

UNIVERSITY COLLEGE CORK

SCHOOL OF ENGINEERING

ME IN ELECTRICAL AND ELECTRONIC ENGINEERING

MODULE EE6050 - ME PROJECT

DISSERTATION

Deep Learning for Neonatal Brain Injury Grading



Emmet Horgan (118449344)

Dr. Gordan Lightbody

Declaration

This report was written entirely by the author, except where stated otherwise. The source of any material not created by the author has been clearly referenced. The work described in this report was conducted by the author, except where stated otherwise.

Friday 14th April, 2023

A handwritten signature in black ink, appearing to be 'Emet' or similar, written in a cursive style.

Acknowledgements

There are many people without whom this project would not be at the stage which it is at now.

Firstly I would would to thank my supervisor, Dr. Gordan Lightbody, who gave me the opportunity to take on this project. He provided me encouraging guidance throughout the course of the project, and without whom I would not have been able to achieve the results which I did.

I would also like to thank Dr. John O'Toole of the INFANT centre. Dr. O'Toole gave me access to a particularly important memory efficient algorithm which has proved to be extremely useful and has drastically increased the speed with which I was able to get results which showed I was on the correct path.

I am grateful to Eimear Shortiss, who worked with the same dataset last year and provided me with code and advice which allowed to me to get started more quickly.

Finally I would like to thank my family, and friends for their continued support throughout my undertaking of this project.

Nomenclature

$\mathcal{F}\{\cdot\}$	Fourier Transform of
$\mathcal{H}\{\cdot\}$	Hilbert Transform of
NaN	Not A Number
aEEG	Amplitude Integrated Electro-cardiography
ANSeR	Automatic Neonatal Seizure Recognition
AUC	Area Under Curve
cEEG	Conventional Electro-cardiography
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
ECG	Electro-cardiography
EEG	Electro-encephalogram
FCL	Fully Connected Layer
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FM	Frequency Modulated
FPR	False Positive Rate
GPU	Graphical Procesing Unit
HIE	Hypoxic-ischemic encephalopathy
HRV	Heart Rate Variability
IAF	Instantaneous Autocorrelation Function
MAD	Median Absolute Deviation
MRI	Magnetic Resonance Imagery

NICU	Neonatal Intensive Care Unit
RGB	Red Green Blue
ROC	Receiver Operating Curve
STFT	Short-Time Fourier Transform
SVM	Support Vector Machine
TFA	Time-Frequency Analysis
TFD	Time-Frequency Distribution
TPR	True Positive Rate
WVD	Wigner-Ville Distribution

Summary

Hypoxic-ischemic encephalopathy (HIE) in neonates is a brain injury caused by the lack of oxygen or lack of blood flow to the brain. HIE is the most common brain injury among full-term neonates and occurs at a rate of approximately 2 per 1000 deliveries. The effects of HIE can be very serious with between 40 – 60% either dying or suffering severe disabilities by the age of two years, such as cerebral palsy or learning disabilities. The current best method of classifying the severity of HIE grade is through the use electro-encephalographic signals (EEG) which is a measure of brain activity. There are a host of problems associated with this signal those being the cost of tools to measure it and the clinician’s inability to interpret the data.

This work introduces a new approach to classifying whether or not a neonate needs to be treated for HIE based on the HRV signal. Several other works have been completed using this signal in combination with classical machine learning methods but this work presents a novel approach by producing a time-frequency distribution of a five-minute segment of HRV data and using a convolutional neural network to perform the feature extraction and classification of the time-frequency image. The approach taken in this project and in other works using the HRV signal is to classify whether or not the neonate needs treatment for HIE rather than the individual HIE severity grades. This transforms the problem from a multi-classification problem to a binary classification problem with a more natural decision boundary.

ResNet18 which is a convolutional-neural-network architecture trained on the ImageNet dataset was used as the architecture and pre-trained weights were used such that the network would not be starting from the ground up during training. Before data was passed into the network it was detrended and then rescaled to be in the same numerical range of an RGB image which is the image format ResNet18 was trained on and expects.

This classifier was able to achieve the best performance of all other classifiers trained on the same dataset but only by a marginal amount. This proves three things, the other machine learning approaches were not missing any features, this approach is valid for use with the HRV signal for classifying HIE severity, and there is most likely a limitation with the dataset currently in use which preventing performance improvements. The performance in terms of AUC was determined to 0.782 with a standard deviation of 0.106.

Contents

Declaration	I
Acknowledgements	II
Summary	V
1 Introduction	1
2 Preprocessing	5
2.1 NaN processing	5
2.2 Hampel Filtering	7
2.3 Resampling	13
3 Time Frequency Analysis	15
3.1 Background	15
3.2 Short-Time Fourier Transform	18
3.3 Wigner-Ville Distribution	19
3.4 Application	21
4 Deep Learning	23
4.1 Classical Machine Learning Simulation	23
4.2 Fully Connected Layers	24
4.3 Convolutional Neural Networks	26
4.4 Transfer Learning	28
4.5 Architectural Tradeoffs	28
4.6 Implementation	29
4.7 Training & Testing	30
5 Results	33
5.1 Architecture Optimisation	33
5.2 Transforms Optimisation	35
5.3 Comparison	37
6 Software Implementation	38
6.1 Tools	38
6.2 Structure	39

Future Work	41
Conclusion	42
A Data Approvals	47
B Logbook	48

List of Figures

Figure 1.1:	The top of the figure indicates the HIE grade of the subject in question according to Table 1.1, the grades are increasing in severity from left to right. The bottom plots show the HRV signal which decreases in variability from left to right [14].	3
Figure 2.1:	HRV signal with a highlighted section (red box) containing NaNs .	6
Figure 2.2:	Example sine wave with a step change to show cubic-spline tracking error	7
Figure 2.3:	Median filter outputs given a noise injected sinusoid with a window length of seven meaning that seven data points are seen when calculating the median.	8
Figure 2.4:	Hampel filter outputs given a noise injected sinusoid.	10
Figure 2.5:	TFD of pure sine: The top plot shows the time-frequency distribution of the signal in the bottom plot which is the time-domain input signal. In this case the input is a pure sinusoid, thus in the time-frequency domain it is a constant frequency through time with some windowing effects at the edges.	10
Figure 2.6:	TFDs showing the effect of Hampel filtering on an outlier-corrupted sinusoid.	11
Figure 2.7:	HRV signal plot showing the original outlier corrupted signal (blue) overlayed with the Hampel filtered output (orange)	12
Figure 2.8:	TFDs of a HRV signal before and after Hampel filtering	13
Figure 2.9:	HRV signals interpolated using the cubic-spline algorithm and resampled at 5 Hz.	14
Figure 3.1:	Collection of plots related to a linear frequency modulated signal. .	16
Figure 3.2:	Spectrogram of the linear FM signal plotted in Figure 3.1a with three different window lengths used, 32, 256 and 512 samples respectively.	19
Figure 3.3:	Step by step calculation of the Wigner distribution for a linear FM signal. $f_{initial} = 0.4$ Hz, $f_{final} = 0.1$ Hz [18].	20
Figure 3.4:	HRV signal and corresponding TFDs of a grade one and grade four subject.	21
Figure 4.1:	Depiction of an artificial neuron which takes, N , inputs	23

Figure 4.2:	Example of a fully connected neural network in which the circles represent neurons and the lines between them represent the connections between neurons	25
Figure 4.3:	Computational graph abstraction from a weight through one full hidden layer a single output neuron/layer and the loss calculation	25
Figure 4.4:	27
Figure 4.5:	ResNet18 architecture which shows the convolutional filters and residual connections between layers to overcome the vanishing gradient problem.	29
Figure 4.6:	Depiction of cross-validation moving from the top row down in which the coloured fold is the fold chosen as the test set and each row produces a separate model	31
Figure 4.7:	Illustration of the ideal early stopping algorithm. In the realistic case only one 10 fold cross-validation is performed in which the number of training epochs is held constant at 50.	32
Figure 5.1:	Example ROC curve depicting the "perfect" curve in which all positives are correctly identified and all negatives are correctly identified along with the "random" curve in which the model appears to be choosing its output randomly as the TPR and FPR are the same and increase linearly with threshold	34
Figure 5.2:	ROC plots for various CNN architectures and their respective average AUCs.	34
Figure 5.3:	ROC plots for various transforms applied to the dataset before training on the ResNet18 architecture and their respective AUCs. Note that the pipe operator 'l' is used to imply the order in which transforms were applied e.g. 'Norm l AbsLog' implies normalisation followed by the absolute logarithm transform.	35
Figure 5.4:	ROC plots for all of the models generated during the 10 fold cross-validation (low opacity) for the lowest and highest variance configurations along with their average ROC plot(high opacity)	37
Figure 6.1:	Software structure flowchart	40

List of Tables

Table 1.1:	HIE Severity Grading based on the EEG classification criteria outlined in [7]	1
Table 4.1:	Model and the approximate number of parameters according to measurements calculated using torchvision’s model library [23]. The maximum batch size was determined empirically through training with the models and is a direct function of the GPUs available memory (powers of two used to algorithm efficiency)	28
Table 5.1:	Summary of training results.	36
Table 5.2:	Comparison of the best performing models which have been trained and tested on the same dataset	37

Chapter 1

Introduction

Globally almost 40 % of all child deaths under the age of five years occurs in the neonatal period, with hypoxic-ischaemic-encephalopathy (HIE) being the most common brain injury among neonates occurring at a rate of approximately 2 per 1000 deliveries [1], [2]. HIE is a brain injury (encephalopathy) due to a lack of oxygen (hypoxia) or impaired blood flow in the brain (ischaemia) and its effect can be very serious with between 40– and 60 % either dying or suffering severe disabilities by the age of two years [2]. The long-term outcome of HIE depends on the severity of the initial HIE injury, although some possible outcomes are life-altering conditions such as intellectual disability and cerebral palsy [3], [4].

Currently the main treatment method of HIE is therapeutic hypothermia which involves cooling the infant down to a body temperature in the range of 33–34 °C for a 72 hour period without interruption [3]. However for this treatment method to be effective, it must be commenced within 6 hours of birth which is a very narrow window of time to identify cases of HIE which require treatment. The current methodology for grading HIE severity in a neonate involves using clinical markers which have been proven to be unreliable [5], and have also shown to be poor at differentiating between HIE grades [6]. A better approach to grading HIE is through the use of electroencephalography (EEG) which is widely considered the gold standard for HIE grading as it is highly predictive of long-term outcome [7]. A standard exists for grading HIE based on features of the EEG signals and is outlined in Table 1.1. This predictive ability

Table 1.1: HIE Severity Grading based on the EEG classification criteria outlined in [7]

Grade	Description
0	Normal: Continuous background pattern with normal physiologic features.
1	Mild abnormalities: Continuous background pattern with mild asymmetric patterns or mild voltage depression.
2	Moderate abnormalities: Discontinuous activity with clear asymmetry or asynchrony.
3	Major abnormalities: Discontinuous activity with severe fading background patterns.
4	Inactive: Background activity of $\leq 10 \mu\text{V}$ or severe discontinuity.

is partly due to its capability to continuously monitor the brain which gives a view of how the symptoms change with time.

Electroencephalography (EEG) is a technique in which electrodes are attached to the scalp of a subject at specific areas and the electric field generated by the electrochemical process used by neurons to communicate with one another is detected by the electrodes [3]. This electric field is quite weak on the scalp, and the voltage generated on the electrodes is in the order of microvolts and thus must be amplified before the signal is usable. Different parts of the brain generate different types of activity, so several electrodes are applied to the scalp in a specific manner and each electrode is measured with respect to a reference to get a full picture of the electrical activity throughout the brain [3]. Thus, a normal EEG will show several channels or signals all of which detail the electrical activity in a certain region of the brain. There are many different approaches which can be taken to placing the electrodes but usually the electrodes are placed over the central region of the brain and also over the frontal, temporal, parietal and occipital lobes, specifically for neonates a modified form of the 10-20 electrode placement is used [3]. A montage refers to different approaches to generating the reference used when measuring the voltage, bipolar montages are most common in neonatal monitoring in which a chain is formed where one electrode which is used for measuring electrical activity will also be the reference to another electrode and vice versa.

The current state of the art HIE classification system is based on a convolutional-neural-network (CNN) which takes in an image produced via a time-frequency transformation of the EEG signal [5]. The system, called 'TFDfeat', significantly outperforms its support vector machine (SVM) based predecessor, called 'GSVfeat', on a validation dataset collected from 4 different European countries as part of the Algorithm for Neonatal Seizure Recognition (ANSeR) study [8]. 'TFDfeat' achieves an accuracy of 69.5 % in comparison to the next best 56.8 % achieved by 'GSVfeat' on the validation dataset. A part of this improved performance can be attributed to the fact that for a classical machine learning approach, features must be carefully chosen and manually extracted whereas in the case of a CNN, the required features are learned by the network during the training phase.

Although EEG is the gold standard when it comes to HIE severity grading, there are several problems associated with its practical use. Using EEG comes with a penalty in terms of cost, the EEG machines, equipment, and electrodes need to be purchased, there need to be EEG technicians available and also neurophysiologists to be able to interpret the EEG signals [9], [10]. A common occurrence is the lack of neurophysiologists to interpret the EEG signals, a solution used in practice is for non-experts to use the amplitude-integrated EEG (aEEG) signal instead [10]. 'aEEG' is a bedside neurophysiology tool that uses a limited number of channels to record raw EEG and is passed through a post-processing system to produce a semi-logarithmic amplitude plot on a time-compressed scaled [10]. The primary reason for its use is that it is much easier for a non-expert such as a neonatologist to interpret and apply the system. Although it has been shown that if 'aEEG' was used exclusively to classify the severity of HIE grade, 17 % of those requiring treatment would be missed and a further 43 % of those with mild or normal HIE would be incorrectly classified as moderate or severe, leading them to be cooled inappropriately [11]. It has also been shown in an analysis of 210 surveys from Europe and the

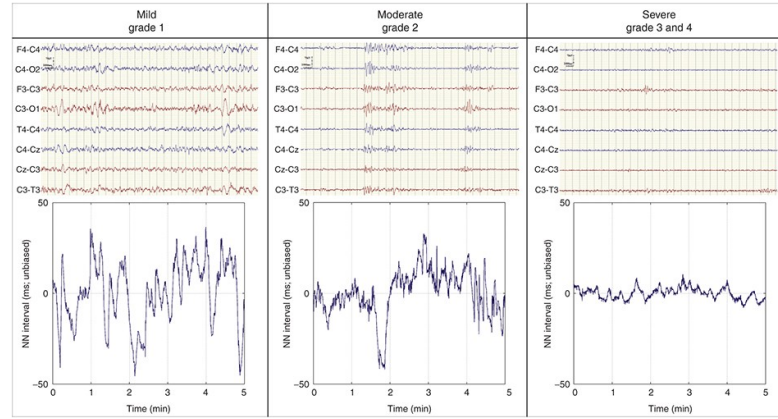


Figure 1.1: The top of the figure indicates the HIE grade of the subject in question according to Table 1.1, the grades are increasing in severity from left to right. The bottom plots show the HRV signal which decreases in variability from left to right [14].

US of neurophysiologists and neonatologists, that only 9 % of respondents said they felt ‘very confident’ in their ability to interpret both ‘aEEG’ and EEG with a further 31 % reporting that they were ‘not confident’ [12]. Half of all those surveyed had received no formal training in EEG interpretation.

One method of solving the issues related to using EEG for classification would be to find another signal which can be directly correlated to HIE grade and is also cheaper and easier to measure and interpret. A signal which meets all of these criteria is the heart rate variability signal (HRV). This signal can be directly derived from the heart rate signal by finding the time between heartbeats using one of many algorithms to identify the time a heartbeat occurs, such as the Pan-Tompkins algorithm used in this study [13]. It has been shown that a strong negative correlation exists between the severity of HIE grade and the HRV signal [4], which has been theorised to be due HIE causing a brain stem injury which controls heart rate rhythm leading to reduced variability in heart rate with increasing severity of HIE grade. Figure 1.1 illustrates this correlation which was taken from [14].

The heart rate signal’s uses extend far beyond grading HIE severity and is used generally within clinical settings to aide with diagnosis. Thus, the heart rate signal is a very common signal and the infrastructure already exists in most medical facilities to measure it, most commonly through electrocardiography. This comparatively low-cost ability to collect heart rate information would also mean that the ability to estimate HIE severity would be extended far beyond just the facilities which can afford the high costs of purchasing and maintaining EEG machines.

Currently, there are several systems that have been developed for grading HIE severity using the HRV signal. The most successful system was heavily influenced by ‘GSVfeat’ using a very similar approach but using only binary classification by choosing not to grade severity according to the arbitrary grading system outlined in Table 1.1 but instead deciding to grade on whether the subject needs to be treated or not, which an argument could be made for this being a more fundamental decision boundary [14]. Another system utilized a custom designed CNN and several HRV features to classify the signal again using this binary classification approach [15]. Finally, the most recent classifier attempted to use an SVM with several features and the Hilbert-Huang transform [16]. Unfortunately, a common validation dataset based on the HRV signal

does not exist at the time of writing but the first system did achieve an accuracy of 81 % on a test dataset. This result is not directly comparable with the EEG based systems outlined earlier due to the binary classification approach taken with the HRV system and was also trained and tested on a different dataset to the two other HRV based classifiers mentioned. The two other HRV based classifiers were able to achieve an AUC of 0.772 and 0.706 respectively.

The approach taken in this paper attempts to utilize a similar approach taken to develop ‘TFDfeat’ and apply this to the development of a classifier using the HRV signal. Thus, the system will generate a time-frequency image of the HRV signal and pass this signal through a convolutional-neural-network to classify the HIE severity grading. Taking note of the previous classification work with the HRV signal [14], a binary classification approach will be used by marking all grade zero and grade one cases as ‘no treatment’ and the rest as ‘treatment’. Thus, in summary, the approach intends to utilize the methodology taken to create the current state-of-the-art and apply it to another signal.

Chapter 2

Preprocessing

To discuss preprocessing, first and naturally the state of the original data and what the data actually represents must be discussed. The data received was essentially 120 sets of two time-series. The 120 sets represents the 120 subjects from which the data was collected. The first time-series, t_1 represents the time at which a QRS peak from an ECG signal was detected using the Pan-Tompkins algorithm [13]. This means that there is no access to the original ECG signal but instead only access to the Pan-Tompkins algorithm processed signal. The second time-series, t_2 which is derived from the first, is the actual heart-rate variability (HRV) signal. This is defined as the time between the QRS peaks of the ECG waveform and is calculated from the first time-series as follows,

$$t_2(i) = t_1(i + 1) - t_1(i) \quad (2.1)$$

2.1 NaN processing

The HRV signal, t_2 was analysed by an expert who removed certain sections which were deemed noise or errors. These errors or artifacts could be due to a multitude of reasons such as movement of the subject causing the sensors to fall off, equipment malfunction or could also not be artifacts and simply be outliers. The actual sections of the data were not removed entirely as this would essentially corrupt the time-series as two completely different sections of the signal would be sliced together which could lead to errors at the interface. Instead these sections were replaced with "Not a Number" objects (NaNs) which is a programming construct. This is a suitable way of labelling this section of the signal as erroneous as the data is essentially unusable until the NaNs are dealt with as these values will cause errors when operations are attempted to be performed on the data. An example plot of a section of a HRV signal which contains NaNs is shown in Figure 2.1.

Thus, the initial section of signal processing required was to formulate a method of dealing with these sections of missing data appropriately and then implementing this method in the Python programming language which was being used to process the data. The first thought that comes to mind in terms of dealing with the missing data is to simply remove the NaNs and treat the data as though they were never there. Unfortunately this simple method of processing the NaNs is not appropriate. The reason for this can be best understood when looking at what section of the signal is of interest. According to the literature [4], the correlation be-

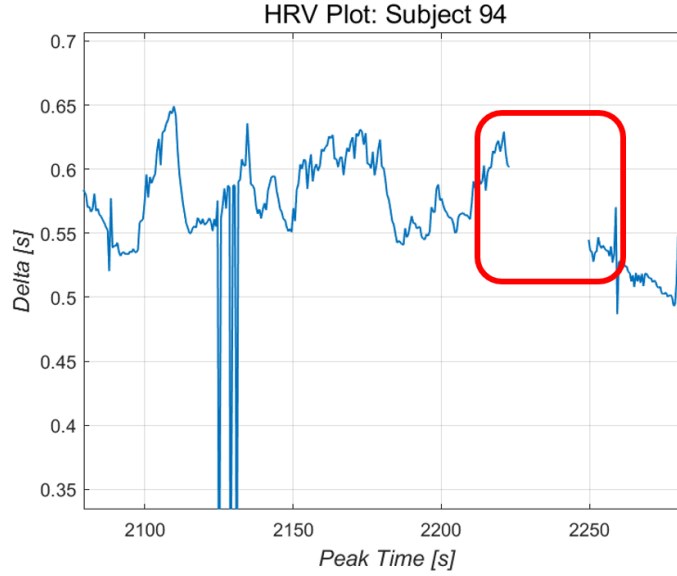


Figure 2.1: HRV signal with a highlighted section (red box) containing NaNs

tween hypoxic-ischemic encephalopathy (HIE) exists in the low-frequency variation of the HRV signal. An example plot that demonstrates this can be seen in Figure 1.1.

If two sections of HRV data which are separated by a series of NaNs are simply spliced together, this could lead to errors in the frequency domain representation if there is a step-like change in the time-domain after the NaN removal. Given that the low-frequency variation of the signal is what is of interest, this method is clearly not appropriate. Another method which is much more suitable if the number of NaNs in a series is very small would be to linearly interpolate between the NaNs which would provide a transition rather than a sudden step in the time-domain. A question might arise here of why linear interpolation would be preferred over perhaps cubic-spline interpolation which would arguably model the data more appropriately. The reason for this stems from the fact that a cubic-spline does exactly what it advertises, that being fitting a cubic polynomial between the data points. This could lead to problems as a polynomial will inherently have a series of curves between the points. This could in cases lead to large rises and falls between the data points which are not representative of the actual signal.

An example which can easily demonstrate this is shown in Figure 2.2 in which two sinusoid signals with a differing DC bias were splined together and then the cubic spline algorithm was applied to the signal and was sampled a much higher rate i.e. resampled the signal. As can be seen in the plot, the cubic spline algorithm is fantastic at tracking the true signal in the slow moving variations but when a sudden high-frequency step occurs, the cubic polynomial fitting can be seen in which there is an under- and over-shoot at the lower- and upper-half of the step.

At this point in the pre-processing stage there are many outliers and high-frequency changes within the data making this technique not advisable, however, cubic spline interpolation is used later in the pre-processing stage in order to resample the signal, after the high-frequency spikes have been removed. Thus, in the case of a small number of NaNs in a series ($\ell < 3$), it has been determined that cubic-spline interpolation is not an appropriate method of overcoming the missing data, instead linear interpolation will be used just to join the two segments of data.

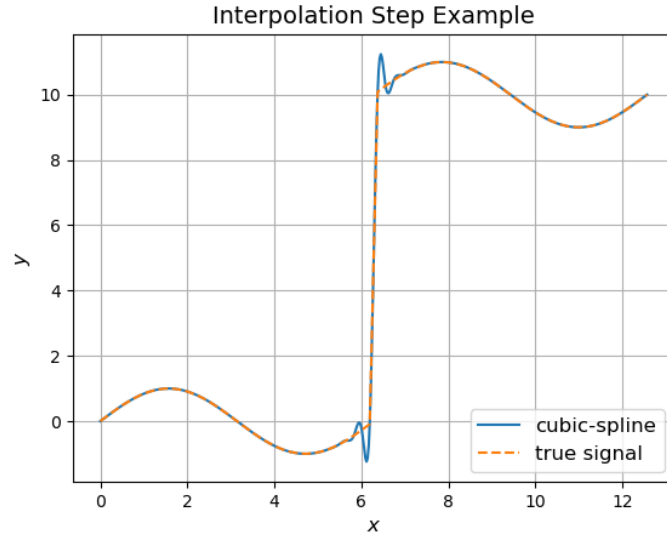


Figure 2.2: Example sine wave with a step change to show cubic-spline tracking error

For longer sections of data, linear interpolation is also inappropriate as it is far too simple of a model of the missing data. In this case, the majority case, it is safest to split the data. Meaning that a singular time-series file will be split at the point where a series of NaNs is encountered, where the length of the series is greater than some arbitrary number, three in this case. One singular time-series could be split into four separate files for example, each file containing an uncorrupted segment of the HRV signal for a particular subject.

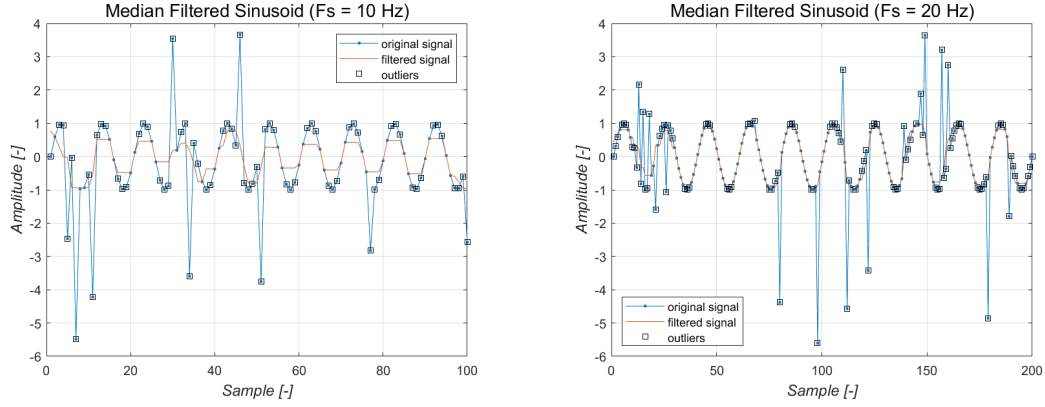
Note also that the timing of the peaks, or the result of the Pan-Tompkins algorithm, is also split and is known as the “**rr_peaks**” signal or t_1 , and is used as the x-axis in plots. This means that in plots of the HRV signal, there is an extremely intimate relationship between the HRV signal and its position along the x-axis, this relationship was outlined in Equation 2.1.

