111.

[43] Tom M Mitchell. “Machine learning”. In: *McGraw-Hill International Editions Computer Science Series* (1997), p. 2.

[44] Klaus-Robert Müller, Benjamin Blankertz, and Gabriel Curio. *Data set IVa ‹motor imagery, small training sets›*. 2005. URL: http://www.bbci.de/competition/iii/desc_IVa.html (visited on 05/10/2018).

[45] John G Proakis. “Frequency analysis of signals”. In: *Digital signal processing: principles algorithms and applications*. Pearson Education International, 2001, pp. 209–280.

[46] Dale Purves et al. “Studying the nervous system”. In: *Neuroscience*. Sinauer Associates, Inc., 2012.

[47] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Searching for activation functions”. In: (2018).

[48] RT Rato, Manuel Duarte Ortigueira, and AG Batista. “On the HHT, its problems, and some solutions”. In: *Mechanical Systems and Signal Processing* 22.6 (2008), pp. 1374–1394.

[49] Gabriel Rilling and Patrick Flandrin. "One or two frequencies? The empirical mode decomposition answers". In: *IEEE transactions on signal processing* 56.1 (2008), pp. 85–95.

[50] Gabriel Rilling, Patrick Flandrin, Paulo Goncalves, et al. "On empirical mode decomposition and its algorithms". In: *IEEE-EURASIP workshop on nonlinear signal and image processing*. Vol. 3. NSIP-03, Grado (I). 2003, pp. 8–11.

[51] SciPy. *scipy.signal.cwt*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.cwt.html (visited on 05/15/2018).

[52] SciPy. *scipy.signal.stft*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.stft.html (visited on 05/15/2018).

[53] Pei-Wei Shan and Ming Li. "Nonlinear time-varying spectral analysis: HHT and MODWPT". In: *Mathematical Problems in Engineering* 2010 (2010).

[54] David Silver et al. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), p. 354.

[55] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[56] Yaniv Taigman et al. "Deepface: Closing the gap to human-level performance in face verification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1701–1708.

[57] "The nervous system and nervous tissue". In: *Anatomy and physiology*. OpenStax, 2013.

[58] Jacques J Vidal. "Real-time detection of brain events in EEG". In: *Proceedings of the IEEE* 65.5 (1977), pp. 633–641.

[59] Jacques J Vidal. "Toward direct brain-computer communication". In: *Annual review of Biophysics and Bioengineering* 2.1 (1973), pp. 157–180.

[60] the Free Encyclopedia Wikipedia. *21 electrodes of International 10-20 system for EEG*. URL: https://commons.wikimedia.org/wiki/File:21_electrodes_of_International_10-20_system_for_EEG.svg (visited on 05/28/2018).

[61] the Free Encyclopedia Wikipedia. *21 electrodes of International 10-20 system for EEG*. URL: https://commons.wikimedia.org/wiki/File:International_10-20_system_for_EEG-MCN.svg (visited on 05/28/2018).

[62] the Free Encyclopedia Wikipedia. *VFPt dipole electric*. URL: https://commons.wikimedia.org/wiki/File:VFPt_dipole_electric.svg (visited on 05/28/2018).

[63] Stephen Wright and Jorge Nocedal. "Line Search Methods". In: *Numerical Optimization*. New York, NY: Springer, 2006. Chap. 3, pp. 30–65.

[64] Zhaohua Wu and Norden E Huang. "Ensemble empirical mode decomposition: a noise-assisted data analysis method". In: *Advances in adaptive data analysis* 1.01 (2009), pp. 1–41.