



**QUEEN'S
UNIVERSITY
BELFAST**

MACHINE LEARNING FOR ELECTRODE MOTION REDUCTION IN ELECTROCARDIOGRAM SIGNALS

by

Ben Russell

Student Number: 40178580

Supervisor: Professor Seán Mcloone

A Master's Thesis

Submitted to the Department of Electronics, Electrical Engineering and
Computer Science

Queens University Belfast

In Fulfilment of the Requirements

For a Master's Degree

September 2024

Final Year Projects 2023-2024

Machine learning for electrode motion artefact removal in ECG signals

Supervisor: Prof. Seán McLoone

	Control	X	Embedded Systems		High Frequency Electronics		Microelectronics
	Electric Power	X	Software	X	Connected Health		MEMS
	Cyber-Security		Wireless Communications	X	Signal/Image Processing	X	Intelligent Systems
	Digital Design		Sensor Networks	X	Data Analytics	X	Electronics

The electrocardiogram (ECG) is a non-invasive method to measure the electrical activity of the heart and can be used to diagnose heart disease. According to the World Health Organisation (WHO), chronic heart disease was the number one cause of death from 2000 – 2019. Long term ECG monitoring is currently the gold standard for diagnosing cardiovascular diseases (CVDs), however obtaining reliable long-term measurements of the ECG signal is challenging because patients are required to collect their ECG signal remotely on a wearable device. Wearable devices are inherently contaminated with noise which can suppress the essential pathological biomarkers and, in some cases, render the ECG completely unusable. ECG signals can be contaminated by many types of noise including: 1) Baseline Wander, 2) Powerline Interference, 3) Electromyographic and 4) Electrode Motion artefacts. Most of these noise sources can be reduced through the use of time and frequency domain digital filters. However, the frequency spectrum of electrode motion noise overlaps with the frequency spectrum of a typical ECG signal making it very difficult to remove in the time and frequency domain. The objective of this project is to explore if AI/ machine learning can be used to learn the characteristics of, and correct for, electrode motion induced noise on ECG signals.

Objectives

1. Conduct a literature review on ECG motion artefact removal algorithms to identify the different approaches that exist, and the challenges involved in developing effective methods and assessing their performance.
2. Using available online sources (e.g. Physionet¹) compile an ECG dataset which can be used to investigate and assess ECG motion artifact removal algorithms.
3. Develop a model for generating synthetic motion artefacts in clean ECG signals and use it to create a reference dataset for ground truth comparisons.
4. Investigate machine learning approaches to reducing electrode motion noise on ECG signals.
5. Develop and implement a candidate approach using Python or Matlab and validate its performance on the datasets from (2) and (3).
6. Compare the performance of the developed algorithm against alternative baseline algorithms from the literature.

MEng Extension

1. Explore advanced deep learning concepts (e.g., transfer learning, data augmentation) to enhance the performance of models and/or develop and evaluate alternative machine approaches for motion artefact removal.
2. Provide a rigorous assessment of all approaches developed with regard to real-time/embedded system implementation constraints.

Learning Outcomes

At the end of the project the student will be able to demonstrate:

1. A good understanding of ECG denoising algorithms.
2. A working knowledge of machine learning/ deep learning models and associated development tools
3. Enhanced programming skills in Python or Matlab, particularly with regard to algorithm development and signal processing.

Abstract

Electrode Motion is a non-linear, non-stationary noise source that can corrupt electrocardiogram signals to an extent where they become unreadable by both human experts, and automated algorithms. The reduction of these artefacts has proved to be challenging using standard denoising methods therefore, an alternative technique that can process this type of noise would be beneficial.

The aim of this thesis is to explore the application of artificial intelligence (AI) for the removal of electrode motion artefacts from electrocardiogram (ECG) signals. A comprehensive reference database was developed, consisting of both clean and noisy ECG signals. The clean ECG signals were generated using three ordinary differential equations (ODE's) to accurately model the heart's electrical activity in a lead II form factor. Noisy signals were synthesised by extracting noise from an online database and employing autoregressive (AR) modelling to expand the size and variability of the noise signals, while maintaining the same statistical properties. Each noise instance was then scaled to various Signal-to-Noise Ratio (SNR) levels and superimposed on the clean signals to create a large and diverse set of EM noise corrupted ECG signals.

The database of synthetic clean signals demonstrated similar statistical properties to that of the MIT-BIH Arrhythmia database, providing evidence that the synthetic generation of ECG signals is a valid method. Similarly, the expansive noise corrupt database inhibited similar time and frequency based statistical properties showing that the generation of a large and valid noise corrupt database is possible.