2.2 Hampel Filtering

Hampel filtering can be viewed as a method of outlier removal. In the data it can be seen at various points that there are several high frequency spikes, these spikes are viewed as noise and artifacts considering that the feature of interest in the signal is the slow moving, low frequency trend. These spikes are a major problem as in the time-frequency domain, they will be dominant and cause a lot of high frequency components which would overshadow the components of interest and could lead to misinterpretations by the deep learning algorithms. Thus, it is important to remove these spikes such that a clear time-frequency view of the data can be obtained.

There appear to be three options for the removal of these components, low-pass FIR filtering, outlier removal as these spikes can be viewed as outliers from a statistical point of view, or both. The primary method of removal was chosen to be Hampel filtering as it gave good results and did not alter the frequency-domain representation whereas with FIR filtering, things like phase distortion need to be considered.

Hampel filtering is a generalisation of median filtering [17], it is a non-linear method of



(a) Median filter output given a noise injected sinusoid. $f = 1$ Hz, $f_s = 10$ Hz.

(b) Median filter output given a noise injected sinusoid. $f = 1$ Hz, $f_s = 20$ Hz.

Figure 2.3: Median filter outputs given a noise injected sinusoid with a window length of seven meaning that seven data points are seen when calculating the median.

filtering. First to discuss median filtering. This is a relatively simple method of filtering in which a window,

$$\mathbf{W}_k^K = \{x_{k-K}, \dots, x_k, \dots, x_{k+K}\} \quad (2.2)$$

is passed over the data. The window is of length $2K + 1$ and the length can be changed as a hyperparameter. As this window is passed over the data, the median of the window is calculated and the central element of the window, x_k , is replaced by this median.

$$x_k = \text{median}\{\mathbf{W}_k^K\} = \text{median}\{x_{k-K}, \dots, x_k, \dots, x_{k+K}\} \quad (2.3)$$

This is a somewhat aggressive method of filtering as the central element is replaced no matter what. Another downside to this method of filtering is that there is only one hyperparameter available for tuning which is quite restrictive. An example of median filtering can be observed in Figure 2.3a in which a generic sinusoid, $x(t) = \sin(\theta(t))$, which has some deliberate outliers inserted is passed through a median filter.

The input signal is a sinusoid of the form, $x(t) = \sin(2\pi ft)$, with $f = 1$ Hz and has been sampled with $f_s = 10$ Hz. Ten percent of the sinusoid's data was replaced with outliers in the range of $[-10, +10]$. This can be clearly seen in the plot where the outliers dwarf the input sinusoid, which is representative of the case for the HRV signal. It should be noted that a point is classified as an outlier in the plot if the value which replaced the central window element is different than the original signal ($x_{k_{\text{original}}} \neq x_{k_{\text{filtered}}}$). The filtered signal result is not ideal. In terms of the amplitude it is not correct as it appears to be oscillating between ± 0.5 while the original signal's amplitude was ± 1 . The signal is also obviously distorted especially at the maxima points where the signal appears to be closer to a square wave than to a sinusoid. This square like feature is to be expected when taking into account the operation being performed. When the window encapsulates the maxima points, the maximum will be replaced by the median which explains the shape. This effect can be mitigated if the window length is smaller than the number of data points making up the maxima turning point, meaning that the window will not

see as much of the turn as with a larger window. Considering the window length in this example is only seven data points, it would not be feasible to reduce the window length. Instead if the sampling rate is increased to 20 Hz the effect should be reduced.

Observing Figure 2.3b, the result here is far better than before, all of the outliers have been removed and the signals morphology is relatively similar to the underlying sinusoid, although it is still somewhat distorted and most outlier detections are still occurring at the turns. The trade-off in this case was that the sampling frequency had to be increased, such that the maxima turning point could be captured with far more data points than make up the median window. In this example, that is fine but for the HRV data that is to be used, this is not a trade-off that can be made.

On the positive side, there do not appear to be any more outliers in the signal, although for this benefit, the signal's morphology has been sacrificed. This is not what is wanted as the filter is far too aggressive, although, the Hampel filter which is a modified extension of the median filter, can be used which introduces another hyperparameter other than the window length. This will give the ability to reduce the aggressiveness of the filter and preserve the morphology.

As mentioned, the Hampel filter introduces a new hyperparameter which can be adjusted, that being a threshold. Meaning that the central element of the window, x_k will only be replaced if the magnitude of the difference between the central element and the median of the window surpasses this threshold. This decision boundary is expressed as,

$$|x_k - m_k| \geq tS_k \quad (2.4)$$

Where t is the threshold value to be chosen and S_k is the median-absolute-deviation (MAD) scale estimator, defined as follows,

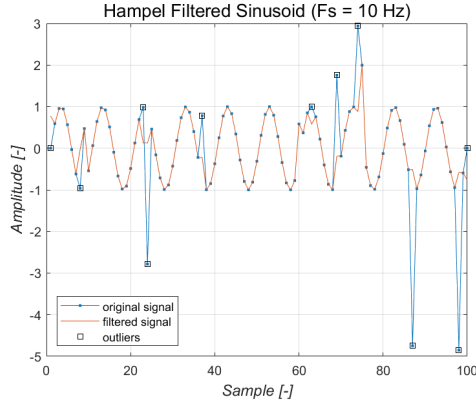
$$S_k = 1.4826 \times \text{median}_{j \in [-K, K]} \{|x_{k-j} - m_k|\} \quad (2.5)$$

This is essentially an estimate of the standard deviation of the window values from the median value i.e. $\hat{\sigma}_{m_k} \approx k \times MAD$, where k is some scaling factor, 1.4826, in this case. Then the output of the Hampel filter is defined as,

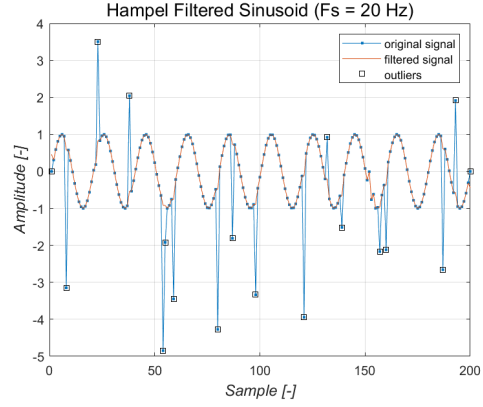
$$y_k = \begin{cases} x_k & |x_k - m_k| \leq tS_k \\ m_k & |x_k - m_k| > tS_k \end{cases} \quad (2.6)$$

This reduces the aggression of the filter as the central element of the window will only be replaced in the case that the absolute value of the difference between the central element and the window median, is greater than some multiple of the standard deviation of window elements about the window median. This method of filtering is much more sophisticated than the simple median filtering method described above.

In Figure 2.4a it is used on the same example as above with a sampling rate, $f_s = 10$ Hz, $|\mathbf{W}_k^K| = 7$ and $t = 1$. It is immediately obvious that this method is much more effective than its median filtering counterpart. The morphology of the signal is almost perfectly preserved and all of the outliers are removed. The outliers are not mapped back to the ideal position perfectly



(a) Hampel filter output given a noise injected sinusoid. $f = 1$ Hz, $f_s = 10$ Hz.



(b) Hampel filter output given a noise injected sinusoid. $f = 1$ Hz, $f_s = 20$ Hz.

Figure 2.4: Hampel filter outputs given a noise injected sinusoid.

but they are much better than the original placing. Looking specifically at which data points are identified as outliers, the power of this filter technique can be seen, every outlier has been marked as an outlier while the vast majority of data points are marked as normal.

Another example showing the same signal sampled at 20 Hz is shown in Figure 2.4b. As can be seen the result is even better again. All of the outliers have been caught and there is not a single misclassification. It is clear that an oversampled signal and Hampel filtering work well together.

Now a look into the time-frequency domain to see the effect of Hampel filtering. A time-frequency plot of the pure uncorrupted sinusoid is shown in Figure 2.5 in order to get an intuitive understanding of what a time-frequency plot is, and how to interpret it.

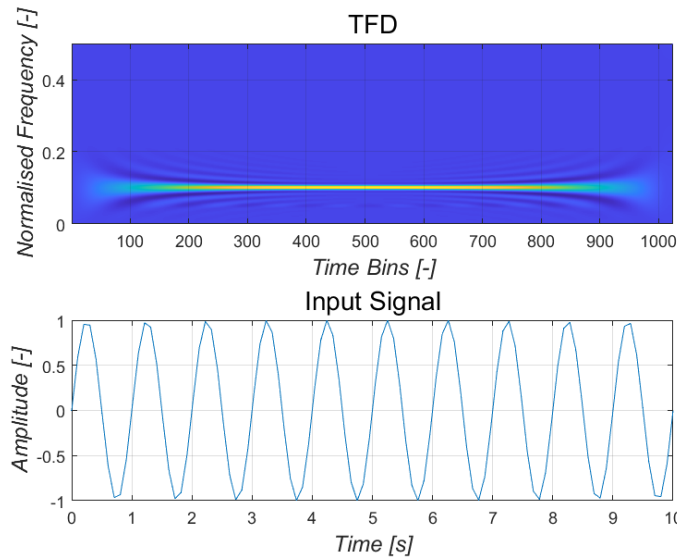
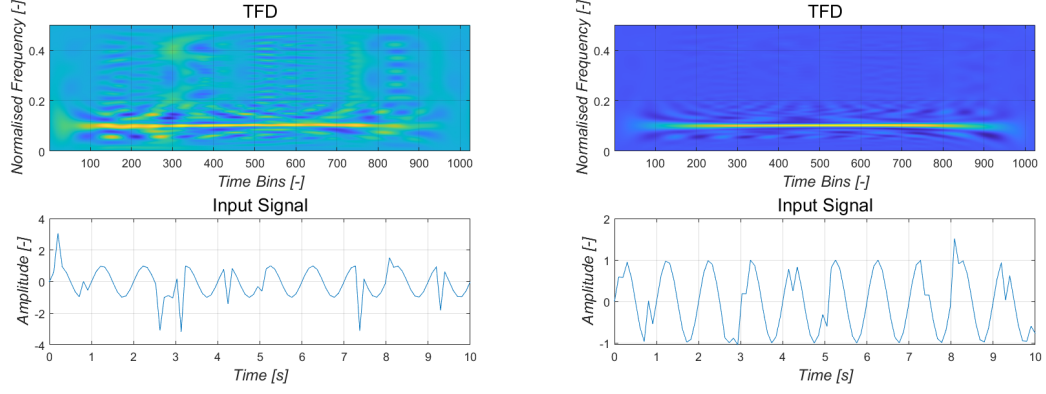


Figure 2.5: TFD of pure sine: The top plot shows the time-frequency distribution of the signal in the bottom plot which is the time-domain input signal. In this case the input is a pure sinusoid, thus in the time-frequency domain it is a constant frequency through time with some windowing effects at the edges.



(a) TFD of outlier-corrupted sine: The time-domain signal is obviously sinusoidal but highly corrupted, the effects on the TFD are obvious broadband noise which shows as vertical lines. (b) TFD of hampel filtered sine: The effects of Hampel filtering are clear on the TFD and the time-domain signal in comparison with Figure 2.6a

Figure 2.6: TFDs showing the effect of Hampel filtering on an outlier-corrupted sinusoid.

The specifics of these plots are discussed in detail in Chapter 3 but in short, the top plot depicts an image with time on the x-axis and frequency on the y-axis. This is the time-frequency distribution (TFD) of the pure sinusoid signal in the bottom plot, which describes how the frequency content of the signal change with time. The brighter the colour in the TFD the larger the signal power whereas a dark blue colour indicates low signal power at that particular frequency and time. In this plot, a horizontal coloured line protrudes across from the $0.1 f_s$ Hz point on the frequency axis. This orange/yellow line is indicative of high power compared to the low-power blue background. This is what would be expected of a pure sinusoid as it shows a mono-frequent signal whose frequency does not change in time. It can also be observed that the power tends to decrease at the beginning and the end of the time axis, this is due to the windowing which must occur when using data of finite duration.

Now that the time-frequency plot has been introduced, a view of the outlier corrupted sinusoid is shown in Figure 2.6a to indicate the problem which these outliers cause in the time-frequency domain. This plot is much more interesting as some other frequency components can be seen throughout the plot. Corroborating with the time-domain signal below the TFD in Figure 2.6a of the plot, it can be seen that at the times where an outlier occurs, the time-frequency plot displays power distributed across a broad range of frequencies i.e. light coloured lines protruding vertically. The larger the outlier the higher the power. It can be seen now, in the case that these outliers occur in the HRV data, the learning algorithms for the convolutional-neural-network could quite easily decipher some correlation between the broadband noise and HIE grade. Thus, it is safer to remove these outliers from the data and move the focus of the learning algorithms toward the low-frequency trends in which a correlation with HIE grade is known to exist.

The effect on the time-frequency plot of passing the data through a Hampel filter is shown in Figure 2.6b As can be seen above the Hampel filter has significantly reduced the amount of broadband power present in the time-frequency plot. Although it should be noted that some of the outliers which were not mapped to an appropriate value do cause some broadband power,

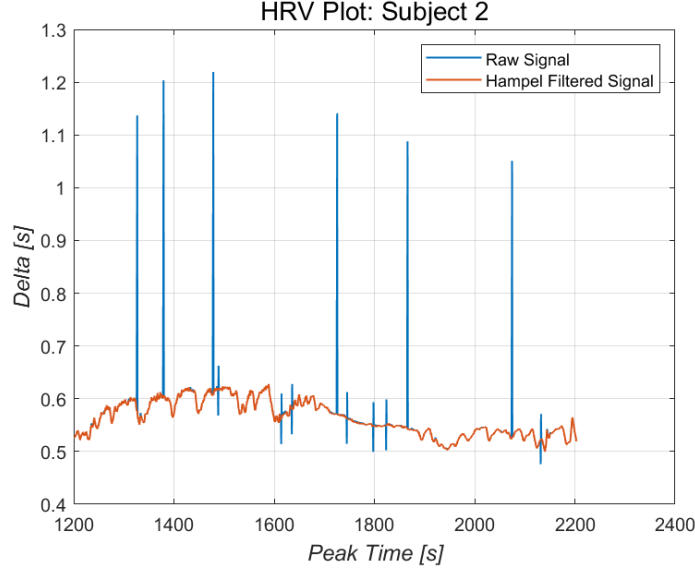


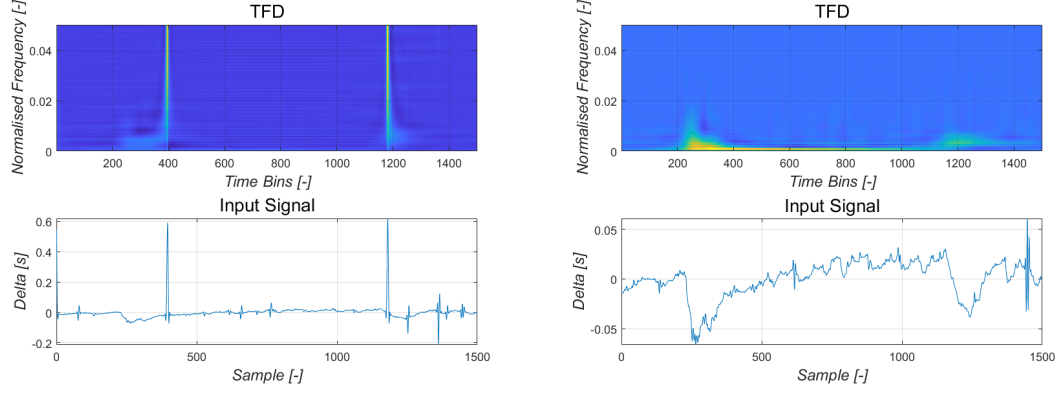
Figure 2.7: HRV signal plot showing the original outlier corrupted signal (blue) overlayed with the Hampel filtered output (orange)

the amount of broadband power present is significantly reduced. The effect of Hampel filtering is also evident in the time-domain plot of Figure 2.6b which is much similar to a sinusoid shape although is not perfect.

The HRV data being used during this research contains high-frequency artefacts. In the time domain, these artefacts appear as high magnitude spikes or outliers. The cause of these artefacts is not clear as it could be the result of a multitude of factors such as a subject moving causing errors in the sensor or misclassifications of QRS peaks by the Pan-Tompkins algorithm. Either way, these high frequency spikes are present in the data which will cause large distortions in the time-frequency plot. The area of interest in the frequency domain is between 0 – 2 Hz [4], as the correlation between HIE grade and HRV signal exists in the slow-moving features of the signal. Thus, these high-frequency spikes will cause large distortions in the time-frequency plot and will overshadow any of the slow moving behaviour which ideally would be captured. An example of these high-frequency spikes in the time-domain HRV data can be seen in Figure 2.7.

Fortunately, the hampel filtering technique discussed above can be used to remove these spikes without losing or distorting the low-frequency component of interest. An example of the TFD of a five-minute segment of HRV data can be seen in Figure 2.8a. It should be noted that the signal has been resampled so that there is a constant amount of time between samples so that time-frequency analysis techniques can be used and the signal was also de-trended to remove the effect of a large dc bias. The time-frequency plot has been restricted to a frequency range of $[0, 0.05f_s]$ so that the areas of interest are clearly visible. In the time-domain signal plot of Figure 2.8a it can be seen that the high-frequency outliers are very large relative to the rest of the signal. This is also evident in the time-frequency plot where large broadband power can be seen. This tends to drown out the rest of the low-frequency power section of the signal which is of interest.

Finally in Figure 2.8b a view of the time-frequency distribution of the same signal section



(a) TFD of a subsection of the outlier corrupted signal plotted in Figure 2.7. The time-domain plot shows clear outliers which are obviously reflected in the TFD at the corresponding vertical locations.

(b) TFD of the Hampel filtered version of the signal in Figure 2.8a. It is clear from the time-domain signal at the bottom that the outliers have been effectively removed and this is reflected also in the TFD in which the low-frequency components are visible.

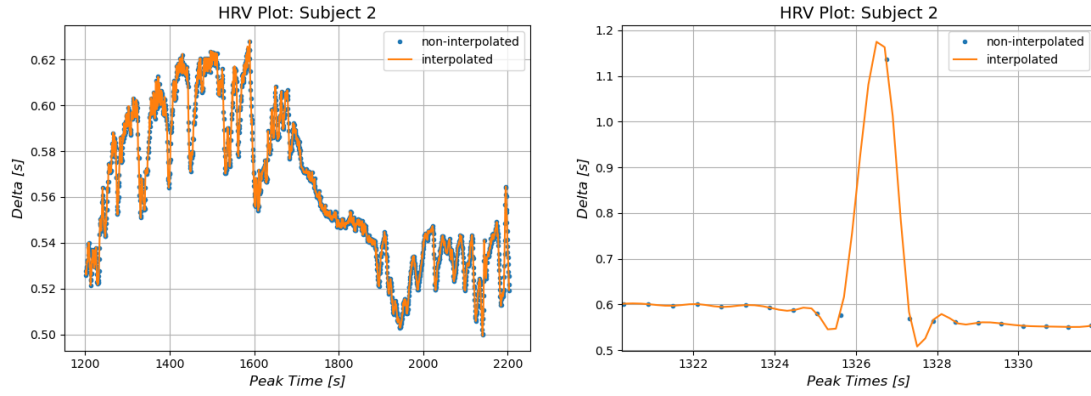
Figure 2.8: TFDs of a HRV signal before and after Hampel filtering

is shown after Hampel filtering. As can be seen, the result is very different. In the time-domain signal, it is obvious that the outliers have been removed and the low-frequency, low-power variations can be seen clearly. In the time-frequency plot the changes in frequency over time can be evidently seen. The signal power at low-frequencies is very clear in comparison to the TFD in Figure 2.8a. This shows the power of the Hampel filtering technique to remove the influence from these high-frequency outliers. It can be understood intuitively that if the unfiltered time-frequency plot shown above was in a training set for training a neural network, then the algorithms could easily find a correlation between grade and high-frequency noise as the noise is obviously amplified in the plot.

2.3 Resampling

Once the Hampel filtering was complete, the data had to be resampled. The reason for this is due to the fact that the t_1 time-series or the ‘rr-peaks’ represents the time at which QRS complex of an ECG signal occurred. This time naturally will not always occur at constant intervals as the heart-rate of the neonate changes. Thus, the signal that results is by nature a non-uniformly sampled signal. To clarify, the ECG signal itself is uniformly sampled but after processing to extract the times at which the QRS complex occurred via the Pan-Tompkins algorithm [13], the resultant signal is not.

It was not possible to move forward with the pre-processing until the data is resampled at a constant sampling rate due to almost all signal processing techniques relying on this, including the time-frequency analysis techniques to be introduced in Chapter 3 and which produced the plots described in the previous Section 2.2. It should also be noted that the ‘rr-peaks’ signal or t_1 is intrinsically related to the ‘rr-interval’ signal or t_2 . The ‘rr-interval’ signals will be resampled which is strange to consider given that the ‘rr-interval’ represents the difference between consecutive ‘rr-peaks’ data points. One might intuitively think that when resampling the signal



(a) Resampled, Hampel-filtered HRV signal where the blue dots indicate the original non-uniform sample points and the orange curve is the interpolated trend curve. This is the same signal as in Figure 2.7.

(b) Resampled signal without Hampel filtering. This is the same signal as shown in Figure 2.9a except without Hampel filtering and a particular outlier in the signal is shown which highlights the over- and under-shooting of the cubic spline algorithm in response to sudden steps.

Figure 2.9: HRV signals interpolated using the cubic-spline algorithm and resampled at 5 Hz.

it does not make sense that the HRV signal could have a changing value between the heart beats as this does not make physiologic sense, but from a signal processing perspective, this signal makes sense and it is best to think of the resampled signal as representing the trend of the HRV signal rather than the actual HRV signal itself.

To resample the signal an interpolation model of the signal must be built which will then be sampled at the desired sampling rate. The sampling rate chosen is that of 5 Hz which was also used in [14]. This sampling frequency encompasses all of the necessary frequency components to grade HIE [4], as the highest frequency component of interest in the signal is 2 Hz. The method of generating an interpolation model chosen was the cubic-spline interpolation algorithm, which has the lowest error among the polynomial interpolation algorithms. An example of the cubic-spline operating on a sinusoid with a stepped DC bias can be seen in Figure 2.2.

Although the cubic spline did not perform well for steps in the signal's DC component as the polynomial which is fit between the points tends to under- and over-shoot if there is a large step change in the signal, it was great at tracking the low-frequency variation of the signal.

At this point in the pre-processing pipeline, the outliers of the HRV signal have been removed via Hampel filtering, meaning that for the majority case, it is safe to use the cubic-spline interpolation algorithm. An example of a section of resampled data overlaid onto the original non-uniformly sampled Hampel-filtered signal can be in Figure 2.9a. It can be seen that the resampled signal appears to track the original signal quite well, with no visibly apparent over- or under-shoots in the plot.

In contrast, looking at Figure 2.9b which shows an equivalently resampled, non-Hampel filtered HRV signal, in fact the same signal as in Figure 2.9a except focused on a different section of the signal, it can be clearly seen that when an outlier is encountered, the cubic-spline does over- and under-shoot the sudden high-frequency change. Looking at the raw HRV signal in Figure 2.7 which contained a huge amount of outliers throughout the signal duration, it can be seen that there would be a large amount of error due to this effect.

Chapter 3

Time Frequency Analysis

3.1 Background

Time frequency analysis is a method of looking at the frequency components of signals as they change over time. It should plot the instantaneous frequency of a signal, which is the frequency of the signal at a specific moment in time, as a function of time. This is different from the Fourier transform, as the Fourier transform looks at the frequency content of a signal over the signal's full duration and essentially averages out the frequency content of the signal. This can be seen in the definition of the Fourier transform below,

$$X(\omega) = \mathcal{F}\{x(t)\} = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t} dt \quad (3.1)$$

where the limits of integration are taken over infinity or equivalently, over the entire duration of the signal where $|x(t)| > 0$. This essentially 'integrates out' the time variable and we see the total frequency content of the signal. A useful signal which displays a weakness of the Fourier transform is the linear frequency modulated signal. This is a simple sinusoidal signal of the form,

$$s(t) = \cos(\theta(t)) \quad (3.2)$$

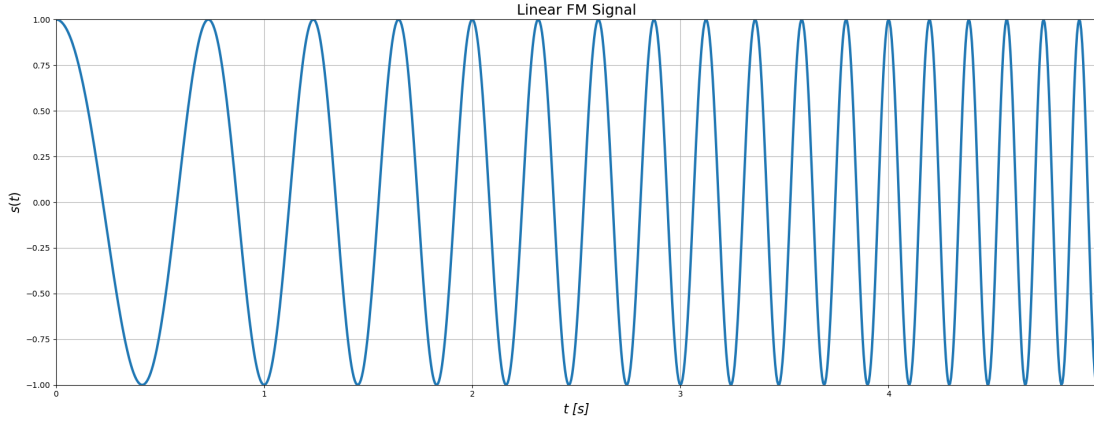
in which the phase, $\theta(t)$, is a function of time like most all other sinusoidal signals. The frequency of the signal, however, is not constant and is a linear function of time,

$$f(t) = \frac{1}{2\pi} \frac{d\theta}{dt} \neq c \quad (3.3)$$

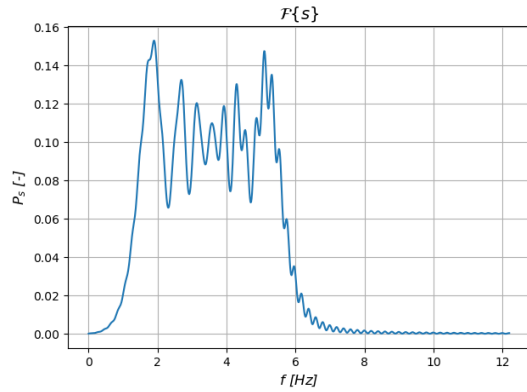
where, c , is some arbitrary constant. Instead, $f(t)$, the instantaneous frequency of the signal should be of the form,

$$f(t) = \frac{1}{2\pi} \frac{d\theta}{dt} = f_0 + \alpha t \quad (3.4)$$

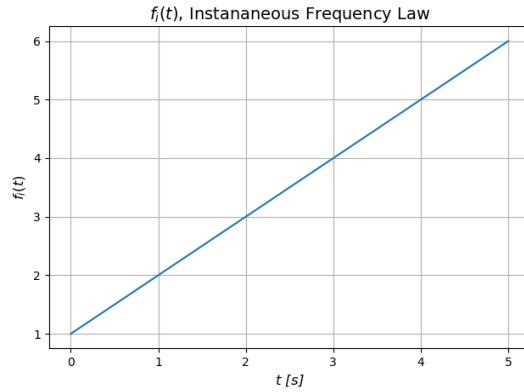
In which, f_0 , is the starting frequency at the beginning of the signal ($t = 0$) and, α , is the rate of change of the frequency of the signal. It is also assumed here that the signal begins at time equal to zero. The phase, $\theta(t)$, which corresponds to this frequency function can be easily obtained by integrating the instantaneous frequency from the beginning of the signal, to some time t . The



(a) Linear FM signal with $\alpha = 1$ and $f_0 = 1$



(b) FFT of the linear FM signal in Figure 3.1a



(c) Instantaneous frequency law of the linear FM signal in Figure 3.1a. The instantaneous frequency law corresponds to Equation 3.4.

Figure 3.1: Collection of plots related to a linear frequency modulated signal.

result of this integration is,

$$\theta(t) = 2\pi \int_0^t f_0 + \alpha t \, dt = 2\pi \left(f_0 t + \frac{\alpha t^2}{2} \right) \quad (3.5)$$

This leaves the following signal,

$$s(t) = \cos(\theta(t)) = \cos \left(2\pi \left(f_0 t + \frac{\alpha t^2}{2} \right) \right) \quad (3.6)$$

A plot of a signal which has this form is shown in Figure 3.1a in which the signal's frequency monotonically increases with time. The reason this is brought up in the time-frequency discussion is that when the Fourier transform of the signal is observed, an interesting result appears and can be seen in Figure 3.1b.

As can be seen, the result is not intuitive. From looking at the time-domain plot of the signal in Figure 3.1a, it is not at all obvious that the signal would appear as it does in the frequency domain. Looking at the plot of what is termed the "instantaneous frequency law" in the literature [18], or the function which determines the instantaneous frequency, in Figure 3.1c the correlation is still not clear.

The reason for this is as hinted at before. The Fourier transform of a signal gives another signal, which describes the distribution of the signal in the frequency domain, independent of time. It does not account for time and thus, there is no time resolution. The idea behind time-frequency analysis is to produce a two-dimensional signal or image, which provides both time resolution and frequency resolution. In an ideal scenario a transform would exist which would give the instantaneous frequency law as seen in Figure 3.1c but unfortunately there is a fundamental limitation on the accuracy achievable to measure this relation. This fundamental limitation in accuracy is due to the uncertainty principal which is stated mathematically as follows,

$$\sigma_t^2 \sigma_f^2 \geq \frac{1}{16\pi^2} \quad (3.7)$$

Where σ_t is the intensity-weighted standard deviation of the signal in the time domain and σ_f is the intensity-weighted standard deviation of the signal in the frequency domain. These are measures of the spread of the signal about their approximate mean in the time- and frequency-domain respectively. Mathematically they are defined as the second moment of the normalised time- and spectral-power density function and can be expressed as follows,

$$\mu_t = \frac{1}{E_s} \int_{-\infty}^{+\infty} t |s(t)|^2 dt \quad (3.8)$$

$$\mu_f = \frac{1}{E_s} \int_{-\infty}^{+\infty} f |S(f)|^2 df \quad (3.9)$$

$$\sigma_t^2 = \frac{1}{E_s} \int_{-\infty}^{+\infty} (t - \mu_t)^2 |s(t)|^2 dt \quad (3.10)$$

$$\sigma_f^2 = \frac{1}{E_s} \int_{-\infty}^{+\infty} (f - \mu_f)^2 |S(f)|^2 df \quad (3.11)$$

$$E_s = \int_{-\infty}^{+\infty} |S(f)|^2 df = \int_{-\infty}^{+\infty} |s(t)|^2 dt \quad (3.12)$$

where, E_s , is the normalising factor and represents the total energy of the signal. Dividing by this factor gives the following density function property, $\int_{-\infty}^{\infty} p(x) dx = 1$, which is required in this case as the signal's power, $|s(t)|^2$, is being treated as a probability density function.

The meaning behind this statement is explained intuitively by the statement that if there is some signal in the time-domain whose energy is mostly concentrated about some point in time, the energy of the signal in the frequency-domain must be highly spread about all frequencies and vice versa. A great example of this can be seen by looking at the Dirac-delta function,

$$\delta(t) = \begin{cases} \infty & t = 0 \\ 0 & t \neq 0 \end{cases} \quad (3.13)$$

The Dirac-delta function is infinitely concentrated at the $t = 0$ point, which leads to the result that $\sigma_t^2 \rightarrow 0$ for the Dirac-delta. Taking the Fourier transform of the Dirac-delta gives the following result,

$$\mathcal{F}\{\delta(t)\} = 1 \quad (3.14)$$

The result is a constant over all frequencies which is essentially the polar opposite to the time-

domain definition of the Dirac-delta function and can be shown by the fact that $\sigma_f^2 \rightarrow \infty$ for the Dirac delta.

3.2 Short-Time Fourier Transform

The basic idea behind time-frequency analysis is most easily understood by first looking into the Short-Time Fourier Transform and spectrogram. This technique looks at short sections of a signal by windowing and then takes the Fourier transform of the signal in this small window. The mathematical formulation of this technique is described as follows for some signal,

$$s_w(t, \tau) = s(\tau)w(\tau - t) \quad (3.15)$$

where $s(\tau)$ is some real signal and $w(\tau - t)$ is some real, even window centred about time, t . Here, τ , is a dummy time variable and t is the actual time. The short-time Fourier transform (STFT) is then defined as follows,

$$F_s^w(t, f) = \mathcal{F}_{\tau \rightarrow f} \{s(\tau)w(\tau - t)\} \quad (3.16)$$

Essentially, the Fourier transform is taken over the windowed signal which is a subset of the original signal centered at time, t .

The short-time Fourier transform image can then be related back to the original time-domain signal according to the following inverse transform,

$$s(t) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} F_s^w(\tau, f) e^{j2\pi ft} d\tau df \quad (3.17)$$

as long as the following condition is met by the window,

$$\int_{-\infty}^{+\infty} w(\tau) d\tau = 1$$

The spectrogram is then defined as the squared magnitude of the STFT and is denoted by $S_s^w(t, f)$,

$$S_s^w(t, f) = |F_s^w(t, f)|^2 \quad (3.18)$$

The window function, $w(\tau - t)$ is the key to the STFT/spectrogram as it allows the localisation in time but also smears the spectrum in frequency [18]. The spectrogram is hard to get right as the window size must be chosen appropriately, as if the window is long compared to frequency variation, then the time resolution will be poor and the instantaneous frequency law will not be observable or if the window is too short, the time resolution will be high and the instantaneous frequency law will be observable but the frequency resolution will not be high enough to show the sideband tones. To illustrate the effect of the spectrogram, a plot of the spectrogram taken with three different window lengths of the FM signal in Figure 3.1a is shown in Figure 3.2. As can be seen in the plot, the larger and smaller of the window lengths give inaccurate views of the instantaneous frequency law, whereas the middle window length gives a clear view of how the instantaneous frequency is changing with time. This highlights how sensitive the STFT is to

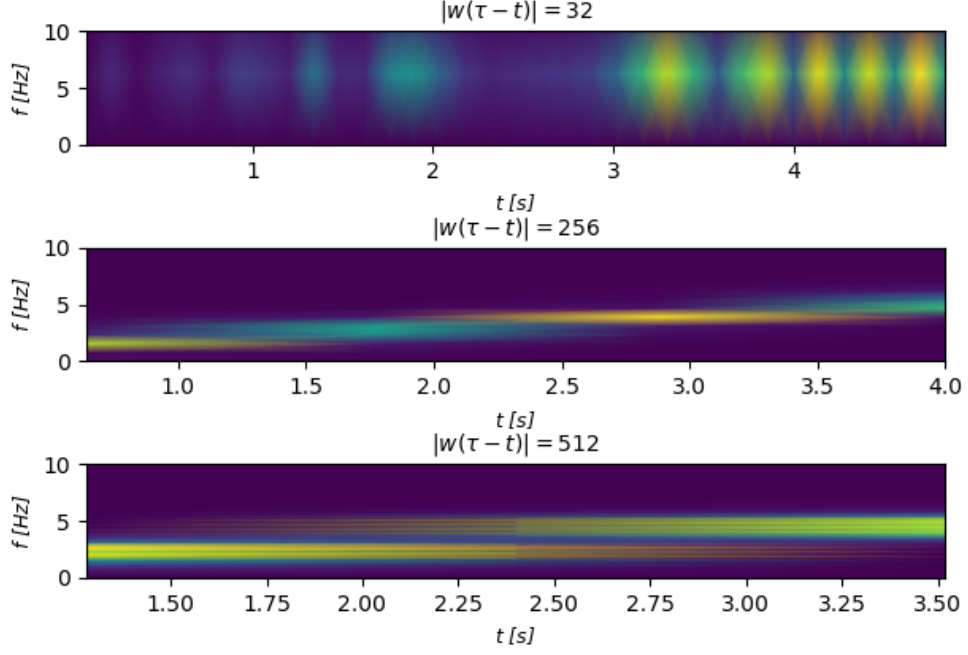


Figure 3.2: Spectrogram of the linear FM signal plotted in Figure 3.1a with three different window lengths used, 32, 256 and 512 samples respectively.

the window length, which is an undesirable feature of this time-frequency analysis technique.

3.3 Wigner-Ville Distribution

The Wigner-Ville distribution (WVD) of a signal is the most general time-frequency analysis technique, as all other techniques can be expressed as some form of the WVD. The WVD is defined as,

$$\rho_z(t, f) \equiv \mathcal{F}_{\tau \rightarrow f} \left\{ z \left(t + \frac{\tau}{2} \right) z^* \left(t - \frac{\tau}{2} \right) \right\} \quad (3.19)$$

$z(t)$ is complex and called the analytic associate of the real input signal $s(t)$. The analytic associate of the input signal is the signal which results from an operation which removes the negative frequency components of the input signal, given that a real signal is Hermitian symmetric in the frequency domain,

$$z(t) = s(t) + j\mathcal{H}\{s(t)\} \quad (3.20)$$

$$\mathcal{H}\{s(t)\} = s(t) * \frac{1}{\pi t} = \mathcal{F}^{-1} \{ (-j \text{sgn}(f)) \mathcal{F}\{s(t)\} \} \quad (3.21)$$

The reason this operation is performed is because the negative frequency components of the signal cause artefacts in the WVD when they are present and thus must be removed. In the WVD definition,

$$K_z(t, \tau) = z \left(t + \frac{\tau}{2} \right) z^* \left(t - \frac{\tau}{2} \right) \quad (3.22)$$

is known as the instantaneous autocorrelation function (IAF) of the $z(t)$ signal. The τ in this function is known as the *lag*. After introducing all of this complexity, it is a good time to show an actual application of this distribution. This can be seen by viewing the plots in Figure 3.3

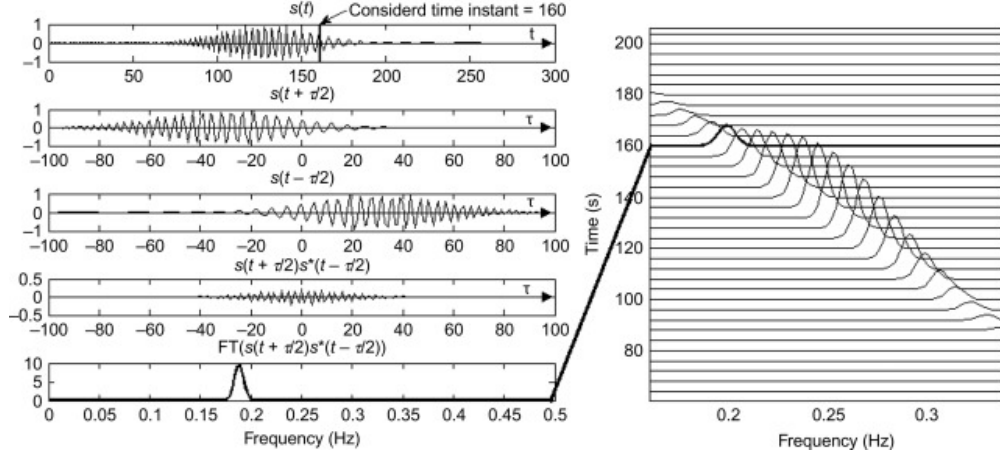


Figure 3.3: Step by step calculation of the Wigner distribution for a linear FM signal. $f_{initial} = 0.4$ Hz, $f_{final} = 0.1$ Hz [18].

which was taken from [18] and shows the intermediate signals produced while calculating the Wigner distribution. The Wigner distribution is simply the Wigner-Ville distribution but using the actual real input signal, $s(t)$ rather than the analytic associate, $z(t)$.

In general for any TFD, $\rho_z(t, f)$,

$$\rho_z(t, f) = \mathcal{F}_{\tau \rightarrow f} \{R_z(t, \tau)\} \quad (3.23)$$

It was shown above that for the Wigner-Ville distribution that,

$$R_z(t, \tau) = K_z(t, \tau)$$

It can be shown that for the spectrogram,

$$R_z(t, \tau) = G(t, \tau) *_t K_z(t, \tau) \quad (3.24)$$

$$G(t, \tau) = w^*(t + \frac{\tau}{2})w(t - \frac{\tau}{2}) \quad (3.25)$$

The function, $G(t, \tau)$ is known as the *time-lag kernel* and the act of convolving this kernel with the IAF is termed *smoothing*. In summary, it has been shown that the spectrogram of a signal's analytic associate can be expressed as a smoothed Wigner-Ville distribution. There are many other kernels that exist and have been derived from other TFD techniques and also many other general window functions which are used as smoothing kernels such as Hanning and Hamming windows. Another common way of representing the smoothing kernel is known as the *doppler-lag kernel*,

$$g(\nu, \tau) = \mathcal{F}_{t \rightarrow \nu} \{G(t, \tau)\} \quad (3.26)$$

Another form of smoothing kernel is to separate the doppler-lag kernel into two separate kernels as is done in [5], meaning that the kernel can be expressed as,

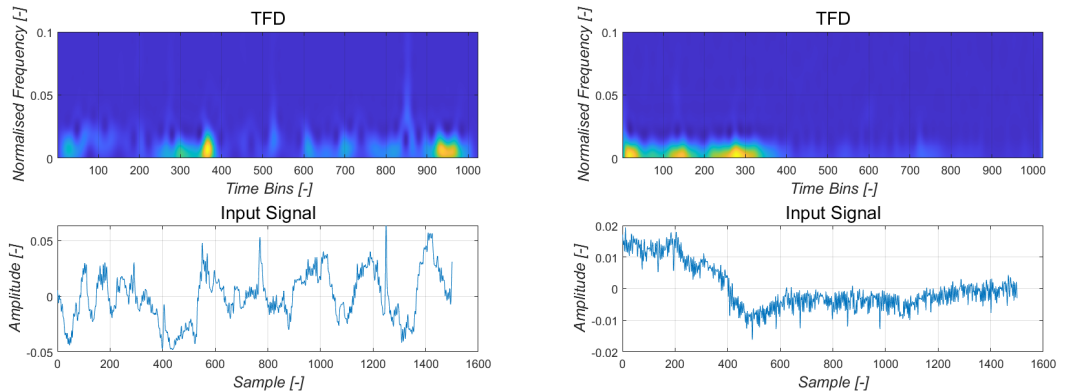
$$g(\nu, \tau) = \gamma(\nu)h(\tau) \quad (3.27)$$

and the smoothing in the time and frequency directions can be independently controlled. When using the memory efficient algorithm for computing the TFD which was kindly provided by Dr. John O'Toole of the INFANT research centre [19], the use of separable kernels allows control of the size of the output matrix of the TFD which directly corresponds to the image size to be input into the neural network. This ability to control the output dimensions gives flexibility in terms of the choice of model.

3.4 Application

It has been shown that there exists a correlation between the low-frequency variation of the HRV signal and the severity of HIE grade [4]. The TFD of the HRV signal will quantify this low-frequency variation. Given that there exists a correlation between HIE grade and the HRV signal, there should be a strong decision boundary that exists between the TFD of a grade four HRV signal and a grade zero or one HRV signal. It was shown in [4] that the more severe the HIE injury, the less variation on the HRV signal. Thus, with this knowledge, one would expect that the TFD of a grade four HRV signal would look a lot less interesting than the TFD of a grade one HRV signal. There is also good evidence that this method should be viable given that in [5], the TFD of the EEG signal was passed into a CNN in order to classify the grade of HIE and achieved the best performance that an HIE grade classifier has achieved to date.

In [5] separable kernels are used with a Hann window for both the lag- and doppler kernels. The length of the Hann window for these kernels was tuned to maximise the performance of the model during training. In order to view the difference between the grades, a plot of the TFD for both a grade four and a grade one should be observed. Looking at Figure 3.4a, which shows a grade one HRV signal and its corresponding TFD, it is clear that there is a lot of power and fluctuation in the low frequency range ($f < 0.5$ Hz). Then in comparison looking at Figure 3.4b, which shows a grade four HRV signal and its corresponding TFD, there is much less fluctuation in general and also in the same frequency range, in comparison to the grade one plot. This is a promising sign as it shows that just from looking at the plots, there is a clear difference between



(a) TFD of a grade one HRV signal. Separable doppler-lag kernel with windows of length 256 samples. (b) TFD of a grade four HRV signal. Separable doppler-lag kernel with windows of length 256 samples.

Figure 3.4: HRV signal and corresponding TFDs of a grade one and grade four subject.

plots that even a person could grade more easily than the time-domain signal.

The method by which time-frequency plots have been computed has been through the use of the algorithm described in [19]. This is a fast and memory-efficient algorithm for the computation of time-frequency analysis distributions. The algorithm works by taking advantage of conjugate symmetries in the time-lag function or the smoothed IAF (Equation 3.24). The algorithm also reduces computational complexity by reducing the level of oversampling during computation. This refers to the fact that a lot of the doppler-lag kernels are concentrated around the origin, with a significant number of zeros away from the origin, which do not need to be computed. This algorithm was implemented in MATLAB and was kindly shared by Dr. O'Toole of the INFANT research centre.

Chapter 4

Deep Learning

Deep learning is sub-component of classical machine learning which utilizes artificial neural networks. An artificial neural network is a network of computational neurons which take inputs through weights and performs a mathematical operation on their inputs. The operation performed by a neuron is essentially a dot product of the input vector with the neuron's weights, the result of this dot product is passed through what is usually a *non-linear* activation function which is necessary for learning otherwise the neuron could be simplified as a linear transformation. A diagram representing the function of a neuron can be seen in Figure 4.1. Deep learning has become very popular in recent years due to the explosion of available of computational devices specifically graphical processing units (GPUs) which are very efficient at performing dot product like operations.

4.1 Classical Machine Learning Simulation

Deep learning can be viewed as a more general view of classical machine learning, as through the use of different activation functions, loss functions, and architectures several other classical machine learning algorithms can be simulated such as regression and support-vector-machines. For the example of simulating a support vector machine the sign function, $\text{sgn}\{\cdot\}$, can be used

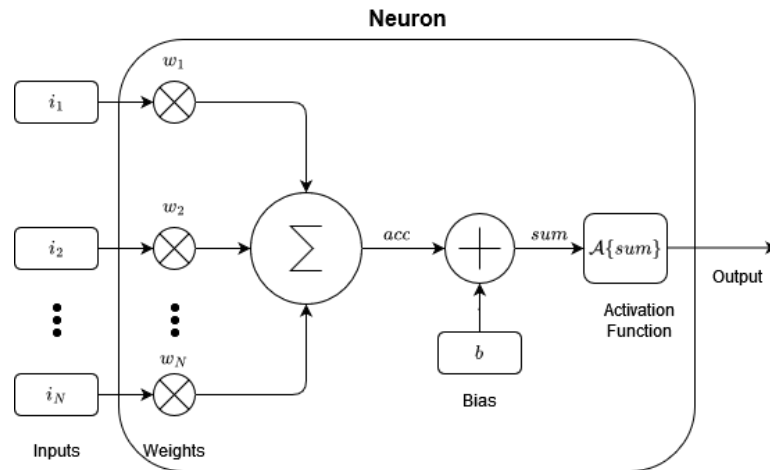


Figure 4.1: Depiction of an artificial neuron which takes, N , inputs

as the activation function which will output values belonging to the set, $\{+1, -1\}$. The loss function used is called hinge-loss and is described mathematically as follows for the, i^{th} , feature,

$$L_i = \max\{1 - y_i(\vec{W} \cdot \vec{X}_i), 0\} \quad (4.1)$$

in which, \vec{W} , is the weight vector for a single neuron, X_i , is the input vector/feature and, y_i , is the feature label. The loss function in Equation 4.1 shows that for a support-vector-machine it is not good enough for the model prediction, $\vec{W} \cdot \vec{X}_i$ to simply be the same sign as the label, y_i , the model prediction must surpass a magnitude of one for non-zero loss which is the same the margin concept for support-vector-machines. A single neuron also called the perceptron is used for the simulation, and mini-batch stochastic gradient descent (SGD) is used for optimisation. In mini-batch stochastic gradient descent, the following algorithm is used to update the weight vectors of the neurons,

$$\vec{W} \leftarrow \vec{W} - \alpha \sum_{(\vec{X}, y) \in \mathbb{S}} \frac{\partial L_{\vec{X}}}{\partial \vec{W}} \quad (4.2)$$

In the general mini-batch SGD algorithm outlined in Equation 4.2, α , is the learning rate of the algorithm which is usually a very small value and, \mathbb{S} is the set of all misclassified features. The reason for subtracting the loss gradient is to move the weight vector in the direction of minimum loss. For the support-vector-machine simulation the algorithm simplifies to the following by substituting Equation 4.1,

$$\vec{W} \leftarrow \vec{W} + \alpha \sum_{(\vec{X}, y) \in \mathbb{S}} y \vec{X} \quad (4.3)$$

4.2 Fully Connected Layers

Using one neuron with the aforementioned loss function and optimisation algorithm gives the same performance achievable through the classical support-vector-machine process. A similar approach can be taken for almost all other classical machine learning algorithms which shows the real power of deep learning.

The next step forward from networks of a single neuron is to use layers of neurons. One of the simplest and most intuitive architectures is a fully-connected layer. This is a layer of neurons in which the output of one neuron is fed as an input into all other neurons in the following layer. An example structure can be seen in Figure 4.2. In Figure 4.2 there is one input layer which is the input to the neural network, two hidden layers which are internal layers which extract features from the input and finally there is one output layer in which the final decision is made. During training the weights and biases of the neurons are adjusted in order to make the output as close as possible to the feature label for some given feature vector input. The weight and bias adjustments are determined by the loss function and the optimisation algorithm.