The performance of each algorithm was rigorously validated using both synthetic and real ECG databases. Key performance metrics such as Root Mean Squared Error (RMSE), Normalised Correlation Coefficient (NCC) and SNR improvement were employed to assess the efficacy of the noise removal techniques. Additionally, the results of the proposed AI-based methods were compared against traditional time-frequency based approaches to establish their relative advantages.

The comparison of various pre-existing Deep Neural Network (DNN) solutions using the training database was performed. Variants of Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Auto-Encoder (AE) Neural Networks were implemented based on previous research and trained on the enhanced database. Results show that Deep Neural Networks, specifically significantly outperform conventional filtering methods. Specifically, Convolutional Neural networks performed the best, achieving low RMSE values of 0.17mV, 0.14mV, 0.07mV, 0.11mV and 0.07mV across SNR levels of 0dB, 6dB, 12dB, 18dB and 24dB respectively. High cross correlation values of 0.48, 0.5, 0.82, 0.9, 0.92 were obtained and SNR improvements of 2.33dB, 2.59dB, 4.19dB, 5.32dB and 3.45dB were observed. This exceeded the performance of the best performing traditional filtering method by 21%, 8% and 17%.

This study demonstrates the potential of advanced AI models, specifically Deep Learning (DL) in improving the quality of ECG signal denoising by effectively mitigating the impact of electrode motion artefacts. The findings suggest that these AI techniques can offer significant improvements over conventional methods, paving the way for more accurate and reliable ECG analysis in clinical settings. Furthermore, since time-series noise is a common

problem across many engineering disciplines, the method may offer a solution in removing any non-linear and non-stationary noise observed.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Seán McLoone, for his continuous support, guidance, and encouragement throughout the development of this work. His expertise and insightful feedback were invaluable in shaping the direction of this work.

I would also like to extend my thanks to the Department of Electronics, Electrical Engineering, and Computer Science at Queen's University Belfast for providing the resources and opportunities to conduct this research.

I would also like to thank my colleagues at B-Secur for their support during the completion of this project.

A special mention goes to my friends and family for their unwavering support, patience, and encouragement during this journey.

Lastly, I would like to acknowledge all the authors and researchers whose work has contributed to this field. Without their dedication and innovations, this project would not have been possible.

List Of Figures

Figure Number	Title	Page
Figure 1	Typical ECG Signal Obtained from a Healthy Heartbeat	9
Figure 2	Noise vs Signal Frequency Overlap in ECG Data	10
Figure 3	Example of an ECG Signal Corrupted by Electrode Motion	10
Figure 4	Fourier Analysis of ECG Signal	14
Figure 5	Deep Neural Network Structure for ECG Denoising	18
Figure 6	Illustration of four primary activation functions used in DNNs	19
Figure 7	Illustration of convolutional neural network structure.	19
Figure 8	Illustration of convolution layer process in CNNs.	20
Figure 9	Illustration of max pooling aggregation function used in the pooling layer of a CNN.	21
Figure 10	Illustration of average pooling aggregation function used in the pooling of a CNN.	21
Figure 11	Typical structure of a recurrent neural network.	22
Figure 12	Illustration of network structure of an auto encoder.	23
Figure 13	Illustration of a sparse AE operation.	24
Figure 14	Training process of a DNN.	25
Figure 15	Individual neuron localisation for back propagation derivation.	26
Figure 16	Raw EM noise signal extracted from Physionet.	29
Figure 17	Effect of a bandpass filter on noise signal.	30
Figure 18	Depiction of Latin Hypercube sampling method.	31
Figure 19	Full fiducial point annotations on an ECG.	32
Figure 20	Plots showing 4 different clean signals for all heart rates.	33
Figure 21	Network architecture for region based CNN	36
Figure 22	Network architecture for convolutional denoising autoencoder	36
Figure 23	Network architecture for Deep Recurrent Neural network.	37

Figure 24	Overall framework concept for deep learning training and evaluation.	39
Figure 25	Denoised ECG signal at 24dB by RCNN.	44
Figure 26	Denoised ECG signal at 18dB by RCNN.	44
Figure 27	Denoised ECG signal at 12dB by RCNN.	44
Figure 28	Denoised ECG signal at 6dB by RCNN.	45
Figure 29	Denoised ECG signal at 0dB by RCNN.	45