In general the network's architecture can be modelled as a directed-acyclical-graph (DAG) and the backpropagation algorithm, which is essentially a dynamic programming optimisation of the SGD algorithm introduced above, can be used to calculate weight and bias updates for the whole network. To simplify matters in order to understand how a weight update is computed,

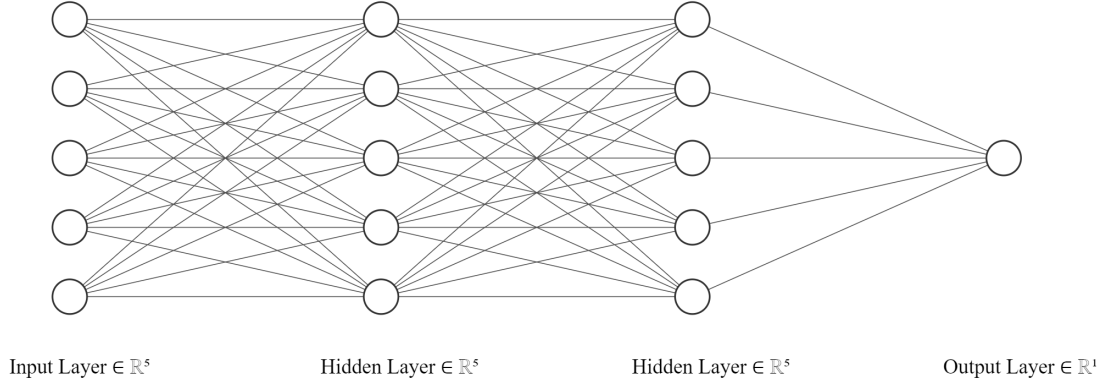


Figure 4.2: Example of a fully connected neural network in which the circles represent neurons and the lines between them represent the connections between neurons

choose one of the weights connecting the input to the first hidden layer in Figure 4.2. In order to calculate the weight update according to the algorithm in Equation 4.2 the derivative of the loss function with respect to that specific weight will have to be calculated or in simpler terms, how much impact does changing that specific weight have on the loss function. There are many different layers and computations to be computed from that specific weight until the loss can be calculated, this can be modelled with a DAG which is shown in Figure 4.3. To calculate the partial derivative of the loss function with respect to the weight of interest, heavy use of the multivariate chain rule is required. The derivation is as follows,

$$\frac{\partial L}{\partial w_{1i}} = \frac{\partial L}{\partial O} \cdot \frac{\partial O}{\partial w_{1i}} \quad (4.4)$$

$$= \frac{\partial L}{\partial O} \left(\frac{\partial O}{\partial h_{21}} \frac{\partial h_{21}}{\partial w_{1i}} + \frac{\partial O}{\partial h_{22}} \frac{\partial h_{22}}{\partial w_{1i}} + \frac{\partial O}{\partial h_{23}} \frac{\partial h_{23}}{\partial w_{1i}} + \frac{\partial O}{\partial h_{24}} \frac{\partial h_{24}}{\partial w_{1i}} + \frac{\partial O}{\partial h_{25}} \frac{\partial h_{25}}{\partial w_{1i}} \right) \quad (4.5)$$

$$= \frac{\partial L}{\partial O} \left(\frac{\partial O}{\partial h_{21}} \frac{\partial h_{21}}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial w_{1i}} + \frac{\partial O}{\partial h_{22}} \frac{\partial h_{22}}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial w_{1i}} + \frac{\partial O}{\partial h_{23}} \frac{\partial h_{23}}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial w_{1i}} + \frac{\partial O}{\partial h_{24}} \frac{\partial h_{24}}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial w_{1i}} + \frac{\partial O}{\partial h_{25}} \frac{\partial h_{25}}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial w_{1i}} \right) \quad (4.6)$$

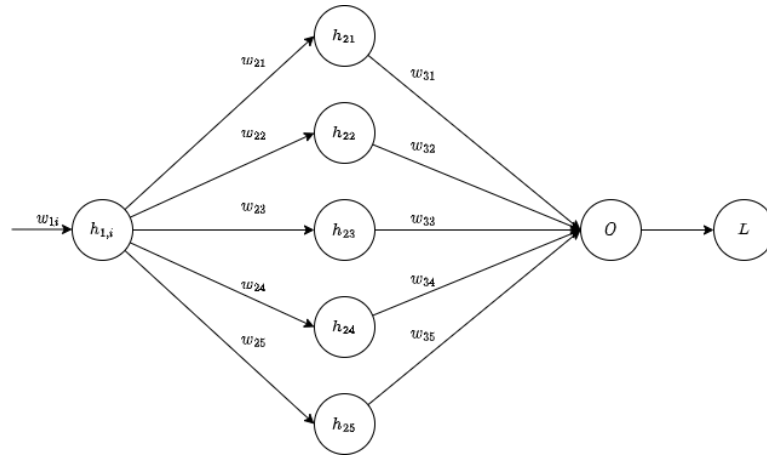


Figure 4.3: Computational graph abstraction from a weight through one full hidden layer a single output neuron/layer and the loss calculation

As can be seen, the expression becomes very complicated even for such a small example network. Each of the product expressions in the final simplification represents a path from the weight of interest to the output. Finally, to look at one more detail of this example, it has been shown how to calculate the partial derivative of the output of a neuron with respect to the output of the neuron before it or a derivative of the form, $\frac{\partial h_{ij}}{\partial h_{i-1k}}$. This kind of derivative is solved by digging into the subcomponents of a neuron and a derivation can be seen below,

$$\begin{aligned}\frac{\partial h_{ij}}{\partial h_{i-1k}} &= \frac{\partial h_{ij}}{\partial a_{ij}} \cdot \frac{\partial a_{ij}}{h_{i-1k}} \\ &= \frac{\partial \phi(a_{ij})}{\partial a_{ij}} \cdot w_{(h_{i-1k} \rightarrow h_{ik})} \\ &= \phi'(a_{ij}) \cdot w_{(h_{i-1k} \rightarrow h_{ik})}\end{aligned}$$

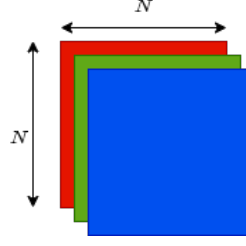
where, h_{ij} , is the i^{th} layer and j^{th} neuron's output, similarly for h_{i-1j} , meaning that in a fully-connected architecture the connection between these neurons is in question, a_{ij} is the i^{th} layer and j^{th} neurons output before being passed through the activation function, $\phi(\cdot)$, is the activation function and finally, $w_{(h_{i-1k} \rightarrow h_{ik})}$, is the weight that connects the neurons in question.

Thus, the computational complexity required to calculate weight updates in a network is very intense especially as the network grows in size. To combat this growing complexity the backpropagation algorithm is used which calculates weight updates in the same fashion as described above but moves backward from the output and takes advantage of some dynamic programming optimisations to compute the weight updates in a recursive fashion.

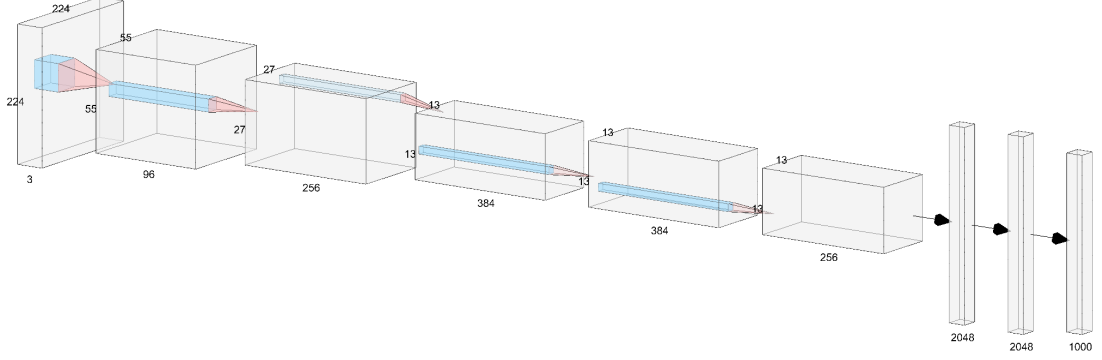
4.3 Convolutional Neural Networks

Fully connected layers are still quite useful but they come with a lot of disadvantages such as their training time and the amount of memory to store the weights. The number of weights used to connect two layers of neurons can be calculated to be, $N_{i-1}N_i$, where, N_j , represents the number of neurons in the j^{th} layer. A flavour of neural networks that are used extensively in the area of image processing and are used in this project are convolutional neural networks (CNNs). One of the advantages of CNNs is that they tend to handle translation invariance reasonably well i.e. they will recognise a dog in an image whether or not the dog is in the centre of the image or the bottom etc. CNNs are designed to work with grid structures such as images. Most CNNs work with RGB images which are three-dimensional tensors of volume, $(N \times N \times 3)$, where each value is an 8-bit number representing the colour intensity of a particular pixel. A diagram representing this can be seen in Figure 4.4a. Thus, the image is more appropriately seen as a three-dimensional tensor. Rather than passing all of the pixel values through neurons, instead a three-dimensional filter is passed over the input tensor, and the filter parameters are convolved with the overlapped image values. Given a filter of size, $(N_f \times N_f \times D_f)$, and an input image of dimensions, $(N \times N \times D)$, the output of the filter will be of size,

$$(N - N_f + 1, N - N_f + 1) \tag{4.7}$$



(a) Depiction of an $(N \times N)$ RGB image in which each coloured square represents a matrix of pixel value which correspond to the colours intensity



(b) Example of convolutional neural network in which a $(224 \times 224 \times 3)$ RGB image is input into the network. The red pyramids intend to show the filter overlap and how the filter's convolution maps to a value in the next layer. At the end of the pipeline two fully-connected layer are seen to make the decision resulting in 1000 output class logits.

Figure 4.4

The output of this filter is a two-dimensional tensor which known as a feature map. In practice several independent filters are used to produce many feature maps, in general if, n , filters are used, n , feature maps are produced or the output can be viewed as a three-dimensional tensor. The parameters in the filters are learned during training and are used to extract features from the input. The convolutional operation performed by the filter can be defined as follows for the p^{th} filter in the q^{th} layer, $W^{(p,q)} = [w_{ijk}^{(p,q)}]$ and the feature maps in the q^{th} layer, $H^{(q)} = [h_{ijk}^{(q)}]$,

$$h_{ijp}^{(q+1)} = \sum_{r=0}^{N_{fq}-1} \sum_{s=0}^{N_{fq}-1} \sum_{k=1}^{D_q} w_{rsk}^{(p,q)} h_{i+r, j+s, k}^q \quad (4.8)$$

where, N_{fq} , is the length and width of the q^{th} layers filters and, D_q , is the depth of the q^{th} layers filters which must all be identical and equal to the depth of the input feature map. The backpropagation algorithm discussed earlier can be again be used to optimise these types of neural networks by expressing the convolutional operation in the form of a matrix multiplication.

Similarly to the neuron described previously, an activation function such as a 'ReLU' is usually applied after a convolutional layer which can be interpreted as the equivalent of the activation function in the neuron after the linear transformation operation. Another common layer used in CNNs are pooling layers, these layers unlike the convolutional layers do not alter the depth of the feature map as they operate on each feature map independently. Pooling layers

pass a window over the feature map and return the maximum value in the window for *max-pooling* and the minimum value in the window for *min-pooling*.

4.4 Transfer Learning

Fortunately, there are many tried and tested CNN architectures out there, a lot of which were trained on the ImageNet dataset [20]. It is common practice, especially in image processing to take a *pre-trained* architecture and to retrain it for a different application. This is known as *transfer learning* and it works quite well because the pre-trained networks have been trained to extract features from an image and to output a predicted label, thus, when retraining the parameters must be modified to extract slightly different features, and some additional layers can be added to alter the number of predicted class etc.

4.5 Architectural Tradeoffs

Transfer learning was used extensively throughout this project as developing a high performance CNN architecture from scratch can take a very long time which was unfortunately not an option. Several architectures were experimented with during the course of the project such as VGG16 [21] and also various flavours of the ResNet architectures such as ResNet18, ResNet34 and ResNet50 [22]. There were many factors to be considered when choosing an architecture and pre-trained model, but the two main points were performance and trainability. In order to determine which architectures gave the best performance, they all had to be tested and their performance measured, but all of the models have a varying number of parameters which determines how much memory they will occupy on the GPU. The GPU in use for training only has 16 GB VRAM memory available which means that the batch size that available for use with a specific model is limited by memory requirements. In Table 4.1 the model and number of parameters for different models experimented with can be seen for comparison.

The reason that the ResNet architectures are able to achieve much lower number of trainable parameters is due to their depth which makes them more suitable for smaller datasets. A problem often encountered in very deep architectures is the ‘vanishing gradient problem’ in which the chain rule multiplications become very long such as in Equation 4.6. If the magnitude many of the partial derivatives is less than one this leads to almost no change to the weight values at the earlier layers in the architecture. This problem is solved in ResNet architectures by adding

Table 4.1: Model and the approximate number of parameters according to measurements calculated using torchvision’s model library [23]. The maximum batch size was determined empirically through training with the models and is a direct function of the GPUs available memory (powers of two used to algorithm efficiency)

Model	No. Parameters	Max. Batch Size
VGG16	138 million	32
ResNet50	25 million	64
ResNet34	21.7 million	128
ResNet18	11.7 million	256

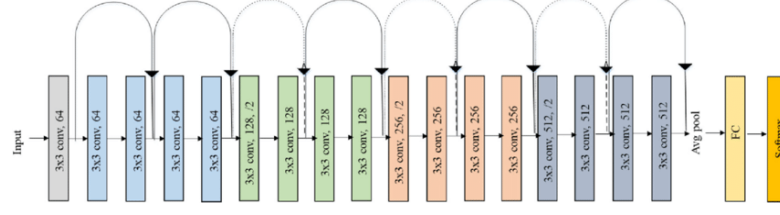


Figure 4.5: ResNet18 architecture which shows the convolutional filters and residual connections between layers to overcome the vanishing gradient problem.

residual connections which allow earlier layers to connect directly to later layers and allows the gradients to propagate more effectively to the earlier layers. This is illustrated in Figure 4.5.

Thus, the trade-off in this design process is that to get better performance we require more memory. Due to the harsh memory restrictions in place, maximum performance must be extracted with minimum memory usage. One way to improve performance which is commonly used is batch normalisation. This method normalises the inputs to the hidden layers in order to reduce internal covariate shift [24]. Mathematically this is described as follows,

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i \quad (4.9)$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad (4.10)$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad (4.11)$$

$$y_i = \gamma \hat{x}_i + \beta \quad (4.12)$$

where the subscript, \mathcal{B} , indicates the batch, m , is the size of a batch, x_i , is the i^{th} feature in the batch, ϵ , is a very small value for numerical stability and both, γ and β are learned parameters. It is obvious that Equations 4.9 and 4.10 are the mean and variance of the batch. Using these statistical properties, Equation 4.11 normalises the value to have zero mean and unity standard deviation. Finally, Equation 4.12 represents the output of the batch normalisation layer, here the learned parameters, γ , and β , can alter the statistical properties of the batch to achieve maximum performance. In order to take advantage of batch normalisation a relatively large batch size is required, otherwise the batch mean and variance calculated will not be representative of the population. Meaning that taking advantage of this performance boost will require increased memory consumption otherwise performance will actually degrade.

4.6 Implementation

All of the aforementioned architectures are freely available to take advantage of with pre-trained weights, but each of the classifiers were trained using the ImageNet dataset meaning that the output has 1000 classes which are far too many for this application in which a binary classifier is being designed. It is very easy to get around this problem by adding an additional fully-connected layer at the end of the network to take the 1000 classes and filter them down to two.

In the actual implementation, the 1000 classes were first filtered to 128 and then another fully-connected layer was used to filter down to two classes. The reason this layered approach was taken was because marginally better performance was achieved with this approach and it makes intuitive sense to gradually filter down to two classes rather than a sudden step from 1000 to 2 classes.

The ImageNet dataset is composed of RGB images which are in the range of, $[0, 255]$, and the networks mentioned above usually perform pre-processing of subtracting the mean of the training set before they enter the network [21], [22]. In order to extract maximum performance the images entering the network should be in a familiar format and numerical range to which the network was trained on. The images being passed in, will of course be the result of a time-frequency distribution on five-minute segments of HRV data, meaning that these are not in RGB format as the values represent signal power at a frequency band and time segment. To counteract this, several transforms were implemented which can be applied to the dataset before use in training and testing. Many transforms were implemented and tested such as linear interpolation using the maximum and minimum values in the complete dataset to shift the time-frequency images to the RGB range of values, normalisation with respect to mean and standard deviation, logarithms to highlight lower values in the time-frequency image etc. The main transforms can be seen in the equations below where all operations are performed element-wise on the image tensors,

$$\text{norm}(x) = \frac{x - \mu_{\mathcal{D}}}{\sigma_{\mathcal{D}}} \quad (4.13)$$

$$\text{abslog}(x) = 10 \log_{10}(|x|) \quad (4.14)$$

$$\text{map}(x) = 255 \left(\frac{x - \min_{\mathcal{D}}}{\max_{\mathcal{D}} - \min_{\mathcal{D}}} \right) \quad (4.15)$$

$$\text{zeromean}(x) = x - \mu_{\mathcal{D}} \quad (4.16)$$

$$(4.17)$$

where, \mathcal{D} , represents the complete dataset, meaning that, $\min_{\mathcal{D}}$, represents the minimum value present in a TFD over to the complete dataset, x , represents the input tensor and the other symbols take their usual meaning. The models also take a very specific image size of (224×224) pixels which meant that the time-frequency images also had to match these dimensions. This can easily be done when using separable kernels for the TFD in combination with the memory-efficient TFD algorithm in use [19], which in this case allows the choice of output dimensions which simplifies matters in this case. Given that the models accepts an RGB input, the actual input volume is $(224 \times 224 \times 3)$ where there are three separate colour channels. The TFD operation produces only one (224×224) image but this can be transformed into a three dimensional tensor by replicating the same image among the three channels.

4.7 Training & Testing

Training refers to the process of determining the optimal filter coefficients and weights for the network. Several optimisation algorithms exist but all of them are based on the backpropagation

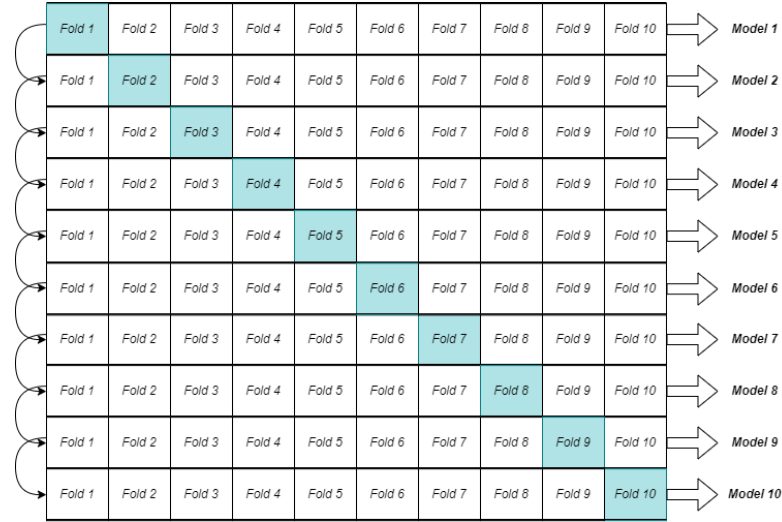


Figure 4.6: Depiction of cross-validation moving from the top row down in which the coloured fold is the fold chosen as the test set and each row produces a separate model

algorithm outlined earlier. In this work, the ADAM algorithm was used for optimisation due to the performance benefits it can provide [25]. The algorithm applies some advanced techniques in combination with the core functionality of backpropagation or stochastic gradient descent as described before. After pre-processing 2 subjects were lost from the dataset due to data loss leaving 118 subjects. Ten fold cross-validation was used to train and report testing results as no validation dataset currently exists. Ten fold cross-validation is a testing method in which the data is split into ten folds and one fold is chosen as the test fold iteratively while the other nine folds are used to train a model. The diagram in Figure 4.6 depicts this process. The folds are split in a subject-independent manner meaning that the data from one subject cannot leave its respective fold in order to avoid bias in the test set due to having seen similar data during training. It was also ensured that each fold would be as close to balanced as possible meaning that the proportion of subjects requiring treatment in the fold would be approximately 50 %. Given that the total dataset is not balanced this is impossible to perfect although the maximum ratio in a fold is 56 %, meaning that there are slightly more subjects requiring treatment than not. Thus, using ten fold cross-validation, ten different models are produced which are all independently tested and their performance is measured. The final results are determined by averaging the performance over all of the models which removes the chance that there was a lucky split in the data which resulted in great performance. The standard deviation of performance metrics such area-under-curve (AUC) for the different models is also a useful metric as a high standard deviation indicates that the results can vary wildly depending on which data is being tested, whereas if the standard deviation is quite low indicates that the performance is relatively constant. The final question to answer with regard to training and testing is at how many epochs to stop.

Firstly, an epoch indicates the number of times that all of the training data was passed through the model and weight updates were made. In this project, early stopping was used which means that a constant number of epochs were completed at which no more training commenced and the model at this point was deemed the output model. The number of epochs to train for was determined by completing several training sessions for 500 epochs and finding

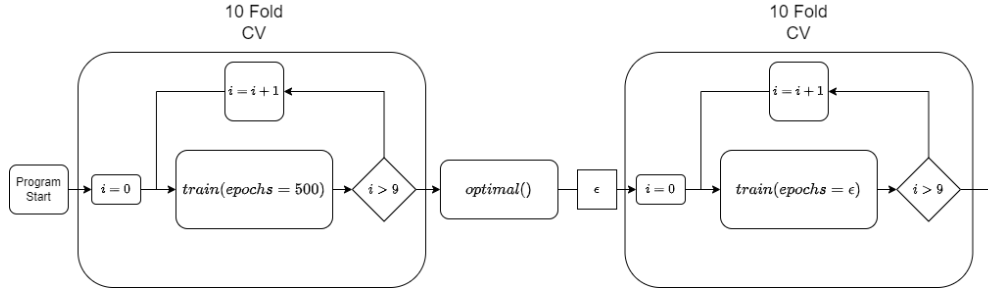


Figure 4.7: Illustration of the ideal early stopping algorithm. In the realistic case only one 10 fold cross-validation is performed in which the number of training epochs is held constant at 50.

the epoch for which on average most of the models achieved highest performance according to their test results after each epoch. On average the results showed that the epoch at which most models achieved simultaneous best performance was approximately the 50th epoch. This result was determined by looking at the curves and determining the average AUC value over all of the models. This method is not ideal and opens up the training to some errors, in the ideal case training would commence for some particular configuration and for a large number of epochs, the optimal stopping point would then be automatically calculated and another separate training instance would begin completely separate from the first one with a different data split. This second training instance would stop at the stopping epoch previously determined, which removes bias from the process of getting the best-performing model. The ideal case is also illustrated in Figure 4.7. It should be noted that the separate data split and the fact that the folds are randomised means that the performance varies depending on the initial conditions. As mentioned, this ideal case was not implemented due to the available computational resources, meaning that a static early stopping point of 50 epochs was chosen and implemented for all model training. This proved to be sufficient but it is clear to see that possible performance benefits could be attained with more computational power.

Chapter 5

Results

As mentioned in Chapter 4, 10 fold cross-validation was used during training which produces 10 different models each one corresponding to a different fold used as the test set. The early stopping point for each training instance was the 50th epoch and the performance of each of the ten models was measured and averaged to calculate the total performance of a particular configuration. The performance metrics used were the receiver-operating-curve (ROC) and area-under-curve (AUC) metrics which are very common metrics used for binary classification due to their ability to deal with imbalanced datasets.

The ROC curve can be calculated by calculating the *true positive rate (TPR)* and the *false positive rate (FPR)* for various thresholds and then plotting this. The *true positive rate* and *false positive rate* are defined below,

$$TPR = \frac{T_P}{T_P + F_P} \quad (5.1)$$

$$FPR = \frac{F_P}{F_P + T_N} \quad (5.2)$$

in which, T_P , refers to the number of correctly identified positive results, F_P , refers to the number of falsely identified positive results and, T_N , refers to the number of correctly identified negative results. Whether the output of the classifier, which is a probability, is identified as positive or negative depends on the threshold used which can be a function of the distribution. For example, the most intuitive case is when the threshold is set to 0.5 in which case an output probability of 0.5 or greater is seen as a positive output. If the model's prediction is correct according to the dataset labels, this will be identified as a true positive etc. The ROC curve is generated by calculating the TPR and FPR for a large number of thresholds and plotting the result. An example of a worst-case and theoretically perfect ROC curve can be seen in Figure 5.1. The AUC is the area under this curve or the integral with respect to the FPR. It is clear to see that in the 'Perfect' case an AUC of 1.0 and in the 'Random' case an AUC of 0.5 is seen.