List Of Tables

Table Number	Title	Page
Table 1	Description and impact of each noise source to an ECG signal.	13
Table 2	Waveform Limits used in LHS.	31
Table 3	Fiducial point limits used to validate clean signals.	33
Table 4	Performance of clean signal generation (Duration)	41
Table 5	Performance of clean signal generation (Amplitude)	41
Table 6	Comparison of statistical properties of generated database against MIT-BIH	41
Table 7	Comparison of statistical properties of noise signals in generated database vs MIT-BIH.	42
Table 8	Performance from each DNN	43
Table 9	Performance of traditional filters.	46

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
AR	Autoregressive
CNN	Convolutional Neural Network
DAE	Denoising Autoencoder
DFT	Discrete Fourier Transform
DL	Deep Learning
DNN	Deep Neural Network
ECG	Electrocardiogram
EM	Electrode Motion
FFT	Fast Fourier Transform
GRU	Gated Recurrent Unit
LHS	Latin Hypercube Sampling
LMS	Least Mean Squares
LSTM	Long Short-Term Memory
MA	Muscle Artefact
NCC	Normalised Cross Correlation
NSTDB	Noise Stress Test Database
ODE	Ordinary Differential Equation
PINN	Physics-Informed Neural Network
PL	Powerline Interference
PSD	Power Spectral Density
RNN	Recurrent Neural Network
RMSE	Root Mean Square Error
SNR	Signal To Noise Ration
SVM	Support Vector Machine
WT	Wavelet Transform

1 TABLE OF CONTENTS

2	<i>Introduction.....</i>	10
3	<i>Literature Review</i>	14
3.1	Digital Signal Processing in ECG Denoising	14
3.2	Electrode Motion reduction In ECG Signals.....	17
3.3	Intelligent Systems in Biomedical Signal Processing	17
3.3.1	Machine Learning	17
3.3.2	Deep Learning.....	18
4	<i>Methodology</i>	29
4.1	Database Generation.....	30
4.1.1	Noise Extraction	30
4.1.2	Clean Signal dataset Generation.....	31
4.2	Corrupting Clean ECGs with Noise.....	34
4.2.1	Noise Modelling	34
4.3	Deep Learning Models	37
4.3.1	Region Based Convolutional Neural Network.....	37
4.3.2	Convolutional Denoising Auto Encoder.....	37
4.3.3	Deep Recurrent Neural Network	38
4.3.4	Loss Function	38
4.4	Performance Metrics	38
5	<i>Implementation</i>	39
6	<i>Results and Discussion</i>	42
6.1	Database Performance	42
6.1.1	Clean Signal Feature Validation	42
6.1.2	Comparison With Standard ECG Database.	42
1.1.1	Noise Signal Validation	43
6.2	DNN Performance	44
6.3	Comparison to Traditional Filters.....	47
6.4	Analysis Of Results.....	47
6.5	Implications And Applications	48
7	<i>Conclusion</i>	48
8	<i>References.....</i>	50

2 INTRODUCTION

The Electrocardiogram (ECG) is a vital tool in modern medicine, offering a non-invasive and direct method for monitoring the electrical activity of the heart. By recording the heart's electrical signals through electrodes placed on the skin, the ECG provides essential insights into the rhythmic patterns and conditions affecting the heart's function. This capability makes the ECG indispensable for diagnosing various cardiac abnormalities, such as ischemic heart disease, myocardial infarction, arrhythmias and cardiomyopathy [1]

An ECG signal represents the sum of electrical potentials generated by the heart muscle during each cardiac cycle. The signal is characterised by a series of waves and complexes, most notably the P wave, QRS complex, and T wave, each corresponding to specific phases of the heart's electrical cycle. The P wave indicates atrial depolarization, the QRS complex represents ventricular depolarization, and the T wave is associated with ventricular repolarization. Analysing these components allows healthcare professionals to assess the timing of cardiac events, the presence of abnormal rhythms, and the health of the heart muscle. **Figure 1** shows an example of the time series electrical signal obtained from one heartbeat.