5.1 Architecture Optimisation

During this project two parameters were varied in order to come to an optimal model, those being the CNN architecture and the combinations of transformations applied to the dataset be-

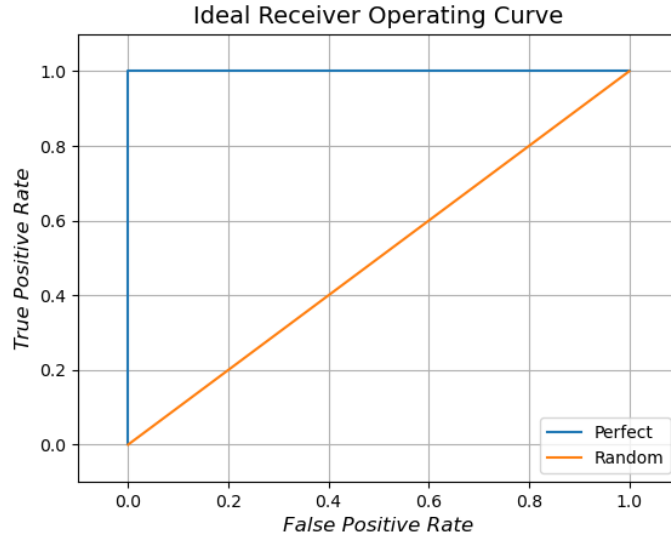


Figure 5.1: Example ROC curve depicting the "perfect" curve in which all positives are correctly identified and all negatives are correctly identified along with the "random" curve in which the model appears to be choosing its output randomly as the TPR and FPR are the same and increase linearly with threshold

fore being passed through the model. There are many other hyper-parameters which could be optimised throughout the system such as the Hampel filter window size and threshold, doppler and lag kernels along with their window lengths, and various modifications to the CNN architecture but not all of this could be done due to the computational power required to achieve this. Firstly, the chosen architecture was determined by means of training using the various models mentioned earlier and determining which model gave optimal performance. Below in Figure 5.2 the various averaged ROC curves for different model choices can be seen. During training these models all had the same transform applied. As can be seen, ResNet18 is the best-performing

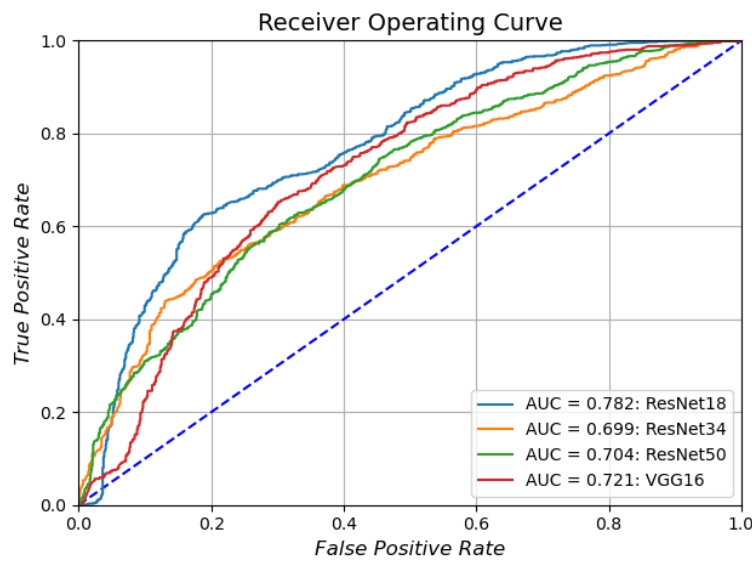


Figure 5.2: ROC plots for various CNN architectures and their respective average AUCs.

architecture with an AUC of 0.782 and it is clear from the plots that the ResNet18 architecture is closer to the optimal ROC curve while ResNet34 and ResNet50 are both the worst-performing architectures and are very close to one another. The reason for this is that these networks are much deeper than ResNet18 meaning that they will require more data to train the filters to extract more complex features. On the other hand, VGG16 is the second best-performing architecture with an AUC of 0.721, meaning that ResNet18 is the best-performing architecture by a reasonable margin. It is interesting that VGG16 is the second best-performing architecture considering that it has the largest number of trainable parameters, although it is not as deep as all other models meaning that it would do better with the lower amount of data at hand.

5.2 Transforms Optimisation

It is clear that ResNet18 is a good compromise between the depth of the architecture and the number of trainable parameters given the amount of data available for training. The next area of optimisation is the combination of transforms to apply to the dataset. The manner in which this was completed is the exact same as described although only using the ResNet18 architecture and simply varying the transforms applied. The ROC curves for these varied transforms can be seen in Figure 5.3. Interestingly almost all of the transforms proved to have detrimental effects on the performance of the classifier with the exception of one, that being the removal of the mean and the mapping of the data to the RGB range of values. The identity transform which is the equivalent of no applied transform proved to be a close second in terms of performance meaning that the use of transformations did not provide extensive performance benefits. It is unsurprising however that the removal of the mean following by mapping to the RGB range of values provided the marginal performance boost that it did, given that the ResNet architectures were

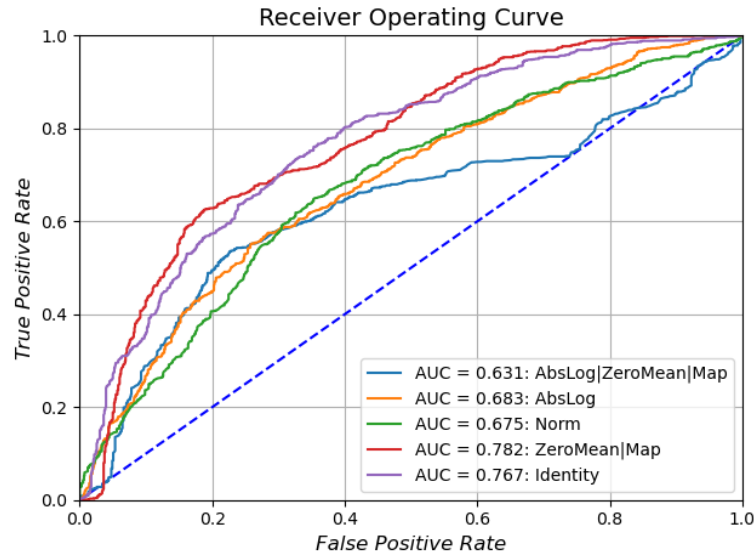


Figure 5.3: ROC plots for various transforms applied to the dataset before training on the ResNet18 architecture and their respective AUCs. Note that the pipe operator ‘|’ is used to imply the order in which transforms were applied e.g. ‘Norm | AbsLog’ implies normalisation followed by the absolute logarithm transform.

Table 5.1: Summary of training results.

Architecture	Transform(s)	AUC	σ
ResNet18	AbsLog ZeroMean Map	0.631	0.180
ResNet18	AbsLog	0.683	0.149
ResNet18	Norm	0.675	0.138
ResNet18	Identity	0.767	0.186
ResNet18	ZeroMean Map	0.782	0.106
ResNet34	ZeroMean Map	0.699	0.132
ResNet50	ZeroMean Map	0.704	0.132
VGG16	ZeroMean Map	0.721	0.156

trained on an RGB image dataset and the only form of preprocessing applied was the element-wise removal of the mean from all of the pixels [22]. Applying the equivalent transforms to the input data makes the time-frequency images appear similar to the image data expected by the model.

However, initially it was surprising that the absolute logarithm transforms had no effect given that they were used in the training of the CNN for use with EEG TFDs [5]. Although after some analysis it became clear that most of the TFD data was in the range of $0 \rightarrow 1$ meaning that upon the absolute logarithm operation, most of these values would be mapped to relatively large in magnitude negative numbers which would over-shadow the larger TFD values which indicate the presence of signal power. It is clear that for use with a pre-trained network, the logarithm transformations are not feasible but this could be changed with a larger dataset and adaptations to the architecture.

A summary of the above results can be seen in Table 5.1. The, σ , column indicates the standard deviation of the AUCs after 10 fold cross-validation. Two examples to better highlight where this calculation is coming from can be seen in Figure 5.4 in which the best and the models with the most and least model variance can be seen. We note that this maps to the overall best-performing system that being ResNet18 with the ZeroMean | Map transform and ResNet18 with the Identity transform, which is a very interesting result because it shows that the transform which maps the image to something more similar to the architecture’s designed-for input gives marginal average performance benefits relative to no transform at all but it does significantly reduce the variance of the models. This is a good sign as it indicates that there is much less luck with regard to the data split and also suggests that the result is far more reproducible. It also shows that the transforms have a significant effect on the performance of the model that are not clearly visible through the average performance over all of the models.

It is visually clear from Figure 5.4 that the use of the transform had a clear impact on the variability of performance as in Figure 5.4b there is one model which even had the performance of being non-randomly incorrect i.e. it learnt the data incorrectly. This is evidently not the case in Figure 5.4a.

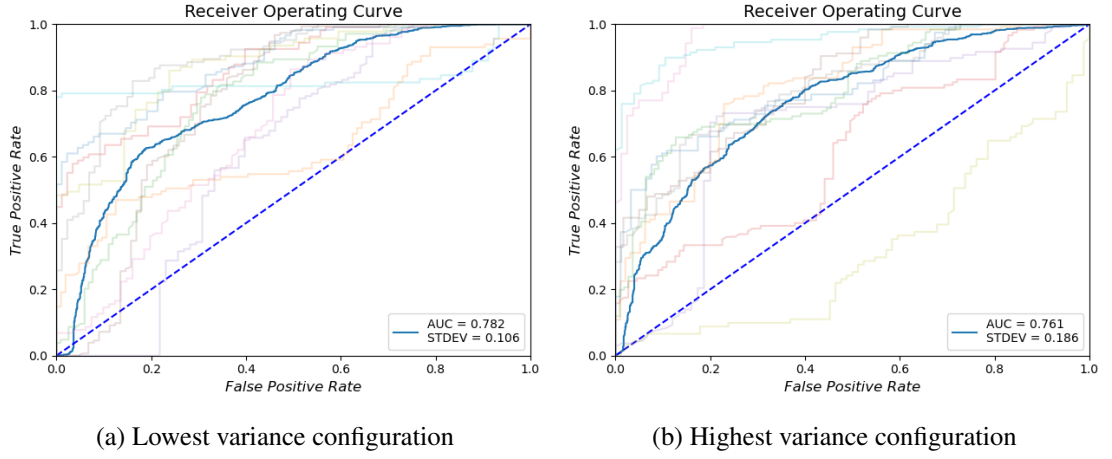


Figure 5.4: ROC plots for all of the models generated during the 10 fold cross-validation (low opacity) for the lowest and highest variance configurations along with their average ROC plot (high opacity)

5.3 Comparison

Finally, a comparison with other systems tested on the same dataset can be seen in Table 5.2. Looking at the comparison it is clear that this work takes the best of the two previous models in that the performance is marginally better than that of ‘Feature CNN’ and the standard deviation, σ , is marginally larger than that of ‘Hilbert-Huang SVM’ meaning that the model developed in this project achieved best in class performance along with a standard deviation in the range as the lowest variant system.

Table 5.2: Comparison of the best performing models which have been trained and tested on the same dataset

Model	Year	AUC	σ
This Work	2023	0.782	0.106
Hilbert-Huang SVM [16]	2022	0.706	0.103
Feature CNN [15]	2021	0.772	0.207

Chapter 6

Software Implementation

6.1 Tools

For this project several software tools were used. From the beginning, the goal was to make heavy use of the Python programming language, as this is generally the language used to build machine learning based systems. For example a very popular machine learning algorithm library called TensorFlow is implemented in the C programming language. Python is extensible, meaning that libraries can be written for Python in C, and Python essentially acts as the user interface for the library. Given that the main focus of the project was deep learning, it made sense to try and use the Python programming language as much as possible to avoid compatibility complications later. The version of Python being used is "Python 3.8.15". The reason for this specific version of Python is that to be able to call MATLAB functions through a Python library, a specific Python version is required to be compatible with the author's MATLAB version.

Another tool which was made great use of throughout this project was MATLAB R2021a. The reason this particular version of MATLAB was used is simply that the author's MATLAB license did not allow further updates. As previously mentioned, the author intended to use Python as much as possible, but in some cases, this was simply not possible. For example the Hampel filter described in Section 2.2 is implemented in MATLAB whereas there is no well-known and community-maintained Hampel filtering function in Python. Although the author could have implemented this function in Python, given that there is time pressure with the project it was decided to use MATLAB's implementation instead. Also as mentioned in Chapter 3, MATLAB was used for the TFD algorithm. This saved a large amount of time and worked quite well.

In order to avoid maintaining two separate implementations, it was decided by the author to find a way to use Python to run the MATLAB scripts. This was done through the `matlab.engine` Python package which allows the calling of MATLAB functions through Python. This way, the author can write the specific MATLAB scripts and simply call them through Python, maintaining the overall Python implementation.

6.2 Structure

An object-oriented approach was taken toward the design of the software. This was done to increase maintainability and also to make interaction with the PyTorch framework easier as an object-oriented approach is also taken during implementation. There are two main libraries of software which were written as part of this project. They can be split into the preprocessing pipeline and the deep learning pipeline. Each of these libraries has several subcomponents which all interact and has an interface for a user driver code. A general flow diagram showing the interaction between the pipelines and their subcomponents is shown in Figure 6.1. Both the pipelines were encapsulated in a single Python package called `anser` for the project but the logic behind their design is summarised in Figure 6.1.

In the diagram, various custom-built classes and functions are shown. A class is indicated by italics in the heading of the block. For example, the *Parse* class in the pre-processing pipeline is shown and is labelled as the parent class. In the class blocks, there are two sections. The upper section is a list of the class attributes, for example the *Parse* class has the attribute `datafile_path`, that being the path to the file containing the HRV data. The lower section describes the methods/functions which the class implements, for example in the *Resample* class, which inherits from the *Parse* class meaning that it has access to all of the methods in the *Parse* class, the `interpolate` method is implemented which does the resampling. The actual attributes and methods implemented are not of importance for this dissertation but they give an understanding of the classes uses. Functions/Scripts are indicated by a rectangle with a vertical line on either side.

The large script block, `dl.py`, is essentially the driver code for the project as it has access to both pipelines. In practice however, `dl.py`, mostly queries the deep learning pipeline to train the model with various different configurations which can be specified by the user as command line arguments. The processing pipeline is more or less stagnant during the training phase as it is more efficient to store the results of pre-processing on disk and simply read them in at runtime. However, the functionality is still there to integrate the pre-processing pipeline into the training process for hyperparameter tuning.

For more information on source code and its implementation, the following public [GitHub Repository](#) can be visited which contains all of the source code visible in Figure 6.1.

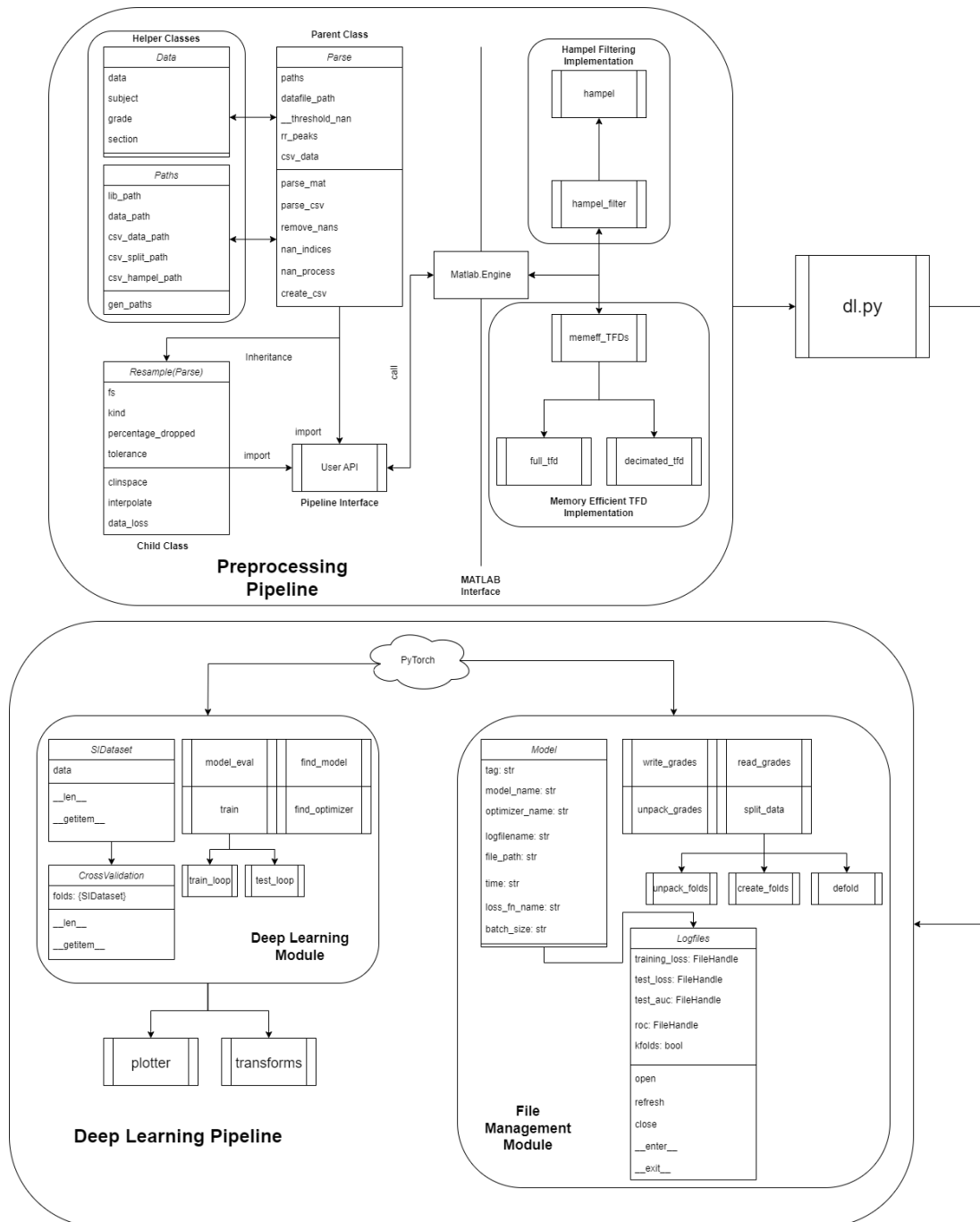


Figure 6.1: Software structure flowchart

Future Work

There are many improvements that could be made on this project in future works. There are two distinct sections to the project those being the preprocessing stage which generates the data to be used for training and testing and the deep learning section which involves choosing an architecture and training a model using the dataset developed in the preprocessing section. Starting from the beginning, major improvements could be made to the quality of the dataset. It is clear that some of the subject's grades are questionable as to whether correct or not and the size of the dataset is relatively small for deep learning applications. In an ideal world a validation dataset would be available, as is the case for EEG, which allows for better comparison of classifiers [8], [26]. Data augmentation could also be used in future works to expand the dataset along with research into using the ECG signal as a whole rather than just the HRV signal.

Unfortunately in this work, there was not much foundation to get started with other than the dataset meaning there was a time constraint to build the preprocessing pipeline while leaving enough time to build the deep learning pipeline. Due to this, not much optimisation was completed in the pre-processing stages such as tuning hyperparameters or testing other methods of regenerating data lost due to NaNs such as autoregression or a moving-average model. Other interpolation methods other than the cubic spline should also be tested such as the cubic-hermite spline. Other filtering methods should be tested in combination with or to replace Hampel filtering and their performance benefits measured. One of the limiting factors of the pre-processing pipeline is the reliance on certain MATLAB functions and libraries such as the Hampel filtering function and the TFA algorithm. A required improvement would be to implement the Hampel filtering and TFA algorithm in the 'Rust' or 'C' programming languages so that these algorithms could be accessed through a Python interface while taking advantage of the performance of a systems programming language, this would drastically speed up the pipeline as the current implementation relies on a python package written by 'Mathworks' for interacting with the MATLAB engine. This improvement would see, along with an immediate performance boost to the speed of the pre-processing pipeline, but also an increase to the usability of the codebase for a move to a server for example which with the current implementation would be difficult due to the reliance on a particular MATLAB version and the license requirements.

On the deep learning side of things more computational power should be acquired to allow more intensive training. Several more intensive techniques were not applicable simply due to the time required to run them such as learning rate scheduling. Several other advanced techniques could be attempted such as adding noise to the two of the other RGB channels which could help with generalisation.

Conclusion

In this dissertation, a classifier was introduced which was able to achieve the best-performance to date in comparison to other systems trained and tested on the same dataset. The classifier achieved an AUC of 0.782 in comparison to the next best 0.772 and also had roughly half the standard deviation.

In order to build this classifier a preprocessing system was built which accounted for various artefacts within the time-series data such as NaNs which represented artefacts annotated by experts and outliers through hampel filtering. The preprocessing system also resampled the signal to a constant 5 Hz sampling rate.

Following this time-frequency analysis was used to generate an image representative of the frequency change over time of the input five-minute HRV segment. The time-frequency distribution was computed using a fast and memory-efficient algorithm in MATLAB and provided by Dr. John O'Toole of the INFANT research centre [19].

Naturally the next step was to build a deep learning system was built using transfer learning in combination with proven architectures which have shown good performance in the ImageNet competition [20]. In particular, the architectures investigated were VGG16, ResNet18, ResNet34, and ResNet50. The final architecture chosen was ResNet18 after completing a 10 fold cross-validation with all of the models and comparing their average AUC and standard deviation. Several transforms were then experimented with before the data was passed through the network to investigate if improved performance could be achieved. It was determined, again through 10 fold cross-validation, that detrending the input image and rescaling its values into the range expected of an RGB image gave best-performance as this is the input format expected by the pre-trained network.

Finally, the networks best-performing configuration was compared against other classifiers which were also trained and tested on the same dataset and also used the same binary classification approach. The results showed that the classifier developed in this work had best-performance in terms of AUC and was extremely close to having the best standard deviation. Although the performance of this classifier was the best it was still within range of the other classifiers showing that it is unlikely that there is some missing feature in the classical approach and that the dataset itself is most likely the main limitation on performance. Overall the results presented in this work are positive proving that this approach is valid and perhaps best for grading HIE with the HRV signal.

Bibliography

- [1] A. M. Qureshi, A. ur Rehman, and T. S. Siddiqi, "Hypoxic ischemic encephalopathy in neonates," *Journal of Ayub Medical College Abbottabad*, vol. 22, no. 4, pp. 190–193, 2010.
- [2] K. A. Allen and D. H. Brandon, "Hypoxic ischemic encephalopathy: Pathophysiology and experimental treatments," *Newborn and infant nursing reviews : NAINR*, vol. 3, no. 11, pp. 125–133, 2011.
- [3] R. Ahmed, "Dynamic classifiers for neonatal brain monitoring," Ph.D. dissertation, University College Cork, 2015, ch. 2.
- [4] R. Goulding, N. Stevenson, D. Murray, V. Livingstone, P. Filan, and G. Boylan, "Heart rate variability in hypoxic ischemic encephalopathy: Correlation with eeg grade and 2-y neurodevelopmental outcome," vol. 77, no. 5, pp. 681–687, 2015.
- [5] S. A. Raurale, G. B. Boylan, S. R. Mathieson, W. P. Marnane, G. Lightbody, and J. M. O'Toole, "Grading hypoxic-ischemic encephalopathy in neonatal eeg with convolutional neural networks and quadratic time–frequency distributions," vol. 18, no. 046007, 2021.
- [6] B. Walsh, D. Murray, and G. Boylan, "The use of conventional eeg for the assessment of hypoxic ischaemic encephalopathy in the newborn: A review," *Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology*, vol. 122, no. 7, pp. 1284–1294, 2011.
- [7] D. Murray, G. Boylan, C. Ryan, and S. Connolly, "Early EEG Findings in Hypoxic-Ischemic Encephalopathy Predict Outcomes at 2 Years," *Pediatrics*, vol. 124, no. 3, e459–e467, Sep. 2009, ISSN: 0031-4005. DOI: 10.1542/peds.2008-2190. eprint: <https://publications.aap.org/pediatrics/article-pdf/124/3/e459/1006347/zpe0090900e459.pdf>. [Online]. Available: <https://doi.org/10.1542/peds.2008-2190>.
- [8] J. Rennie, L. de Vries, and M. Blennow, "Characterisation of neonatal seizures and their treatment using continuous eeg monitoring: A multicentre experience," *Archives of disease in childhood. Fetal and neonatal edition*, vol. 104, DOI: <https://doi.org/10.1136/archdischild-2018-315624>.
- [9] S. Karamian, A. G. and C. J. Wusthoff, "Current and future uses of continuous eeg in the nicu.," *Frontiers in pediatrics*, vol. 9, 2021. DOI: <https://doi.org/10.3389/fped.2021.768670>.
- [10] H. Glass, C. Wusthoff, and R. Shellhaas, "Amplitude-integrated electro-encephalography: The child neurologist's perspective," *Journal of Child Neurology*, vol. 28, no. 10, pp. 1342–1350, 2013, PMID: 23690296. DOI: 10.1177/0883073813488663. eprint: <https://doi.org/10.1177/0883073813488663>. [Online]. Available: <https://doi.org/10.1177/0883073813488663>.
- [11] E. Evans, J. Lerner, R. Sankar, and M. Garg, "Accuracy of amplitude integrated eeg in a neonatal cohort.," *Archives of disease in childhood. Fetal and neonatal edition*, vol. 95, DOI: <https://doi.org/10.1136/adc.2009.165969>.

- [12] G. Boylan, L. Burgoyne, C. Moore, B. O’Flaherty, and J. Rennie, “An international survey of eeg use in the neonatal intensive care unit,” *Acta Paediatrica*, vol. 99, no. 8, pp. 1150–1155, 2010. DOI: <https://doi.org/10.1111/j.1651-2227.2010.01809.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1651-2227.2010.01809.x>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1651-2227.2010.01809.x>.
- [13] “A real-time qrs detection algorithm,” no. 3, 1985.
- [14] R. Ahmed, “Dynamic classifiers for neonatal brain monitoring,” Ph.D. dissertation, University College Cork, 2015, ch. 6.
- [15] D. Crowley, “Grading hypoxic-ischaemic encephalopathic injury in neonates from heartrate variability,” Bachelor’s Thesis, University College Cork, 2021.
- [16] E. Shortiss, “Use of the hilbert-huang transform to analyse heart rate variability for the classification of neonatal brain injury,” M.S. thesis, University College Cork, 2022.
- [17] R. Pearson, Y. Neuvo, J. Astola, and M. Gabbouj, “Generalized hampel filters,” 2016.
- [18] B. Boashash, “Chapter 2 - heuristic formulation of time-frequency distributions,” in *Time-Frequency Signal Analysis and Processing (Second Edition)*, Second Edition, Oxford: Academic Press, 2016, pp. 65–102, ISBN: 978-0-12-398499-9. DOI: <https://doi.org/10.1016/B978-0-12-398499-9.00002-9>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780123984999000029>.
- [19] J. O’ Toole and B. Boashash, “Fast and memory-efficient algorithms for computing quadratic time–frequency distributions,” *Applied and Computational Harmonic Analysis*, vol. 35, no. 2, pp. 350–358, 2013, ISSN: 1063-5203. DOI: <https://doi.org/10.1016/j.acha.2013.01.003>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1063520313000055>.
- [20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.
- [21] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. DOI: 10.48550/ARXIV.1409.1556. [Online]. Available: <https://arxiv.org/abs/1409.1556>.
- [22] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. [Online]. Available: <https://arxiv.org/abs/1512.03385>.
- [23] “Models and pre-trained weights.” (Mar. 2023), [Online]. Available: <https://pytorch.org/vision/stable/models.html>.
- [24] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. [Online]. Available: <http://arxiv.org/abs/1502.03167>.
- [25] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].
- [26] A. Pavel, J. Rennie, and L. de Vries, “A machine-learning algorithm for neonatal seizure recognition: A multicentre, randomised, controlled trial,” *The Lancet. Child & adolescent health*, vol. 4, DOI: [https://doi.org/10.1016/S2352-4642\(20\)30239-X](https://doi.org/10.1016/S2352-4642(20)30239-X).

- [27] R. Ahmed, "Dynamic classifiers for neonatal brain monitoring," Ph.D. dissertation, University College Cork, 2015, ch. 3.
- [28] "Dive into deep learning." (Sep. 2022), [Online]. Available: <https://d2l.ai/>.
- [29] I. Goodfellow, Y. Bengio, and A. Courville. "Deep learning." (Sep. 2022), [Online]. Available: <https://www.deeplearningbook.org/>.
- [30] "Interpolate." (Oct. 2022), [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/interpolate.html>.
- [31] M. Mesbah, B. Boashash, M. Balakrishnan, and P. B. Coldiz, "Heart rate variability time-frequency analysis for newborn seizure detection," in *Advanced Biosignal Processing*, A. Naït-Ali, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 95–121, ISBN: 978-3-540-89506-0. DOI: 10.1007/978-3-540-89506-0_5. [Online]. Available: https://doi.org/10.1007/978-3-540-89506-0_5.
- [32] B. Boashash, "Chapter 1 - time-frequency and instantaneous frequency concepts," in *Time-Frequency Signal Analysis and Processing (Second Edition)*, Second Edition, Oxford: Academic Press, 2016, pp. 31–63, ISBN: 978-0-12-398499-9. DOI: <https://doi.org/10.1016/B978-0-12-398499-9.00001-7>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780123984999000017>.
- [33] L. Cohen, "The time and frequency description of signals," in *Time Frequency Analysis*. Prentice Hall, 1995.
- [34] "Neonate." (Nov. 2022), [Online]. Available: https://www.datadictionary.nhs.uk/nhs_business_definitions/neonate.html.
- [35] "Seizures." (Nov. 2022), [Online]. Available: <https://www.mayoclinic.org/diseases-conditions/seizure/symptoms-causes/syc-20365711>.
- [36] "Electrocardiogram (ecg or ekg)." (Nov. 2022), [Online]. Available: <https://www.mayoclinic.org/tests-procedures/ekg/about/pac-20384983>.
- [37] "Guidance on anonymisation and pseudonymisation." (Nov. 2022), [Online]. Available: <https://www.dataprotection.ie/en/dpc-guidance/anonymisation-and-pseudonymisation>.
- [38] Y. Wu. "Clinical features, diagnosis, and treatment of neonatal encephalopathy." (Nov. 2022), [Online]. Available: <https://www.uptodate.com/contents/clinical-features-diagnosis-and-treatment-of-neonatal-encephalopathy>.
- [39] D. M. Murray, G. B. Boylan, I. Ali, C. A. Ryan, B. P. Murphy, and S. Connolly, "Defining the gap between electrographic seizure burden, clinical expression and staff recognition of neonatal seizures," *Archives of Disease in Childhood - Fetal and Neonatal Edition*, vol. 93, no. 3, F187–F191, 2008, ISSN: 1359-2998. DOI: 10.1136/adc.2005.086314. eprint: <https://fn.bmj.com/content/93/3/F187.full.pdf>. [Online]. Available: <https://fn.bmj.com/content/93/3/F187>.
- [40] C.-C. Lin, C.-Y. Yang, Z. Zhou, and S. Wu, "Intelligent health monitoring system based on smart clothing," *International Journal of Distributed Sensor Networks*, vol. 14, p. 1550147718794318, Aug. 2018. DOI: 10.1177/1550147718794318.
- [41] R. Ahmed, A. Temko, W. Marnane, G. Lightbody, and G. Boylan, "Grading hypoxic-ischemic encephalopathy severity in neonatal eeg using gmm supervectors and the support vector machine," *Clinical Neurophysiology*, vol. 127, no. 1, pp. 297–309, 2016, ISSN: 1388-2457. DOI: <https://doi.org/10.1016/j.clinph.2015.05.024>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1388245715006215>.

- [42] C. C. Aggarwal, “An introduction to neural networks,” in *Neural Networks and Deep Learning*. Gewerbstrasse 11, 6330 Cham, Switzerland: Springer International Publishing, 2018, pp. 1–51, ISBN: 978-3-319-94462-3. DOI: 10 . 1007 / 978 - 3 - 319 - 94463 - 0. [Online]. Available: <https://doi.org/10.1007/978-3-319-94463-0>.
- [43] C. M. Bishop, “Single layer networks,” in *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995, pp. 98–105.
- [44] “Learn the basics.” (Jan. 2023), [Online]. Available: <https://pytorch.org/tutorials/beginner/basics/intro.html>.
- [45] “Pytorch homepage.” (Mar. 2023), [Online]. Available: <https://pytorch.org/>.
- [46] “Visualizing models, data, and training with tensorboard.” (Jan. 2023), [Online]. Available: https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html.
- [47] M. Simon, E. Rodner, and J. Denzler, *Imagenet pre-trained models with batch normalization*, 2016. DOI: 10 . 48550 / ARXIV . 1612 . 01452. [Online]. Available: <https://arxiv.org/abs/1612.01452>.

Appendix A

Data Approvals

COISTE EITICE UM THAIGHDE CLINICIÚIL
Clinical Research Ethics Committee of the Cork Teaching Hospitals

Tel: +353-21-4901901
Email: crec@ucc.ie

University College Cork
Lancaster Hall
6 Little Hanover Street
Cork
Ireland

Our Ref ECM 4 (aaaaa) 07/05/13
& ECM 3 (mm) 01/11/2022

9th October 2022

Professor Geraldine Boylan
Department of Paediatrics and Child Health
Cork University Maternity Hospital
Wilton
Cork


Re: ANSeR1: Algorithm for neonatal seizure recognition.

Dear Professor Boylan

The following amendment has been approved:

- Cover Letter dated 6th October 2022
- Amendment Application Form signed 6th October 2022
- Addition of Emmet Horgan, Engineering Masters Student as a co-investigator in the study.

Yours sincerely



Professor David Kerins
Chairman
Clinical Research Ethics Committee
of the Cork Teaching Hospital

(a) ANSeR-1 Approval

COISTE EITICE UM THAIGHDE CLINICIÚIL
Clinical Research Ethics Committee of the Cork Teaching Hospitals

Tel: +353-21-4901901
Email: crec@ucc.ie

University College Cork
Lancaster Hall
6 Little Hanover Street
Cork
Ireland

Our Ref ECM 3 (www) 06/06/17
& ECM 3 (nn) 01/11/2022

10th October 2022

Professor Geraldine Boylan
Department of Paediatrics and Child Health
Cork University Maternity Hospital
Wilton
Cork


Re: ANSeR2: A multi-centre, randomisation, controlled, clinical investigation of a standalone decision support Algorithm for neonatal seizure recognition – Neurodevelopmental follow up.

Dear Professor Boylan

The following amendment has been approved:

- Cover Letter dated 6th October 2022
- Amendment Application Form signed 6th October 2022
- Addition of Emmet Horgan, Engineering Masters Student as a co-investigator in the study.

Yours sincerely



Professor David Kerins
Chairman
Clinical Research Ethics Committee
of the Cork Teaching Hospital

(b) ANSeR-2 Approval

Appendix B

Logbook

Semester 1

Week 1

Readings

- R. Ahmed, "Dynamic classifiers for neonatal brain monitoring," Ph.D. dissertation, University College Cork, 2015, ch. 2.

Paper Summaries

[3] This week I read Chapter 1 & 2. Chapter 1 contained a literature review of the techniques used up to now which I did not find particularly interesting as I do not yet have the background to understand the material fully. Although I did note that there was no mention of any deep learning techniques used and Dr. Lightbody has mentioned to me previously that deep learning has not been used in this area thus far. Chapter 2 was of much more interest to me and detailed the medical information required, such as explaining that Hypoxic-Ischemic Encephalopathy (HIE) is a non-specific term for brain dysfunction caused by a lack of blood flow and oxygen to the brain. Some chilling facts were also presented such as 10% of new-borns require admission to neonatal intensive care units (NICUs) which is a much larger percentage than I would have originally estimated and that HIE is the second leading cause of neonatal deaths globally (10.5%). This information helped me to understand the nature of the problem at a large scale.

The paper then continued by detailing how a clinician would attempt to diagnose HIE in a neonate, how the severity of the brain injury would be graded, and the treatment of the neonate once a diagnosis is made. What I found striking about this section was detailing the process of monitoring a neonate and grading an injury. The process usually begins by taking an EEG reading of the neonate to measure their brain waves, this is the best method of monitoring the progress of the injury continuously as other methods such as an MRI can only take a snapshot of brain activity. Typically these EEG recordings will last for 24 hours, and a neurologist must analyse this recording in 10-30 second windows in order to identify unusual activity which could be a symptom of HIE. This gave me a very good understanding of my role within this area, in terms of helping a neurologist to make these diagnoses.

Progress

- This week I met Dr. Lightbody and had a brief discussion about the flow of the project. He gave me a PhD thesis to read [3] and also emailed me some other students' previous dissertations to read.
- During the week I read through [3] and gained a reasonably good understanding of the problem at hand. I feel as though the medical background given was thorough enough and also 'beginner friendly'. I took notes on everything new I learned.

Plan

- Next week I plan on speaking to Dr. Lightbody on Monday at 2 p.m. which is our agreed upon time, I hope to learn a little bit more about the directions which I can take this project.
- I also plan on moving onto the next section of [3] which is the actual technical section describing the basics of support vector machines. I will need to briefly cover this section before I can read into Chapter 5 & 8 which contains relevant information to my project but the implementation used was a support vector machine. I believe that this will be useful information nonetheless.
- Must start reading into the other FYP reports in order to get some information on what I must learn to pre-process the data. Having spoken with Dr. Lightbody, I feel that getting the data prepared for the neural network will be a large part of this project.

Week 2

Readings

- R. Ahmed, “Dynamic classifiers for neonatal brain monitoring,” Ph.D. dissertation, University College Cork, 2015, ch. 3.
- I. Goodfellow, Y. Bengio, and A. Courville. “Deep learning.” (Sep. 2022), [Online]. Available: <https://www.deeplearningbook.org/>.
- E. Shortiss, “Use of the hilbert-huang transform to analyse heart rate variability for the classification of neonatal brain injury,” M.S. thesis, University College Cork, 2022.

Paper Summaries

[27] This week I read Chapter 3. I found it to be a very mathematically heavy chapter for which I do have some of the mathematical optimisation background to fully understand. Although I did find the chapter to give a very good overview of these technical topics such that I feel prepared to read onto the application of these techniques as I understand the purpose behind them albeit not the specific mechanics of their implementation. There was a derivation of the decision boundary equation used in support vector machines, specifically, $f_{svm}(\vec{x}) = \text{sign}(\sum_{i=1}^{\infty} \alpha_i y_i (\vec{x}_i \cdot \vec{x}_j) + b)$, and also an explanation of how kernel functions could be taken advantage of in SVM to map data to a higher dimension in which it could be separated with a hyperplane. Following this was an explanation of sequential kernels which could be used to extract contextual information from a sequence of feature vectors. A description of various sequential kernels was then given. The chapter as a whole was very interesting and gave a both a high level description and also a very technical description of the techniques used.

[29] I spent a few hours on Monday revising concepts from Part 1: Probability and Information Theory. I found the information to be very useful and right to the level which I required. I revised some concepts such as discrete and continuous random variables, probability mass functions and probability density functions, marginal probability distributions, conditional probability and how this leads to Bayes’s Theorem, and finally Expectation, Variance and Covariance. Overall I found the material to be thorough and informative and a good foundation for the work ahead.

Progress

- I started off this week by identifying a weak area of mine mathematically which is used extensively in the area of deep learning, that being statistics and probability. I decided it would be good to take a few hours before I dig into research papers and revise some concepts and remind myself of terminology. Monday morning I did just that, using the background probability section of [29], which I found covered almost everything I was looking for. I felt as though I had to hold myself back from diving too deep into probability theory and also linear algebra as I know that this is a black hole of time-wasting for me as the study will never end, resulting in work not being produced.
- On Monday at 2 p.m. I met with Dr. Lightbody, as this was a pre-arranged meeting time. We discussed all things to do with the project for about 55 minutes. After the meeting I felt quite excited about the project and noted down some key areas I had to look into such as, spectrography, wavelets, downsampling and pre-processing in general. I also looked into the specifics of what I have to provide for the project and devised a system in which I could document everything, I found the logbook structure in [16] to be very well laid out and decided to also follow this structure. This system involved writing by hand, what I am working on, and reviewing papers which I had read. I also allocated some time Friday mornings to transcript everything I had noted down during the week. On Friday I also set-up a system using \LaTeX which allowed me to note down all my references into a single file which I can use for my dissertation as well as my logbook. I also realised the importance of using my time wisely during this relatively calm period, so that I can read and learn as much as possible with regards to my FYP. I noted down some key papers which I need to finish reading and set myself the task of finding more papers to read especially in the fields of biomedical signal processing and deep learning.
- During this week I also spent a great deal of time reading [27] but became bogged down reading the section explaining the theory behind Support Vector Machines (SVMs). I was using a lot of time looking into the

mathematics behind optimisation and Lagrange Multipliers, but realised that I was essentially wasting my time as I could study that material for a very long time and my understanding of it does not contribute anything to the project, so I decided to skim over the SVM material and become familiar and get a basic understanding so that I could properly understand the rest of the paper, but not spend too much time researching the area.

- I was able to use information from [16] to write a parsing function which should be able to parse the HRV data when I get access to it.

Plan

- The data must be looked at manually to identify missing data points and anything that looks wrong in general.
- I need to create an approximate timeline of events which I can attempt to work towards throughout the semester.
- I need to start thinking about writing and structuring the interim report and possibly writing background and theoretical sections as I learn the material.
- I need to further research into the how the HRV signal is created and possible problems which I must solve regarding the signal.
- Finally I need to research into a suitable method of performing a frequency analysis on the data to generate an image matrix which can be inputted into a convolutional neural network.

Week 3

Readings

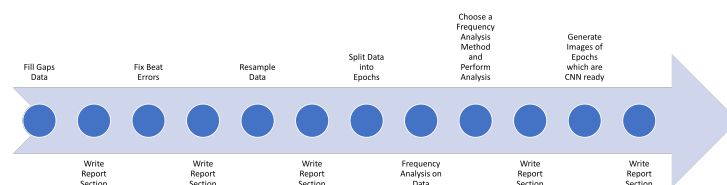
- R. Ahmed, "Dynamic classifiers for neonatal brain monitoring," Ph.D. dissertation, University College Cork, 2015, ch. 6.
- E. Shortiss, "Use of the hilbert-huang transform to analyse heart rate variability for the classification of neonatal brain injury," M.S. thesis, University College Cork, 2022.
- "Interpolate." (Oct. 2022), [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/interpolate.html>.

Paper Summaries

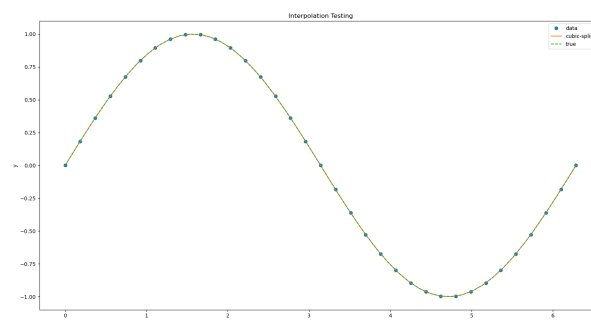
[14] This week I read Chapter 6. I decided on this chapter as it is most related to my work. In fact I honed in on the beginning section where the motivation for using heart-rate variability is discussed and also the data set and pre-processing used as these directly pertained to my research. I found these sections to be particularly interesting to me. During my reading of the motivation section I found myself asking the question of why not feed the raw ECG signal to a deep learning model as there could be other features which the deep learning algorithms could extract that we do not know about. But after some thinking, I came to the conclusion that the ECG signal is intricately related to the functioning of the body as a whole meaning there is a lot that could be misinterpreted based on the training data, and given that we know a strong correlation exists between HRV and HIE, it would be foolish to not take advantage of that correlation, early on in the research area. I also learned a bit more about the ethics of receiving the data itself such as collecting written parental consent and also approval from the Clinical Ethics Committee of the Cork Teaching Hospitals for *each* neonate. From this paper I also learned a lot about the ECG artefacts and what corresponding HRV signals looked like based on an ECG artefact. I feel as though I will need to discuss further with Dr. Lightbody about this as I am not sure how to go about filtering out some of these artefacts and do not feel particularly confident about it. I learned that a Hermite-Spline was used to interpolate the data and subsequently resample it. I am not sure about what a Hermite-Spline is and may need to do some more research such as whether the current cubic spline I am using is inferior.

Progress

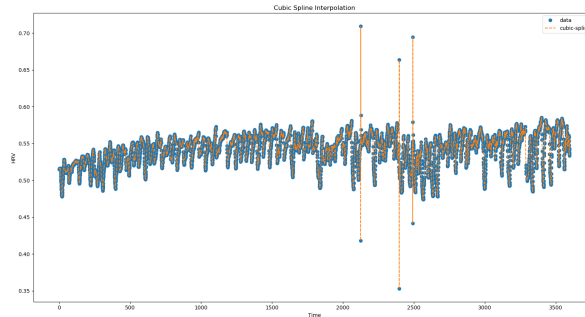
- I found that this week I did not get a lot of reading done which is dissapointing and I believe I will need to improve on this.
- Earlier this week I contacted Eimear Shortiss who authored [16]. She worked on the same data as I am currently working on and thus had to have dealt with similar problems. I messaged her asking if she might be able to share some of the code she used to pre-process the data which would save me a great deal of time. She replied saying saying that it would not be a problem and she would look for it and email me with what she has on Saturday.
- Earlier in the week I also decided to re-structure my logbook as I felt that it was not structured enough and certain information would slip through the cracks. Thus, I re-wrote my earlier sections using a structure inspired by [16].
- I was able to fully dedicate a few hours to writing the code to parse, plot and some other functions to modify and process the data. In particular I wrote a code which could identify and remove NaN types, this particularly useful as a lot of the data contains these values which will have to be removed and replaced with an approximate.
- I created a approximate timeline events. I did not add dates to events as of yet but more so specified the order and flow of the task which are ahead. The timeline can be seen below,



- I decided to write some preliminary code to get started on resampling the HRV signals at a constant rate. To do this I looking into the `SciPy` library which had an interpolation function [30] which can be used to create an interpolation model of the signal using the cubic spline method. I experimented with this function on a sine wave which I generated and got the following results,



Once I felt I had an understanding of how to use the function, I decided to test it out on some of the actual data. I created a cubic spline model of the data and then re-sampled the signal at 256 Hz. Plotting the results appeared the give good results as can be seen below,



Plan

- Read the papers which I have amalgamated and review them.
- Write up a section on the Pan-Tompkins algorithm for the dissertation.
- Wrap the code which I have written for parsing the data into a class which will be easier to work with in the future.
- Finish writing the section for interpolating and resampling the code.
- Test out some FFTs and Wavelet transforms on simple signals to get a feel for how they work and how to process the information, and then apply this to some of the data.
- Look into how to remove some of the signal artefacts.
- Read into the code sent by Eimear and extract what could be useful.

Week 4

Readings

- R. Goulding, N. Stevenson, D. Murray, V. Livingstone, P. Filan, and G. Boylan, “Heart rate variability in hypoxic ischemic encephalopathy: Correlation with eeg grade and 2-y neurodevelopmental outcome,” vol. 77, no. 5, pp. 681–687, 2015.

Paper Summaries

[4] The purpose of this paper was to analyse the correlation between heart variability and both HIE grading and neurological outcome. The measure of correlation was through Spearman’s Correlation Coefficient, which measures monotonic relationships whether linear or not. A lot of the paper contained information which I already knew, but there was also new bits of information and I learned more about how certain HRV features are correlated with HIE. For example, the low-frequency to high-frequency ratio correlates poorly with HRV grade while the HF and LF features themselves correlate well. I also read of the fact that there was a strong decision boundary between healthy and mild HIE neonates, and moderate to severe HIE neonates. This was very useful for me as it goes well with my previous discussion with Dr. Lightbody in which he suggested that the use of a binary classifier CNN could give the best performance as the group had achieved good results when classifying based on when to treat the neonate, which lies of that decision boundary. This also makes intuitive sense to me, as the current grading system in place is forced to be linear i.e. grade 0-4. But there is no evidence, which I have seen, to suggest that the order of severity of HIE progresses linearly, and by that I mean that the current grading system suggests that a grade 2 HIE case is one-unit magnitude worse than a grade 3 case, but in reality the difference in outcome could be exponentially worse. Thus, making a classifier which maps to this linear grading system would be quite difficult as you are mapping what could be a non-linear severity score to a linear scoring system. Choosing to classify based on the grounds of when to treat the patient makes intuitive sense to me as this is a non-subjective and definite point in the scale, i.e. perhaps the point where the non-linear severity scale and linear scale cross paths.

In the study they mention that all of the data was taken before the introduction of therapeutic hypothermia to the neonates who required such treatment, this is an important fact as far as I know, the data which I am using does not have this added benefit meaning that some of the subject could have been treated and this can not be included in the model. This could prove to be a reason perhaps why eventual neural network may not have the performance that is desired.

As previously mentioned, the frequency domain features performed well in terms of correlation with HIE grade. This was encouraging to read as eventually the training set which will be used for this project will contain spectrograms of the data which will solely be in the frequency domain.

Progress

- I had a very productive meeting with Dr. Lightbody on Monday. We discussed in detail the faults I had made in how I pre-processed the data. I had essentially removed the NaNs from the data and then proceeded to splice the data together. This not a good approach due to the manner with which cubic splines are calculated which is used later to re-sample the data at a higher rate. These splices essentially cause discontinuities in the data and the cubic-spline algorithm will attempt to fit a cubic polynomial to the discontinuity which will not be accurate and could lead to large spikes if there is a sudden step change. The solution which we devised was to set a threshold, for example two NaNs, over which linear interpolation could be used to replace that lost information. Anything over this threshold should result in a data split. What this means is for example if there one subject with a number of NaNs in the centre of the data which is greater than the threshold NaNs *allowed in a row*, then the data should be split into two files, the first half of the data containing the information before the NaNs and the second half, containing the information after the NaNs, and of course throwing away the NaN data. To automate this task was actually more difficult than I expected as I had to write some obscure algorithms which I had not considered before, such as an algorithm which, takes a list of indices, creates another list containing sub-lists of these indices where each sub-list contains the indices in increasing order. For example, starting with the list, [1, 2, 3, 4, 8, 9, 10, 11, 50, 90, 91, 92], the result would be the following list, [[1, 2, 3, 4], [8, 9, 10, 11], [50], [90, 91, 92]]. The algorithm then had to determine whether or not any of these lists were larger in length than the threshold and if so, interpolate and replace the data, otherwise create the sub-files. This took approximately three days to get in working order after most of the bugs had been found and fixed.
- As reviewed above, I also read a paper, but unfortunately did not get through the papers which I had assigned myself to read, as I spend too much time working on solving the issues with the aforementioned algorithm.

Plan

- Read at least two papers from my list including the original Pan-Tompkins algorithm paper
- Write code to use a Hampel filter on the data to remove the spikes which are parts of the signal we are not interested in and could affect performance when training the neural network as the algorithms may become interested in these sudden spikes, while we know there is a correlation with the slow moving variation of the data.
- Write code to re-sample the Hampel filtered and NaN processed data.
- Test out some FFTs and Wavelet transforms on simple signals (did not get a chance to do this last week)
- Begin some sections of the report, for example the sections describing the pre-processing which you have done thus far and also on an introduction, perhaps even laying out the key-points you would like to get across in the introduction.

Week 5

Readings

- R. Pearson, Y. Neuvo, J. Astola, and M. Gabbouj, "Generalized hampel filters," 2016.
- "A real-time qrs detection algorithm," no. 3, 1985.