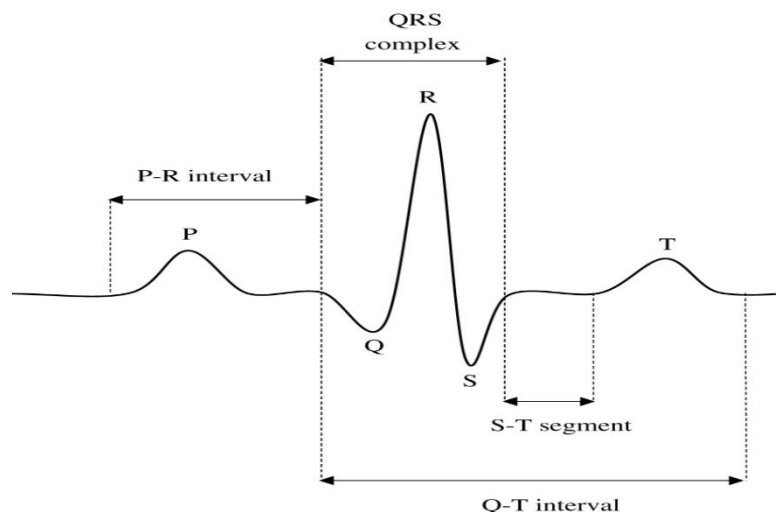


FIGURE 1 ILLUSTRATION OF ECG BEAT.

Monitoring ECG signals is crucial for several reasons:

- **Early detection:** Continuous or periodic ECG monitoring can help detect early signs of heart disease, even before symptoms appear.
- **Diagnosis:** ECG readings are essential for diagnosing various cardiac conditions, including arrhythmias, ischemic heart disease, and congenital heart defects.
- **Treatment monitoring:** For patients undergoing treatment for heart conditions, ECG monitoring provides valuable feedback on the effectiveness of interventions, such as medications, pacemaker function, and recovery after cardiac procedures.
- **Prognosis:** ECG findings can inform prognosis, helping predict the likelihood of cardiac events such as sudden cardiac death or recurrence of heart attacks.

Despite its importance, the accuracy of ECG monitoring can be compromised by various sources of noise that can significantly affect the quality of the signal [3]. There are 4 major sources of noise in ECG signals: 1) Baseline

Wander (BW), 2) Powerline Interference (PL), 3) Muscle Artefact (MA) and 4) **Electrode Motion (EM)** [4]. BW and PL have a relatively unique frequency content and thus are easily removed by simple digital filters. However, MA and EM noise are more challenging to remove as they can have a wide frequency content that overlaps with that of the ECG signal, specifically the PQRST complex [5]. This is depicted in **Figure 2** below.

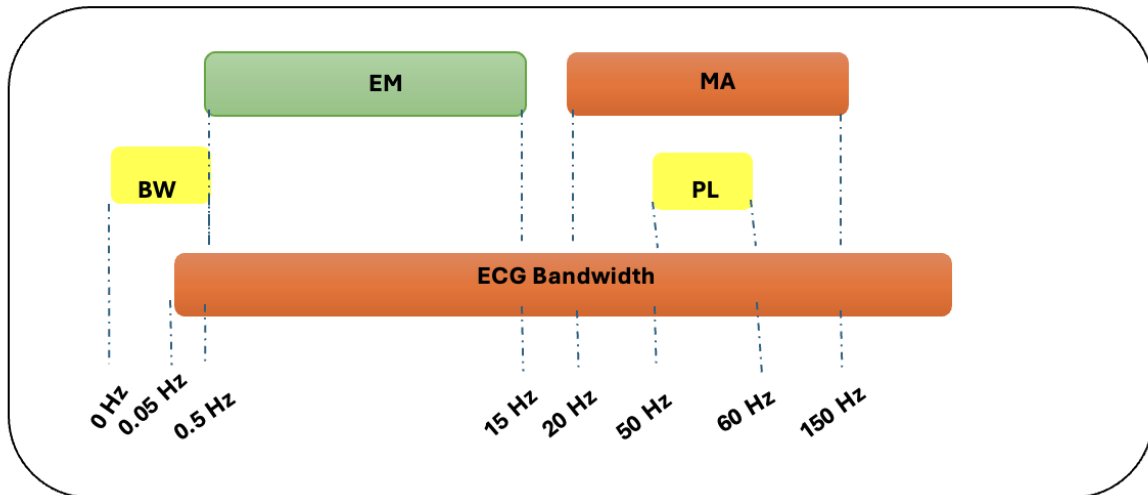


FIGURE 2 FREQUENCY CONTENT OVERLAP OF NOISE SIGNALS AND ECG.