Paper Summaries

[17] This paper introduces the "Generalized Hampel Filter" as being a superset of the standard median filter. It shows the differences between the hampel filter and the standard median filter, the main difference being that the hampel filter has an extra hyper parameter on top of the window length, that being the thresh-hold. It is also shown that the hampel filter can be reduced to the median filter by setting the thresh-hold to zero. The paper then goes onto show exactly how this hampel filter works. Given some data window,

$$\mathbf{W}_k^K = \{x_{k-K}, \dots, x_k, \dots, x_{k+K}\}$$

The median of the window is defined as,

$$m_k = \text{median}\{\mathbf{W}_k^K\} = \text{median}\{x_{k-K}, \dots, x_k, \dots, x_{k+K}\}$$

In a standard median filter, once the window median is calculated, the central window element is replaced with the median,

$$x_k = m_k$$

But in a hampel filter there is a thresh-hold introduced which will only replace the central elements if a certain condition is met, namely,

$$|x_k - m_k| \geq tS_k$$

Where t is the thresh-hold value and S_k is the MAD scale estimator, defined as follows,

$$S_k = 1.4826 \times \text{median}_{j \in [-K, K]} \{|x_{k-j} - m_k|\}$$

This is essentially an estimate of the standard deviation of the window i.e. $\hat{\sigma} = k \times MAD$. Then the output of the hampel filter is defined as,

$$y_k = \begin{cases} x_k & |x_k - m_k| \leq tS_k \\ m_k & |x_k - m_k| > tS_k \end{cases}$$

What this essentially says, is that the central window element x_k will be replaced if the magnitude of the difference between x_k and the window median m_k is greater than some scale times the standard deviation of the window. The paper then continues into more detail about the hampel filter such as when, using the hampel filter had no advantage over the standard median filter in the case of an *implosion sequence* which is defined as a sequence in which more than 50% of the values are the same, in this case $S_k = 0$ and the hampel filter is equivalent to the standard median filter. The paper also showed in general that larger the thresh-hold or value of t , the less aggressive the filtering will be. This is shown by observing the *root sequences* of the filter as t is varied. It can be show that as $t \rightarrow \infty$, the filter approaches that of an *identity filter*. Overall I felt that this paper helped a lot with my understanding of hampel filter, and I found it to be a very interesting read. It gave me a good indication of when the filter could possibly go wrong, but given the data I am using, it gave me confidence that the filter would perform well. I also got the idea while reading the paper, that I could filter the data with difference thresh-hold values to see what thresh-hold level gives me optimal aggression.

[13] This paper describes the Pan-Tompkins algorithm for QRS complex detection. I felt the obligation to read the paper in order to fully understand the pipeline which lead to the data I am working with, i.e. the pre-processing which was done before I obtained access to the data. I found the paper to be interesting but perhaps it was a mistake on my part in choosing to read the exact paper from 1985. The reason I say this, is because a lot of the paper focuses on their implementation of the algorithm on a microprocessor from the time and they also discuss some of the sacrifices they made in terms of filtering the data in order to make the algorithm implementable on the hardware of the data. While, I did learn a lot about the process of the algorithm, that being, bandpass filtering, derivative

calculation, squaring, moving window integration and the determination of the QRS time via thresh-holds, in the end I found myself trying to filter out a lot of the information which was no longer relevant today. I feel as though it would be better to read a more modern description of the algorithm.

Progress

- I read [17] before my meeting with Dr. Lightbody, which was quite a productive meeting in which we discussed the results of the hampel filtering. During that meeting Dr. Lightbody stressed the importance of experimenting with the project, as at the end of the day, this is an experiment. We also discussed the possibility of running the data through a low-pass FIR filter to really try and isolate the low-frequency components of interest in the signal, it would be interesting to see if the FIR filtered data improved performance greatly or had no effect. During the meeting we also discussed the deep learning side of the project at a high level. When creating the training set, the data will be overlapped by about 80% in order to create more data artificially, and I also mentioned the possibility of using a facial identification CNN model as during my placement, I had watched a video on how the model was structured and I believe it was designed for extracting frequency domain features which would be very beneficial and I will need to look into that.
- On Monday after reading [17], I start to look into how I was actually going to implement the hampel filtering on my data. As I written everything in python thus far, that was my go to choice but after looking for any hampel functions in libraries, I was sourly disappointed to find only one `hampel` library on PyPI, the Python Package Index. Unfortunately, after looking at the code, I did not feel comfortable using it and knew if I had to do it in python, I would have to modify/contribute the library myself, to get what I wanted. Although Matlab also implemented a hampel filtering function by default, and after testing it out I decided it would be easiest to go with that. The only annoying part of this was that I had to rewrite a code function on parsing the CSV directory structure which I created in python. This was a pain but I eventually got it done and working. I also wrote a test program which randomly choice a subject and and a section which that subject had, and plotted the hampel input and output over one-another so that I could view the affect.
- After writing the MATLAB code for doing the hampel filtering, I realised that as the project continues the structure of my directories was becoming more complicated and I would need to come up with a better way to manage this. After some thinking I wrote a new python module called `paths.py` which contains a class which will manage the paths used. When an instance of this class is instantiated, the paths are automatically calculated and they are not relative paths, they are full paths calculated through the use of the OS environment variables. I need to update all of my current code to implement this. This allows me to change the structure of my file system in the future from one dedicated module and allow the change to be reflected everywhere. Parameters can also be passed to the class such as the value of K and N for the hampel filtering which will be reflected in the class, after the instance is created these parameters can also be changed and the change will be reflected in the class attributes automatically.

Plan

- This week I need to standardise the code for interpolating and resampling the hampel filtered data. I have already written code to do this but it must be implemented properly.
- I need to read another paper more focused on deep learning and also another paper depending on what I am doing during the week.
- I need to design a low-pass FIR filter to band-limit the signal.
- I also need to write up some sections for the report.

Week 6

Progress

- This week I was quite busy and did not get enough reading done. I was able to read the abstract and introduction of one paper but will finish it next week and review it then.
- Of the work that I did this week it was mostly based on writing the resampling code. This was more complicated than I originally thought. My original implementation of the resampling code worked fine but it was designed for one file. This implementation must resample on the CSV files containing the split, and hampel filtered data. This is where all of the data loss will occur. There is no point in interpolating data which will not contain enough information to form a training epoch. Meaning if there is not enough data in the CSV file to occupy a five minute interval this data should be dropped. My initial implementation which counted and logged which data must be dropped estimated a lost of 17.8 % of the total data. This is quite high. I spent Monday and Tuesday crawling through the code looking for mistakes. I did find some small errors but nothing significant. In my search for errors I began to develop a tunnel-vision which I know will not lead to any progress so I decided to leave the problem to my subconscious so a while. On Monday I will look through everything again before my meeting with Dr. Lightbody and will discuss the implications then. Although Eimear Shortiss did use this data last year so I know it should be fine.
- I wrote a new class which inherits from the original Parse class and performs the interpolation of the data using a cubic-spline. I believe I found the correct way to organise my software from here out. I will attempt to create a class which inherits from the necessary parent classes when required and performs operations on data in an object-oriented fashion. I believe this methodology will create re-usable maintainable and easily configurable software which if down the line of this project, I need to test something new and alter the pre-processing this should be absolutely possible.
- I forked the GitHub repository which Dr. O'Toole gave me access to use to generate the time-frequency distribution of my data.

Plan

- Read both the papers to completion which Dr. Lightbody sent.
- Clone the repository for generate the time-frequency distribution and look through the code, consider porting over the MATLAB implementation to Python, this will require a look at the MATLAB functions taken advantage of and ensuring that stable pythonic implementation exist.
- I have upgraded to a better laptop and the new one will arrive on Monday next week, this laptop runs Windows 11 and I will need to test my software out and ensure it functions correctly, this shouldn't be a problem for the python code but from my own research, Windows 11 isn't currently supported by MathWorks although several users report that MATLAB functions will small errors for Simulink.
- I have withdrawn a book from the library on Time-Frequency Analysis and I need to read some of the relevant chapters, in order to use and understand this algorithm from Dr.O'Toole, I will require some research in the area.

Week 7

Readings

- S. A. Raurale, G. B. Boylan, S. R. Mathieson, W. P. Marnane, G. Lightbody, and J. M. O'Toole, "Grading hypoxic-ischemic encephalopathy in neonatal eeg with convolutional neural networks and quadratic time-frequency distributions," vol. 18, no. 046007, 2021.
- M. Mesbah, B. Boashash, M. Balakrishnan, and P. B. Coldiz, "Heart rate variability time-frequency analysis for newborn seizure detection," in *Advanced Biosignal Processing*, A. Naït-Ali, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 95–121, ISBN: 978-3-540-89506-0. DOI: 10.1007/978-3-540-89506-0_5. [Online]. Available: https://doi.org/10.1007/978-3-540-89506-0_5.

Paper Summaries

[5] I found this paper to be quite interesting. I focused mostly on the "Methods" section of the article as this pertained mostly to my project and was the section of the article most understandable to me. For example, a lot of the terms in the "Results" section I do not know or understand, I must become familiar with these when I begin the actual training and testing of the CNN but as it stands now I am more focused on the pre-processing. From reading this paper I found several things which could be useful to me especially with regards to the time-frequency techniques used. The time-frequency techniques used are quite complicated and I need to do more study in the area to gain a better understanding but there were several practical use descriptions of the TFD analysis section in this article. One particular section being on the use of a pre-whitening filter which was used to flatten the spectrum of the signal and to emphasise the higher frequency components of the EEG signal. My immediate thought was could this be possible to use for the HRV signal to emphasise the lower frequency components more which would essentially point the CNN training algorithms in the direction which they should focus. This technique required the use of the finite difference method to approximate the derivative of the signal and referenced a paper which uses the technique which I will need to look into. They also mentioned how the low- and high- frequency components of the signal were removed before the TFD analysis in order to avoid "wrap-around" effects. Unfortunately this is not possible for the HRV signal as it is exactly the low-frequency components between 0-2 Hz which are of interest. Perhaps an experiment will be required to observe this effect in action so that this can be identified in practise. In this particular article the log of the absolute TFD matrix was used as the input to the CNN, there is no particular reason given but the reason will need to be determined and perhaps a similar approach required in my project. A zero-phase FIR filter was used to band-limit the signal which is interesting and I will need to research zero-phase FIR filters as I have been considering how the phase response of the filter would affect my signal considering that it is a time-series and I do not want to shift around the frequencies of the signal in time as that could cause undesired effects. The CNN model used in this article is described in detail which was quite interesting to see. I do not understand some of the operations used in detail and will need to research them. I also do not understand why this particular model structure was used and how it is optimal. I will need to research this area of work further when it comes to choosing a model for the project, but it is useful to have a model which although uses a different signal could be useful and would perhaps be beneficial to even use this model for the project considering it already operates on time-frequency data. I will need to discuss this further with Dr. Lightbody as per our previous discussions, my initial thought was to find a model such as a cats vs dogs or facial identification model which operates on image data and to retrain the model and perhaps modify certain aspects, but considering there is already a model that performs on the exact time-frequency structure is there any point in trying to fit another model to this task or would it be possible to train on two separate models in order to compare ?

[31] This paper was quite complicated to read through and I felt that I could not get an understanding of the technical content in the paper for which I came to it for. For example the paper skims over the description of the Wigner-Ville distribution, while I do feel that it gave me a better understanding of the purpose of the Wigner-Ville distribution and how it fit into the class of quadratic time-frequency domain techniques, the actual distribution itself was brushed over and a very complicated picture was given which I do not fully understand. Although there was a great section explaining the need for the time-frequency analysis, especially when used for the analysis of non-stationary physiological signals. They gave a very good example by looking at a chirp signal of linear increasing and decrementing frequency and calculating its fourier transform followed by a time-frequency representation of the signal. This example was great in showing how the fourier transform itself shows how the signal energy is distributed in the frequency spectrum with no time resolution for the signal as the Fourier transform is taken over the entire lifespan of the signal and is essentially averaging. Whereas the time-frequency distribution showed a clear picture how the frequency was changing with time. This is great as it is a fantastic example which I can recreate for me report to emphasise the importance. Overall for this paper (which I did not finish to completion yet) I found the content was simply going over my head and I need to study some more fundamentals before coming back with an understanding which will allow me to properly interpret the information. I did find a good reference through this paper, which is a comprehensive reference to time-frequency analysis written by the author of this particular paper which I will begin to study to ensure that my understanding is correct. I will need to work from the ground up and study the short-time fourier transform and then gradually move up to the more complicated techniques. Although, I will have to be careful and spent an appropriate amount of time on this study and not get too bogged down.

Progress

- During my meeting with Dr. Lightbody this week, I gained a much better understand of the data loss. After a preliminary calculation it was determined that with the default (and best) configuration of the NaN threshold and length of training epoch (that being 5 minutes) the loss of data was 17.8% of the total data. Throughout last week I had the false perception that this loss was too high and need to find a way to reduce it, but during the meeting Dr. Lightbody reassured me that this result was not out of the ordinary and was perfectly fine. During the meeting we also discussed how much time-frequency analysis which I should study into as I find the topic very interesting and will need to restrain myself from going down a rabbit hole. One of the other productive outputs of the meeting was a discussion of how to introduce the time-frequency analysis section from the view of the report. It is best to introduce the minimum theory required and then give suitable examples which display the technique and show why it is necessary. This point in particular is important for all sections of the report and it is necessary to learn this earlier on so that the plots and relevant examples can be generated as that cannot be completed last minute.
- I was able to concentrate on finishing the resampling code in its entirety throughout the week. Although this did take some time as I encountered some problems. One problem was caused due to the python "os" library function `os.listdir()` which creates a list of all the items in the current directory, analogous to the unix command `ls`. The code which I had written made the assumption that this method would create a list of the contents in the same order as in the file explorer or when displayed in the console, unfortunately this was not the case and caused problems which trying to write the resampled data to disk. There was also a problem with regards to the string processing required when renaming resampled files and some other problems which regards to how the underlying data was represented in python via the dictionary data structure which it made more sense to include a `Data` class which stored more information about the data other than just its location in the dictionary hierarchy. The object created from this class would then be stored in this dictionary structure.
- I found a book on time-frequency analysis in the library which I need to study soon. I read a section of the "Uncertainty Principle" chapter which explained the uncertainty principle as it was applied to signal processing and Fourier Analysis. I found this book to be very good and thorough although quite technical. For example the section which I read I found to be quite understandable conceptually and intuitively although I could not properly follow the mathematical proof which led to the conclusion as it required some more study of an earlier chapter which is quite technical.
- I began to play with the time-frequency analysis coded provided by Dr. John O'Toole throughout the week also. I created simple signals and passed it through the algorithm using a kernel and configuration which I do not understand but was used in the README as an example. I was able to interpret the results and they made sense. I gradually increased the complexity of the signals passed in. For example, using a monofrequent sinusoid, following by a chirp signal and then following by their sum. as can be seen below,

I also used the algorithm on one section of the actual resampled HRV data but the result made no sense to me.
- I began the Hampel filtering section of the report. I wrote an introductory section which covered some of the theory starting with a median filter and then giving an example of its application using a sinusoid with some outliers. I then intend on moving onto the hampel filter and giving another example of its use on a simple signal before showing the result of applying the filtering technique to the actual HRV data.

Plan

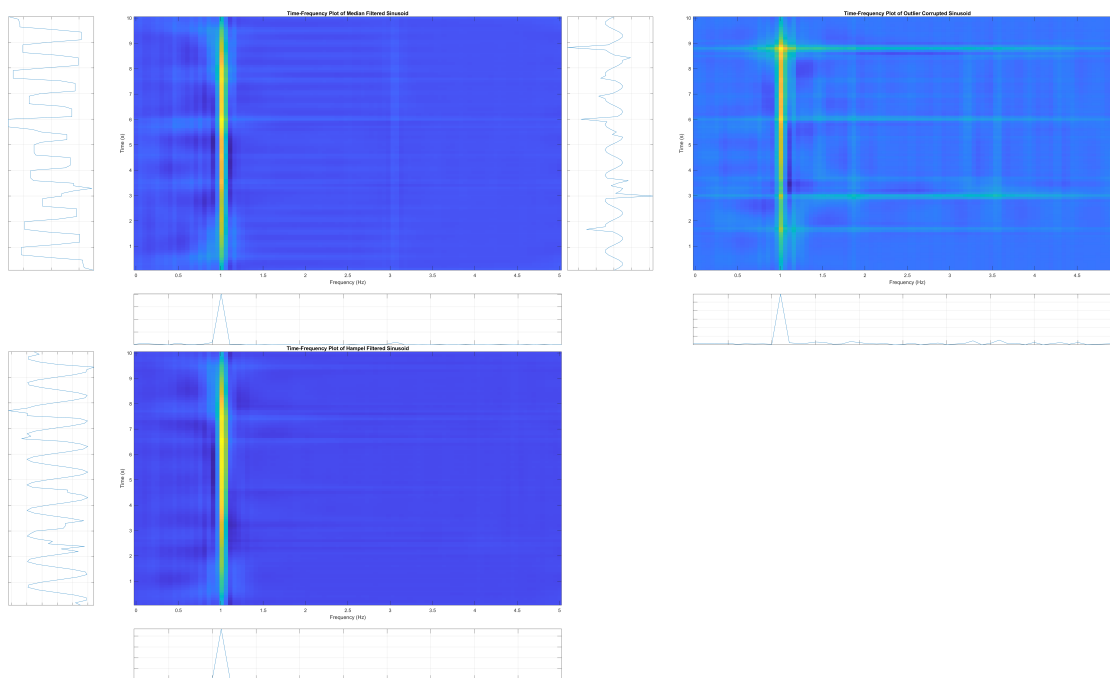
- Finish the hampel section of the report.
- Study up on the necessary time-frequency analysis.
- Look into porting the TFA m-file library to python.
- Play with the library more and get an understanding of the effects of different kernels and what might suit best.
- Design an simple FIR filter for the data.

- Study into autoregression.

Week 8

Progress

- This week most of my effort went into solving a problem I was having generating the time-frequency plot of data for examples in my report. Initially I had problems with the outlier injection into the sinusoids. The idea behind this example is to show the effect of the high-frequency outliers on the time-frequency plot and then to demonstrate the effectiveness of the hamper filter in removing the noise. I had to increase the amplitude of the outliers in order to generate a good example but eventually got this done as is shown below,



- The other example I was trying to create was an example of the time-frequency plot of a section of HRV data, before and after the hamper filtering. My first attempt of performing the time-frequency plot on pre-hamper filtered data was a failure due to the fact that that particular data set had not been resampled (only the post-hamper filtered data had been resampled). I then wrote a small amount of python code to resample the data, which worked fortunately due to the way in which I set up the code. After resampling there was essentially no plot to see after the algorithm was applied. I could not figure out what the problem was. Although after speaking to Dr. Lightbody, he gave me the idea of normalising the data. After normalisation I saw better results but was still relatively bad. I had been applying the algorithm to a long section of data, approximately 20 minutes in duration, but the FFT was dominated by a huge DC component which washed away all of the other frequencies. I then performed the analysis on a 5 minute segment which is what is to be used for the training and testing epochs, and this gave a great result which showed what I was looking for.
- I had a productive meeting with Dr. Lightbody this week where we discussed what the project expectations were for this semester and we were both on the same page of hopefully producing TFD of the data by the end of the semester. We also discussed possibly using the EEG CNN architecture for the project if possible which I will need to look into if the image sizes line up correctly which would be fantastic luck if they did and I could start with a proven network and try to improve upon it. I will need to look into the *receptive field* of CNNs.
- Unfortunately the only work I really did this week was towards my dissertation. Realistically a lot of my work during this last month will be focused on my interim report but I intend to have good TFD generated by Christmas time.

Plan

- Write a NaN and pre-processing section of the interim report.
- Study up on the necessary time-frequency analysis.
- Look into porting the TFA m-file library to python.
- Play with the library more and get an understanding of the effects of different kernels and what might suit best.
- Design an simple FIR filter for the data.
- Study into autoregression.
- Study CNN receptive field and other CNN concepts.

Week 9

Readings

- B. Boashash, "Chapter 1 - time-frequency and instantaneous frequency concepts," in *Time-Frequency Signal Analysis and Processing (Second Edition)*, Second Edition, Oxford: Academic Press, 2016, pp. 31–63, ISBN: 978-0-12-398499-9. DOI: <https://doi.org/10.1016/B978-0-12-398499-9.00001-7>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780123984999000017>.

Paper Summaries

[32] I found this chapter to be very useful. There are still some sections which I must revise over and cover but overall I found the chapter to be very digestible and understandable. I believe the section which describes what the *analytic associate* of a function is was extremely well written. The analytic associate is essentially a version of a signal whose frequency domain contains no negative frequency components. Mathematically, given $z(t)$ the analytic associate of $s(t)$,

$$Z(f) = 0 \quad \forall f < 0$$

The analytic associate can be obtained via the following operation,

$$z(t) = s(t) + jy(t)$$

where,

$$y(t) = \mathcal{F}^{-1}\{(-j\text{sgn}(f))\mathcal{F}\{s(t)\}\}$$

This is also known as the Hilbert transform a signal,

$$\mathcal{H}\{s(t)\} = s(t) * \frac{1}{\pi t}$$

Thus,

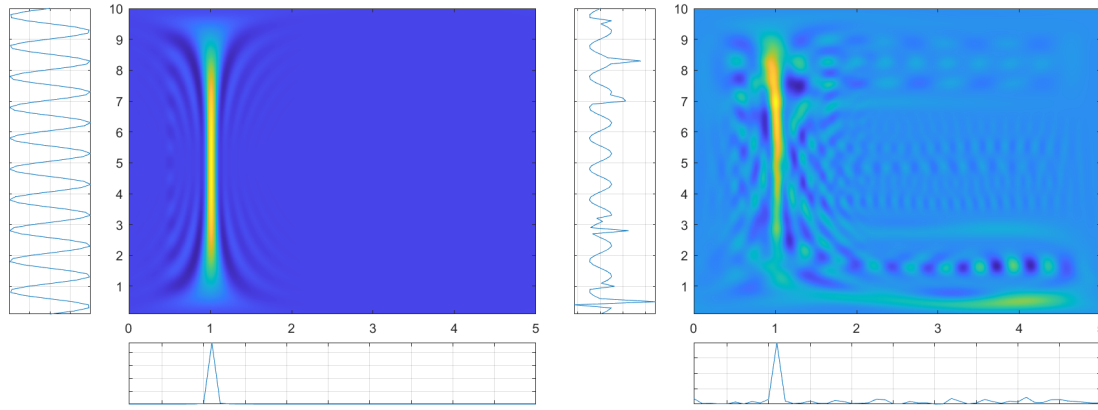
$$z(t) = s(t) + j\mathcal{H}\{s(t)\}$$

This explanation was very much welcomed as during my reading of other papers I had seen the term "analytic associate" mentioned but with definition and could not find an answer anywhere. The chapter also defines several signal properties such as the effective bandwidth,

$$B_e^2 = \frac{1}{E_s} \int_{-\infty}^{+\infty} f^2 |S(f)|^2 df$$

where $E_s = \int_{-\infty}^{+\infty} |S(f)|^2 df$ and the effective duration,

$$T_e^2 = \frac{1}{E_s} \int_{-\infty}^{+\infty} t^2 |S(t)|^2 dt$$



These definitions made sense to me although they were slightly conflicting with other definitions I had found in [33]. Looking at the above definitions it is clear that they both represent standard deviation about zero. In the above definitions there is a division operation by the total signal energy. Looking at the calculation from the perspective of a random variable and a probability density function, this division operation makes sense to me as it normalizes the $|S(f)|^2$ term which acts in place of the probability density function. This division ensures that the following probability density function property is upheld,

$$\int_{-\infty}^{\infty} p(x)dx = 1$$

But in [33] the total signal energy is assumed to be one and this division factor is left out which I found to be particularly confusing due to the conflict. I find [32] to be more thorough in the notation aspect although one of my other critiques would be that the use frequency and not angular frequency is used throughout the text. I still have to finish this section in which more definitions are defined although I am beginning to get slightly confused with certain definitions.

Progress

- While generating TFD plots I found a bug in my code which would have been a major problem down the road. I looked in the folder containing "subject9"'s data and discovered that there was no data to be found. After looking at the data before being interpolated I noted that the should absolutely be data in this folder. I re-ran the code to generate the CSV files for this section and stepped through with a debugger to discover that I misplaced a line of code which resets a boolean variable. The result was that if a subject contained one section of data which was not long enough for an epoch, then all of the subjects data after that particular epoch would not be written into the folder. The bugfix was quite simple, all that was required moving the line that resets the boolean variable to a more logical area. Fortunately I found the bug in the manner in which I did, otherwise this would have cause strange errors that would have been hard to debug.
- I worked on changing the parameters of the TFA algorithm in order to see how this affected the plots as a whole. I tested using separable kernels with a Hann window. The image size could also be chosen with this particular setup which was very interesting. An example of the results can be seen below.
Above is a pure sinusoid in which the plot looks almost like a contour map although it is hard to tell if this is in fact more accurate. Finally a plot of a sinusoid with outlier corruption is shown, which has a very interesting response which again similar to above, almost looks like a contour map.
- This week I also finished up my section for the interim report on NaN processing.

Plan

- I need to continue my work on the TFA study, getting a better understanding of the mathematics and also of what particular choices I need to make and can alter with the current TFA algorithm which I am using.

- I need to write a section on resampling for the interim report and get my introduction finished so that I can do the time-frequency analysis section, and then finally the plan for next semester and also the miscellaneous sections like risk analysis and ethics.
- If I have time do some research into the cubic-spline calculation which could be useful to mention in the report.

Week 10

Readings

- B. Boashash, "Chapter 2 - heuristic formulation of time-frequency distributions.," in *Time-Frequency Signal Analysis and Processing (Second Edition)*, Second Edition, Oxford: Academic Press, 2016, pp. 65–102, ISBN: 978-0-12-398499-9. DOI: <https://doi.org/10.1016/B978-0-12-398499-9.00002-9>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780123984999000029>.

Paper Summaries

[18] I found this to be very informative. I was able to get a decent understanding of TFA as a whole and also the specifics of the Wigner-Ville distribution. Although I do not have an intuitive understanding of how the transform works, I do understand the calculations, derivations and the properties. From my reading, the Wigner-Ville distribution appears to be the Fourier transform of the instantaneous autocorrelation function with respect to the lag. This instantaneous autocorrelation function at a particular time, t , is a measure of the correlation between both sides of the signal hinged upon the time, t . This makes some intuitive sense to me but I still do not fully understand it. This chapter mainly helped me with my understanding of the terminology and how the distribution was actually being calculated. I now understand about the kernels at play and what they are actually doing. I also re-read the section on TFA in [5] which was extremely helpful to see what kind of kernels they used in practice and how they tuned the window lengths etc. There is a section on the discrete wigner-ville distribution which I also skimmed over but it also very complicated and although I would be interested in learning about it, I feel as though my time would be better spent looking towards the deep learning aspect of the project.