EM noise in ECG signals refers to the interference caused by the movement of electrodes attached to the skin. When electrodes move, even slightly, they can pick up additional electrical activity not related to the heart's electrical signals. This extra activity is seen as noise in the ECG signal, which can distort the true reading. It's particularly problematic during physical activity or if the electrode doesn't adhere well to the skin. This noise appears as irregular spikes or a fuzzy baseline in the ECG trace, making it challenging to accurately interpret the heart's electrical activity. Managing EM noise is crucial for ensuring reliable ECG readings, especially in scenarios requiring patient movement or long-term monitoring. **Figure 3** shows an example of an ECG signal corrupt with EM noise, it is evident that no clinical interpretation or diagnosis of this signal could be made by an expert or algorithm. This reinforces the need for accurate and reliable filtering.

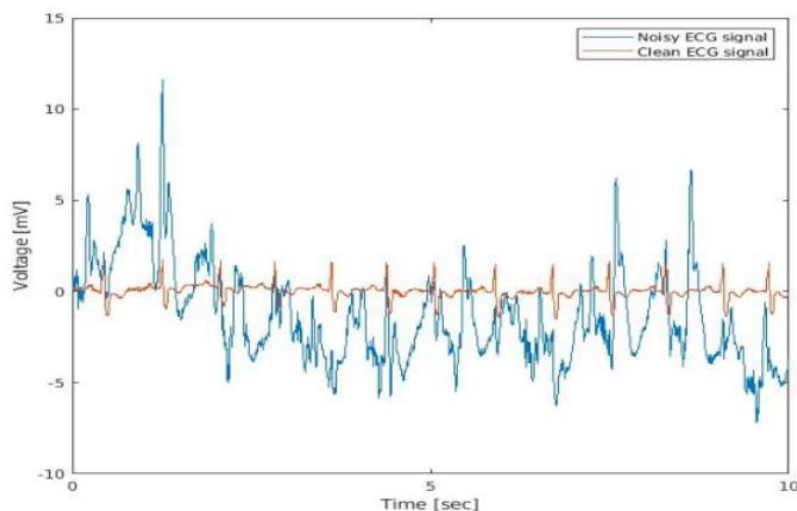


FIGURE 3 ECG SIGNAL CORRUPT WITH EM NOISE.

EM is difficult to remove from an ECG signal for various reasons. Firstly, the signal attenuated by electrode motion can be very similar to the ECG signal itself, this similarity can make it difficult to remove with digital filters which use both time and frequency domain characteristics to separate the two signals. The variability of EM noise also adds to the difficulty to remove the noise, this comes from the wide range of movements a subject can undergo such as running, walking, jumping or any other form of activity. Each activity will return a different characteristic shape of noise. Electrode placement can also vary the shape of the noise signal. Finally, individual differences such as skin type, amount of hair and other factors influencing the electrode-skin contact will affect the amount and frequency content of EM noise added during movement [6], it is important that algorithms can deal with this problem effectively.

The use of machine learning for improving ECG signal denoising has both direct and indirect environmental and societal benefits. Environmentally, by leveraging AI to enhance the accuracy of ECG readings, the need for repeated medical tests is minimized, thereby reducing the overall consumption of disposable medical equipment and lowering hospital energy consumption. Moreover, this aligns with the UK Government's Net Zero Strategy [7], which aims to decarbonize healthcare by optimizing resource use and reducing waste. The reduction in computational costs, enabled by optimized neural networks such as CNNs and RNNs, also supports energy efficiency in line with the Clean Growth Strategy [8], which emphasizes reducing energy consumption in technological sectors.

From a societal perspective, improving the quality of ECG monitoring contributes significantly to public health, especially in diagnosing and treating cardiovascular diseases, one of the leading causes of death globally. This ties into the NHS Long Term Plan [9], which highlights early detection and prevention of heart diseases as critical pillars for reducing preventable mortality. Furthermore, as AI-powered solutions become more integrated into healthcare, they promote equitable access to high-quality diagnostic tools, supporting the broader objectives of the Health and Social Care Act 2012 [10], which advocates for improving patient outcomes while addressing health inequalities across various demographic groups. By advancing these technologies, the societal benefits extend to improving patient well-being, reducing healthcare costs, and contributing to the overall efficiency and sustainability of health systems.

The objective of this work is to investigate the performance of various intelligent algorithms by developing an expansive training database and employing existing DNN architectures to reduce the amount of EM noise in a range of corrupt ECG signals. An up-to-date review on where the field of denoising ECG signals with both traditional filtering, and the more recent developments by using AI will be presented. This report will be structured as follows:

Literature Review – This section will give an overview of ECG signal processing techniques currently employed. It will then investigate EM specifically to get an appreciation of the underlying source of the noise and why it is problematic. Existing methods for removing EM noise will be presented, and their effectiveness will be discussed. Finally, AI in biomedical signal processing will be introduced, an in-depth discussion around the operating mechanisms of DNNs is presented. Any gaps remaining in the literature is discussed.

Methodology – This section will be split into two parts. The first part will focus on the strategy followed to build an expansive reference database that was used to train subsequent DNN models. This will provide details on the generation of clean lead II ECG signals alongside the generation of a large and diverse dataset consisting of EM noise signals. Finally, the methodology followed to scale each noise signal to the appropriate SNR level and combine with the clean ECG signal is detailed.

The second part will provide information on the algorithm development and subsequently, the training procedure used to determine the network parameters of each model. Finally, the strategy employed to validate the different models is presented.

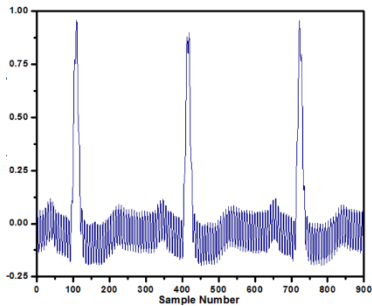
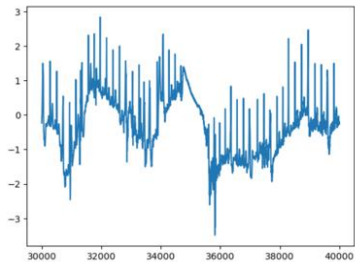
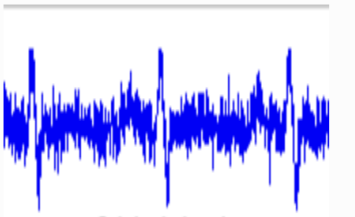
Implementation – This section will display all software tools used throughout the project life cycle. Tools such as version control, data storage, training hardware and libraries/modules will be discussed. The overall workflow of the project will be presented, and any limitations will be discussed.

Results and Discussion – This section will present a detailed overview on the results. Visual examples of the denoised signals will be presented and compared to alternative methods. Furthermore, a quantitative results section will be presented using metrics such as Root Mean Square Error (RMSE), Signal to Noise Ratio (SNR) and Cross Correlation (CC). The meaning of these results will be discussed and the applicability to real life scenarios.

3 LITERATURE REVIEW

3.1 DIGITAL SIGNAL PROCESSING IN ECG DENOISING

ECG signal processing is a critical aspect of cardiovascular disease diagnosis and monitoring. ECG signals are often contaminated by various types of noise, which can hinder accurate interpretation. This section provides information on the primary noise sources in ECG signals and the corresponding signal processing techniques used to reduce these interferences. A detailed description of each noise source can be seen in **Table 1** below:

Noise Source	Description	Impact	Example
Power Line Interference	PLI is caused by the 50/60 Hz power supply frequency coupling with the ECG signal.	Introduces a sinusoidal interference that can obscure the ECG signal components.	
Baseline Wander	Low-frequency noise resulting from patient movement, respiration, and electrode impedance changes	Causes slow variations in the baseline of the ECG signal, making it difficult to identify the true isoelectric line.	
Muscle Artefact (Electromyographic)	High-frequency noise generated by voluntary or involuntary muscle contractions.	Overlaps with the ECG signal frequency, particularly affecting the QRS complex.	


Electrode Motion	Noise caused by the movement of electrodes relative to the skin.	Produces transient baseline shifts and spikes. Overlaps with all features in an ECG.	
------------------	--	--	--

TABLE 1 TABLE SHOWING A DESCRIPTION AND THE IMPACT OF EACH NOISE SOURCE TO AN ECG SIGNAL.

To mitigate the aforementioned sources of noise, scientists, engineers and researchers utilise a variety of techniques. A primary method in signal processing for removing ECG artifacts involves decomposing the original signal into its' constituent waves or harmonics, a process known as Fourier Analysis. This approach leverages the principle of superposition, which asserts that a signal can be represented as the sum of a finite series of harmonic components. The decomposition performed in Fourier Analysis is depicted in **Figure 4** below:

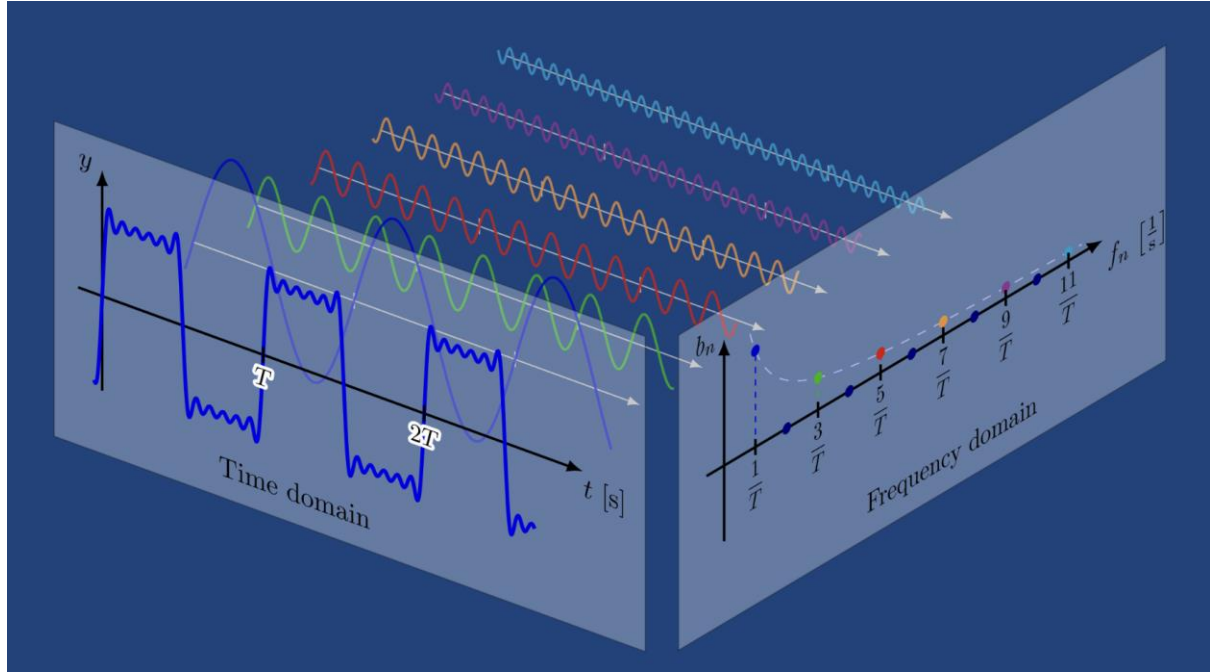


FIGURE 4 VISUAL DEPICTION OF FOURIER ANALYSIS USING THE PRINCIPLE OF SUPERPOSITION.

The decomposition of a signal into its' constituent harmonics (Sine and Cosine waves) is called the Fourier Transform [11]. It describes the process of decomposing a signal from the time domain to the frequency domain, and vice-versa. It can be described mathematically as:

$$f(\epsilon) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\epsilon x} dx \quad (1)$$

$$f(x) = \int_{-\infty}^{\infty} f(\epsilon) e^{i2\pi\epsilon x} d\epsilon \quad (2)$$

Where Equation (1) describes the transform to the frequency domain and (2) is the inverse transform back to the time domain. $f(x)$ is the original time domain signal and $f(\epsilon)$ is the corresponding frequency-domain representation.

The Fourier Transform, specifically the Discrete Fourier Transform (DFT) denoted by the above equations, is computationally complex and as such, is slow and not practical for large datasets [12]. To counteract this, an algorithm was developed that exploited the symmetry and periodicity properties of the DFT to reduce complexity, this is known as the Fast Fourier Transform (FFT) [13]. The intricacies of this algorithm are out of scope for this report, however the FFT can be stated as a computationally efficient method to transform signals into all constituent harmonics, thus providing a means to reject unwanted frequencies when reconstructing the signal. This idea underpins noise reduction in ECG signals and in fact, noise reduction in many domains.

The most popular technique to reduce noise in signals is the use of digital filters, namely low-pass and high-pass filters. Low-pass filters removed high frequency noise by allowing frequencies below a certain cut-off to pass while attenuating higher frequencies, whereas high-pass filters remove low-frequency noise by allowing frequencies above a certain cut-off to pass while attenuating lower frequencies. The combination of Low-Pass and High-Pass filters is called a Bandpass filter [14]. Different filter designs can be employed, each with their own trade-offs. For example, notch filters are used and are specifically designed for the removal of PL noise by targeting the 50/60Hz frequency band, whereas high-pass filters can be used to remove BW noise under 1Hz. Overall denoising solutions can be designed by combining these three different filter types with the objective of reducing all noise sources within an ECG signal.