Progress

- I met with Dr. Lightbody today and had a productive conversation in which we discussed the lack of requirement of filtering the HRV signal before downsampling due to the fact that the signal is artificially generated using the cubic spline. I also learned of the fact that I need to produce a Gantt chart which will plan the work which I require myself to complete during the second semester and also specify a backup plan in case certain things fail.
- I wrote up a section on resampling the HRV data.
- I started the introduction and have been gathering all the information which I have to write up a section on the background material. I have been finding this to be slow work as I feel the responsibility to be very careful with what I say considering a lot of the information is based in the medical domain which I am not particularly familiar with and could make some errors if I am not careful.
- I studied further into time-frequency analysis. I still am struggling to fully understand the mathematics behind the analysis but I do have an intuitive understanding of how it works and all of the terms related to the analysis and what they mean. For example, my study this week gave me an understanding of the terms "doppler" and "lag" which until now I did not fully understand. I also got an understanding for what a kernel is with regards to time-frequency analysis and the different types of kernels such as separable, non-separable, doppler-independent and lag-independent. I also now understand the relationship between the Wigner-Ville distribution and some of the other time-frequency analysis techniques such as the short-time fourier transform which I already have an intuitive and mathematical understanding of. Interestingly the

Wigner-Ville distribution is a general version of these transforms, the other transforms can be represented as a Wigner-Ville distribution with the instantaneous autocorrelation function being convolved or "smoothed" with some kernel which is derived from the original transform. For example, the Wigner-Ville distribution can be represented as the following,

$$\rho_z(t, f) = \mathcal{F}\{K_z(t, \tau)\}$$

The general time-frequency transform can then be represented as,

$$\rho_z(t, f) = \mathcal{F}\{R_z(t, \tau)\}$$

The STFT can then be represented as the following,

$$\rho_z(t, f) = \mathcal{F}\{G_{STFT}(t, \tau) \underset{t}{*} K_z(t, \tau)\}$$

where,

$$G_{STFT}(t, \tau) = w^*(t + \frac{\tau}{2})w(t - \frac{\tau}{2})$$

in which w represents the window used with the STFT. Thus, the kernel in this case is the instantaneous autocorrelation of the window.

- I did some testing with the TFA algorithm which I have access too. Initially I began reading through the code to get an overview of how it works. It is quite complicated but I was able to get better understanding of what was happening under the hood. After this I then began to run the analysis using different kernels and changing the kernel parameters to see how this affected the input. I put together a plan in which I will generate the TFA plot for a linear FM signal, which are generally used to test these techniques, using the Wigner-Ville distribution without any kernel or equally a kernel of $\delta(t)$. I will then use other kernel types and view the changes that they cause and create a library of plots for reference. I can also use this library to choose certain examples which best highlight the effects of the kernels.

Week 11

Progress

- During my meeting with Dr. Lightbody I showed him plots of a the time-frequency distribution of a linear FM signal which different length windows of Hanning smoothing kernel. I also showed the time-frequency distribution but with the absolute values and the log of the distribution. Essentially this is the TFD but in decibel form. This version of the distribution was much more interesting as there was much more to see with harmonic appearing in the upper frequency range. I also discussed my intentions for the rest of the semester, those being to start work on the machine learning side of things.
- I wrote a python function which will create my desired epochs for the machine learning aspect of the project. The function takes in the required length of the epoch in seconds and the percentage overlap of the epochs and then returns a list of lists containing all of the epochs.
- I tested the matlab library for python which allowed me to run some simple test scripts in matlab from python. This means that I can begin to write some scripts in matlab which will automate certain matlab specific tasks such as the hampel filtering and the TFA algorithms and can then implement these into my python machine learning pipeline.

Week 12

Progress

- Worked on the interim report for the full week.

Semester 2

Week 1

Readings

- C. C. Aggarwal, "An introduction to neural networks," in *Neural Networks and Deep Learning*. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer International Publishing, 2018, pp. 1–51, ISBN: 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0. [Online]. Available: <https://doi.org/10.1007/978-3-319-94463-0>.

Paper Summaries

[42] After looking through some of the deep learning reading material which I had amalgamated, I wasn't particularly satisfied with the material covered. I wanted to get a book which was good fundamental reference that analyzed networks from a fundamental perspective. Upon looking online I found that Aggarwal's book was consistently mentioned. After looking at a PDF, I decided to purchase the book as a reference. So far I have been studying the introductory chapter and leaving no stone unturned. I have studied the single layer perceptron in detail and feel as though I have a working understanding of how it all fits together. During the christmas break is when I got started on the book, but this week I gradually moved through the material further. I was able to understand a proof which proved that given linear separable data, the perceptron criterion will also converge. I also was able to understand how the single layer perceptron can be equivalent to a support vector machine, in the case that the hinge-loss function is used. For the generic perceptron criterion, the loss is for a single feature is,

$$L_i = \max\{-y_i(\bar{W} \cdot \bar{X}_i), 0\}$$

which is equivalent to the loss of a minibatch or several features if expressed as follows,

$$L^{perc}(\bar{W}) = - \sum_{i \in \mathcal{M}} \bar{W} \cdot \bar{X}_i y_i$$

where the set \mathcal{M} represents the set of misclassified feature vectors. Whereas the hinge loss can be expressed as,

$$L_i^{hinge} = \max\{1 - y_i(\bar{W} \cdot \bar{X}_i), 0\}$$

After analysing both expressions, it becomes clear that the hinge loss expression has a more stringent requirement for a feature vector to be classified as lossless. Not only must the model output the correct sign, but magnitude of the output of the model which could be perceived to be the degree of "correctness" must greater than one. Meaning that the notion of margin is introduced. This can be interpreted physically if a two dimensional hyperplane is imagined. When the decision boundary is visualised which can be done by plotting the linear hyperplane equated to zero, the hinge loss function will ensure that hyperplane is oriented such no feature vector will occupy the space between the two hyper planes generated by setting the original hyperplane equal to both positive and negative one. I must also admit that several of expressions in this chapter did confuse me at first. It was only when I read a section from [43] which was recommended in the chapter helped to clear that up. In that section a great breakdown of the perceptron is done along which a proof that given linearly separable data, the perceptron criterion which is used to find the optimal hyperplane and can be expressed as the following for optimization using minibatches,

$$\bar{W} \leftarrow \bar{W} + \alpha \sum_{(\bar{X}, y) \in \mathcal{M}} y \bar{X}$$

Which can be derived from the more general stochastic gradient decent method which can be expressed as,

$$\bar{W} \leftarrow \bar{W} - \alpha \nabla_{\bar{W}} L^{perc}(\bar{W})$$

Progress

- During the christmas break I was able to clean up all of my code and reorganise everything into packages and generally clean up all of the structure.
- I was also able to get the matlab-python interface to work correctly which allowed me to rewrite all of the matlab which I had previously written and package it into a function which took all the require parameters such as file/directory paths, input data etc. I then added all of these matlab functions to matlab's search path have them all stored in separate directories which clearly labels them as either "mfiles" or "mfuncs". I also wrote the python functions which initiated the matlab engine and the called the functions meaning that I should not have to deal with too much matlab anymore.
- On top of organising the matlab code, this finally gave me the opportunity to build a complete preprocessing pipeline which could take the NaN processed data, and perform the hampel filtering, resampling, and apply the tfd algorithm. This required that an epoch generation function be created also which would generate the required length epochs with the requested percentage overlap. The resultant pipeline system was built which takes in the requested parameters such as hampel filtering cofiguration, tfd windows and length, image size etc and produces a directory filled with images for each subject representing the dataset which has been fully preprocessed.
- I also did my interim presentation this week, it went quite well.
- I took the faults in my reporting system for the previous system and created a new and better method of writing up both my dissertation and logbook for the semester. I have re-used the same titlepage and other meta things like the summany, appendix structe, nomenclature and essentially all working components from the previous semester. I fixed some formatting issue like using roman numeral page numbering for the introductory pages. I also downloaded the IEEE conference paper template and have done some preliminary work to set it up to write my conference paper.
- I also did several PyTorch tutorials on getting setup and have installed and tested with PyTorch. I was even able to write a test class which can parse the image data which I have stored in csv files. This is good although after meeting with Dr. Lightbody , I have realised that I need to account for how I am going to split the test/training data etc which is a problem I can perhaps kick down the road and focus more on building a system which will train the network before building another system around this with organises how the data is fed in.
- I also updated the repository on my local machine at home aka my desktop PC. The graphics card which I have on this PC is a NVIDEA GTX 1650 which is CUDA enable meaning that I can train the model and take advantage of the GPU. After updating the repository, I ran the pipeline to ensure its functionality which required a lot of tweaking such as installing a different matlab version etc. This now works meaning that I simply need to keep this local repository up-to-date and ensure everything works which will eventually allow me to train the model on this machine.
- An annotations file was also generated which contains the subject specific grading information. Two files were generated, one with the original classifications and another with a modified binary classification system based on whether of not to treat the patient. The classifications were mapped as follows,

$$\{x \in \mathbb{Z} \mid 0 \leq x \leq 4\} \rightarrow \{+1, -1\}$$

according to the following criteria,

$$\begin{cases} x = +1 & 2 \leq x \leq 4 \\ x = -1 & 0 \leq x \leq 1 \end{cases}$$

Plan

- Work on some PyTorch tutorials, maybe build a system which trains using an example dataset to get a feel for the framework and have built one working pipeline.

- Write the background material for my introduction, taking into account the critiques made by Dr. Lightbody for the interim report. Also think about how to write this for the conference paper.
- Study some more deep learning material such as backpropagation and write up a section on it for the dissertation. Look into the "ADAM" optimization algorithm.
- Get the data in a better format such as unpacking it from subject specific images and think about how to track the grade of a particular image after unpacking and splitting data into test/train sets.

Week 2

Readings

- C. C. Aggarwal, "An introduction to neural networks," in *Neural Networks and Deep Learning*. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer International Publishing, 2018, pp. 1–51, ISBN: 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0. [Online]. Available: <https://doi.org/10.1007/978-3-319-94463-0>.

Paper Summaries

[42] This week I delved into the backpropagation algorithm and also learned the matrix notation for representing neural networks, along with learning a little about graph theory, particularly the terminology used and some definitions, specifically to understand the meaning behind the term "Directed Acyclical Graph" (DAG). I was actually remarkably surprised by how simple the backpropagation algorithm was. The purpose behind the algorithm, is to find the rate of change of the loss function with respect to each of the weights/biases in the system, specifically,

$$\frac{\partial L}{\partial w_i}$$

The neural network can be viewed simplified as a directed acyclic graph. This is a directed graph, meaning that edges have direction, with no directed cycles, meaning that there is no closed loop.

Progress

- I got a much better understanding of how the backpropagation algorithm functions.
- I setup a deep learning pipeline to allow me to train the FashionMNIST dataset using [44].
- I setup tensorboard which allows easy plotting of the training/testing loss curves along with other visualisation techniques with the help of [46].
- I setup my personal desktop PC for remote login through software called AnyDesk. I also setup my personal desktop with an installation of PyTorch which came with the pre-built CUDA runtime/compiler tools required to take advantage of GPU during training. I then modified the tutorial setup to use the GPU and noted an impressive increase in the speed of training.

Plan

- Create a subset of training and testing data from the current TFD images which I have and build a suitable schema in order to create a dataloader object for the training and testing data which includes the correct label.
- Look into the VGG16 and ResNet50 CNN models as candidates for the model intended to be used for this project. Interestingly, from preliminary reading, it appears as though models accept tensor inputs of the dimension, (3, 224, 224), which I was not expecting. Meaning they accept RGB images of dimensions 224×224 . Having already discussed how to handle the RGB problem with Dr. Lightbody, it appears as though the solution could be as simple as given each colour input the same image. The fact that the images are not of dimension 256×256 does puzzle me as I assumed this would be the case. Nonetheless, I believe

the workaround for this could be as simple as just generating images of these dimensions, given that using separable kernels allows for the modification of image size.

- Write the background material for my introduction, taking into account the critiques made by Dr. Lightbody for the interim report. Also think about how to write this for the conference paper.
- Make further progress in my study into deep learning.

Week 3

Progress

- I did a little more research in VGG16 and discussed model selection with Dr. Lightbody . He recommended VGG16 as it was not as deep as ResNet50 which means it will be easier to train and the depth is not necessary, especially with the amount of data available.
- I regenerated a dataset with the image size of 224×224 as required by VGG16 and other models. I discussed this aspect of with Dr. Lightbody and he was in agreement that this was a fine tradeoff to make given that this gives us access to a proven image processing model.
- I discussed what plots I should have available on the tensorboard with Dr. Lightbody and he recommended that I just stick with the training and testing loss for the moment and I could implement other plots later on in the development process when required. At this point I felt comfortable that I have all of the material necessary to create a "mini-dataset" which is a subset of the total dataset in order to train on and see if I could the expected training and testing loss curves. Thus, I wrote a function to randomly select a configurable number of subjects from the dataset and use their images in a configurable training and testing split.
- I then created a custom class which inherited from PyTorch's "Dataset" class. This allowed me to index the dataset properly. I stored the subject grade label in the file names for simplicity. This used the template from the pytorch testing I had done previously.
- Following this ability to create the Dataloader object, I essentially copied the code I had developed for the FashionMNIST dataset and modified to suit my application. This included the tensorboard plotting. Considering that I was not using the same model, I imported the VGG16 model, as can be done in PyTorch and used the most up-to-date pretrained weights. Given that the VGG16 model has 1000 output classes, I took a simple approach and removed the last fully-connected or linear layer and replaced with my own custom linear layer which only had two output classes. This was a very simple approach, a better approach may be to just add another fully connected layer to bring the 1000 classes down to 2.
- I then set the model to be trained on my personal PC's GPU although immediately ran into problems with memory. I was running out of VRAM immediately, from searching online I followed some advice in simply reducing the batch size until it would train properly, this led me to a batch size of 8. There is probably a better solution to this, perhaps I have some variables inside my training loop that are causing memory issues, this will be solved later, I just wanted to get the model training and have some results.
- Finally I set the model to train, I trained multiple models and saved their training and testing curves in the tensorboard "runs" directory. The models took quite some time depending on the learning rate, but on average it took between 6 – 8 hours. Fortunately, I was able to see the correct curves in the training and testing loss when a moving average "smoothing" filter was applied. The actual loss curves were quite noisy. Unfortunately the accuracy stayed stagnant throughout the training and testing process. The accuracy per epoch ranged between 60 – 80%. Having looked at the distribution of the mini-dataset upon its creation, I knew that the dataset contained approximately 70% "treat" subject or grades in the range of 2 – 4. This led me to believe that the models accuracy was simply choosing the treat option each time as this would explain the variability of the accuracy and its lack of improvement. Although, I was not particularly interested in the accuracy, more the loss curves which provided encouraging results.

Plan

- Now that the loss curves are proven to be positive, I will have to investigate the accuracy problem.

- The dataset should be expanded and perhaps different proportions of grades should be introduced to see the effect on the accuracy and loss.
- A better method of storing results and trained models needs to be created as they all cannot simply be dumped into a single directory as is the case now.
- The final layer of the model must be discussed with Dr. Lightbody and maybe a better layer created like a softmax layer.
- Need to comb through the code and clean it up, as it currently stands, the code is somewhat messy and it needs to be easily debuggable, perhaps the training and testing loop functions and the Dataset class should be added to the anser package and simply imported. This would make sense.
- Write the backround material for my introduction, taking into account the critiques made by Dr. Lightbody for the interim report. Also think about how to write this for the conference paper.
- I need to read some more papers as I have been lacking on that this week, particularly into the VGG16 model and some other models like it.

Week 4

Readings

- G. Boylan, L. Burgoyne, C. Moore, B. O’Flaherty, and J. Rennie, “An international survey of eeg use in the neonatal intensive care unit,” *Acta Paediatrica*, vol. 99, no. 8, pp. 1150–1155, 2010. DOI: <https://doi.org/10.1111/j.1651-2227.2010.01809.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1651-2227.2010.01809.x>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1651-2227.2010.01809.x>.
- E. Evans, J. Lerner, R. Sankar, and M. Garg, “Accuracy of amplitude integrated eeg in a neonatal cohort,” *Archives of disease in childhood. Fetal and neonatal edition*, vol. 95, DOI: <https://doi.org/10.1136/adc.2009.165969>.
- S. Karamian, A. G. and C. J. Wusthoff, “Current and future uses of continuous eeg in the nicu.,” *Frontiers in pediatrics*, vol. 9, 2021. DOI: <https://doi.org/10.3389/fped.2021.768670>.

Paper Summaries

[12] This paper described a compilation of several surveys which questioned clinicians (both neonatologists and neurophysiologists) about their confidence with regards to diagnosing via cEEG and aEEG. The paper was very interesting and it was great to be able to put numbers to it rather describe the ‘general feeling’.

[11] This paper looks at the the accuracy of amplitude integrated EEG (aEEG) in comparasion with conventional EEG (cEEG). The results were very interesting showing the failings of aEEG.

[9] This paper discusses the use of EEG in hospitals. I was looking at this paper from the aspect of the challenges to the use of EEG such as the expenses that come with EEG and also complications in terms of having the specialised staff to actually make use of the EEG data. This was an interesting paper as it gave substance to the previously unsubstantiated claims which I had made about the complexities of using EEG in hospitals today.

Progress

- Wrote a portion of the introduction for the conference paper
- Changed the VGG16 implementation by keeping the last layer to converge to 1000 different classes. Instead I added another fully connected layer which brought those classes down to 2. This improved performance.
- Cleaned up the code that performs the training. Perhaps it would still be better to move some of the static boiler plate code to the anser class. The naming convention for models was improved to include the model name, optimiser, loss function etc along with the date and time of creation to ensure no namespace pollution.

- Tested increasing the batch size for training again but to no avail. It appears as though the fundamental limitation on training is my current GPU. The GPU in use only has 4 GB of VRAM which from preliminary research suggests is on the low end. Perhaps it may be required to train on a UCC server to maximise performance.

Plan

- Discuss direction with Dr. Lightbody . At the moment a lot of progress has been made but I need to discuss the results and the nature of their production with Dr. Lightbody in order to find our next steps and find what methodologies are important.
- Move static boiler plate deep learning code to the answer class.
- Expand the current mini-dataset to something bigger maybe the full dataset.
- Look into setting up a 10-fold cross-validation.
- Finish conference paper introduction.
- Read a VGG16 paper.

Week 5

Progress

- Discussed training and testing curves with Dr. Lightbody . He made a good point about the amount of noise on the curves. Although on average the curves look positive, the amount of noise is worrying and I will need to try to understand why it is occurring. We also discussed the end product of this project, that being the 10-fold cross validation which must be done proving that there is merit to this approach. Dr. Lightbody mentioned that for this validation, the dataset will need to be subject independent, meaning that the same babies data cannot be used in both training and testing. Given our previous conversations I assumed this was not a requirement so I will have to consider how to go about doing this.
- During my discussion with Dr. Lightbody I mentioned my problem of not being able to increase my batch size beyond 4 images. This was a limitation on the VRAM capacity of my GPU. To counteract this, a new GPU would be needed or access to more powerful compute. The easiest option was to find a spare GPU for my personal PC as then there would be no real software issues. Dr. Lightbody was able to find a spare NVIDIA GeForce 1070, which I was able to install with little to no hassle. After installing, I let the model train overnight with the maximum batch size the GPU could handle, that being 30 images. I was also training on 90% of the available data, so I was hoping that the noise level would reduce. Unfortunately, the noise level appeared to get worse the more epochs I trained on. For 350 epochs the training took approximately 17 hours.
- This week as a whole was somewhat unproductive as I had quite a bit of work outside the project. I hope to be able to focus more next week on getting work done.
- I cleaned up the code quite a bit and made the actual implementation very concise and simple.
- Used the full dataset for training and validation.

Plan

- Play with the model, add an additional fully connected layer between the 1000 classes and the binary output, look into batch normalization and implement it.
- Read the VGG model paper and research noise in training curves.
- Study cross-validation and devise a method of implementing it with a subject independent dataset.

- Conference paper (for university marks) is due in Week 8 of this semester. This is two weeks from the time of writing, I need to finish the introduction to the conference paper and modify the material from the interim report to account for suggestions by Dr. Lightbody and add a results section.
- Save the model with lowest moving average test loss.

Week 6

Paper Summaries

<empty citation>

Progress

- In my discussion with Dr. Lightbody , we came to the conclusion that the network was definitely learning something but it needed improvements as the test loss curve was not pushing down enough. Dr. Lightbody suggested that I begin subject independent testing to remove any bias that occurs between the training and testing sets from baby's data which is shared between the two sets. He also suggested that I implement batch normalization which would greatly improve performance and also later down the line, learning rate scheduling. After doing some research, I unfortunately came to the conclusion that I had already been using a batch normalized version of VGG16 which meant I could not try out the new technique. Perhaps I should try and remove it to see the effect. I was also able to start training in a subject independent manner. To do this I had to be completely change how the data gets into the model. Originally, I chose to take a "generator"/lazy retrieval approach by reading the data from the file upon request, but throughout this implementation I had to change several things and I decided to read all of dataset in at once which I anticipated would improve training speeds provided that there was no memory problems by loading that amount of data in. Fortunately, no memory problems occurred, there is now quite a wait before training as all the data is read in and organised, but this drastically improved training speeds.
- I wrote a function that implements the logic of a K-fold cross-validation. The code for this was also used to implement the subject independent training logic as this can be viewed a single training/testing iteration in cross-validation.
- I began training with increased speed and changing certain configurations to see their effect. After implemented the subject independent scheme, the effect of bias was most definitely removed as a noticable step down in the accuracy was seen. This was a relief as I had a inkling that my previous were somewhat too good to be true.

Plan

- Discuss further steps with Dr. Lightbody .
- Write main body of conference paper.
- Identify hyperparameters and look into tuning.
- Think about the training implementation of K-folds cross-validation.
- Continue to change configuration and interpret results.

Week 7

Progress

- Improved code base significantly to allow the changing of the configuration for training without manually changing parameters in the code through the use of command line arguments.

- Build and improved the logging infrastructure to avail use of the python *with* keyword to ensure that the files are closed properly. Added a tag command line argument with allowed for personalized naming along with the already date and configuration based naming system.
- Implemented the ability to do cross-validation training. By default the system to divide the data into 10 folds and use each fold as a test set while training on all other folds, this is looped through for all the models and all of the data is saved in one directory.
- To get rid of my dependence on the tensorboard plots, I was able to write a plotting module which automatically interprets all of the data I have saved in csv format, plots the data along with formatting it and calculating the moving average for plotting purposes.
- Implemented the ability to apply custom transforms on the image data like scaled and normalizing the image data. Having training with these configurations, there did not appear to be a huge advantage other than using the log of the absolute value.
- Wrote up a decent amount of my conference paper.

Plan

- Make a presentation for my meeting with Dr. Lightbody as I felt having a powerpoint presentation with all of my plots displayed allowed for a more productive discussion.
- Discuss my results with Dr. Lightbody and also discuss my conference paper format and maybe show him some of my paper to get a comment and see where I can improve.
- Do more training with different configurations and come up with some ideas.

Week 8

Progress

- This week the conference paper is due and I have been struggling to get results which have variance. When I do the 10 fold cross-validation there always tends to be one fold which learns nothing. Fortunately, I decided to think about changing the architecture, I felt that I had done almost everything I could with the current VGG16 implementation and that perhaps it was just too big a model. Then I figured out that if I reduce the model size, I can increase the batch size which should allow batch-normalization to really kick into gear because the batch would be a good representation of the population. I decided to use VGG18 which only has about 2 million parameters in comparison to VGG16 which has 138 million. The reduction in parameters allowed me to crease my batch size to 256. I was able to get performance with the least variance observed to date which showed me that this was the way forward.
- After a discussion with Dr. Lightbody I learned how to display my results correctly. I need to run a 10 fold cross-validation to determine an early stopping point. Then re-run the 10 fold cross-validation with different randomized folds and stop at the early stopping point. My performance will be dictated by the performance at that point.
- I wrote my conference paper and achieved an AUC of approximately 0.78 at this early stopping point which is marginally better than previous systems in comparison to Andrew and Eimear but still below Rehan's system at an average of 0.82. I need to test different approachs such as a smaller batch size but ResNet50 and other transforms like using the the log of the image.

Plan

- Start writing dissertation, need to get my introduction written soon which should not be too difficult as I improved it for my conference paper and then shortened meaning I can essentially use the unshortened version.

- Test different approaches.
- Save a model or two to generate ROC curves.

Week 9

Progress

- This week I didn't get a huge amount done because I had to play catch up on some continuous assessment from other modules because I had put all my effort into producing results for this project up until now.
- I used the DataAnalysis class I wrote to measure the dataset mean, maximum and minimum after applying the absolute logarithm transform. I was then able to write two new transforms which zero mean the absolute log of an image and convert the logged image to the RGB range.
- I have been testing the system but I am not sure how effective it will be.

Plan

- Start writing dissertation, need to get my introduction written soon which should not be too difficult as I improved it for my conference paper and then shortened meaning I can essentially use the unshortened version.
- Rigorously test the logarithm approach, try ResNet50 and maybe ResNet30.
- Save a model or two to generate ROC curves.

Week 10

Progress

- Worked on getting my results and preparing them to display to Dr. Lightbody for discussion in our meeting.

Plan

- Need to stop work on the project and begin fully dedicating my time to finishing the dissertation write up.

Week 11

Progress

- Worked on dissertation

Plan

- Keep working on dissertation

Week 12

Progress

- Worked on the dissertation and submitted it !

Plan

- Clean up the code in the repository and think about maybe porting some of the TFA and preprocessing algorithms to Rust during the summer and make this available as a python package.