Wavelet Transforms are another commonly used technique to remove noise, this technique works by decomposing the ECG signal into different frequency components, allowing for the selective removal of noise. Namely, the Discrete Wavelet Transform (DWT) [15] is often used for its ability to handle non-stationary signals, that is, the statistical properties of the signal change with time.

Empirical Mode Decomposition (EMD) decomposes ECG signals into Intrinsic Mode Functions (IMFs) for adaptive filtering [16]. IMFs are a series of oscillatory modes which can be used for adaptive filtering, making them especially useful for dealing with non-linear and non-stationary signals.

Adaptive filtering uses digital filters previously mentioned, however dynamically updates the filter parameters to capture time varying noise. Adaptive filters utilise the Least Mean Squares (LMS) and Recursive Least Squares (RLS) algorithms to adjust the parameters. In the context of this report, these are important filters as they are the most used filters for removing EM noise. Romero et al [17] provide an in-depth review of various adaptive filters. Although these filters, in general, perform well in reducing EM noise, a major limitation is that they require a reference signal alongside the ECG signal. This introduces issues when integrating into hardware such as additional computational demand or overheating. Furthermore, the effectiveness of the LMS and RLS algorithms is heavily dependent on the quality of the reference signal, if the reference signal itself is noisy the adaptive filter may struggle to effectively cancel the noise. Romero et al add skin-electrode impedance (SEI), skin stretching measured with optical sensors and accelerometers as the reference signal.

The effective processing of ECG signals requires a comprehensive understanding of various noise sources and the application of appropriate filtering techniques. The choice of method depends on the specific noise characteristics and the clinical requirements of the ECG analysis [18]. By applying these techniques, clinicians

and researchers can achieve more accurate and reliable ECG signal analysis, leading to better diagnosis and monitoring of cardiovascular conditions.

3.2 ELECTRODE MOTION REDUCTION IN ECG SIGNALS

The most widely accepted explanation for EM artifacts comes from Tam and Webster [19], who identified changes in skin potential at the skin-electrolyte interface as the primary cause of motion artifacts. Edelberg's skin model [20] explains that the skin potential is influenced by four factors: the potential across the sweat duct membrane, the total resistance within the sweat duct, the potential across the epidermis barrier membrane, and the total series resistance of the epidermis. Any variation in these factors can lead to changes in the skin potential.

The performance of many different solutions to reduce EM in ECG signals have been investigated in many studies [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]. Among these, five adaptive filters have been shown to be promising at removing EM noise [36, 37, 38, 39, 40], however as discussed above, adaptive filters require another reference signal, making them computationally demanding. Although there is evidence that adaptive filters with reference signals such as accelerometer, or skin-electrode impedance signal can be useful to reduce motion artefact, there is no conclusive evidence which reference signal performs best [41]. Furthermore, some findings contradict each other, while others are inconclusive. Removing EM noise from ECG signals is evidently a difficult problem to solve with traditional filter methods and thus it is necessary to explore alternative approaches.

3.3 INTELLIGENT SYSTEMS IN BIOMEDICAL SIGNAL PROCESSING

3.3.1 MACHINE LEARNING

Machine learning (ML) has significantly impacted biomedical signal processing by offering robust tools for the analysis and interpretation of complex biomedical data. Traditional signal processing techniques often struggle to handle the vast variability and non-stationarity of biomedical signals [42] such as ECGs, electroencephalograms (EEGs), and electromyograms (EMGs). ML algorithms, particularly those based on supervised and unsupervised learning, provide a way to automate feature extraction, classification, and pattern recognition in these signals. For instance, Support Vector Machines (SVMs) and Random Forests have effectively been used for detecting arrhythmias in ECG signals [43] and classifying different stages of sleep using EEG signals [44]. The ability of ML models to learn from data and improve over time makes them invaluable for predictive diagnostics and personalized medicine. Furthermore, these algorithms can integrate heterogeneous data sources, offering a comprehensive analysis that considers various physiological parameters [45].

In the context of noise reduction in time-series signals, traditional ML methods have not been readily investigated. Traditional ML methods face several issues when it comes to denoising ECG signals. Firstly, traditional ML algorithms like support vector machines (SVM), K nearest neighbours (KNN) and decision trees rely heavily on feature extraction which requires significant domain expertise to identify the right features characteristic of an ECG signal. This would require the extraction of R-R interval, peak amplitudes, waveform onsets and offsets amongst others. Furthermore, features would need to be extracted from the noise signals that emphasise

irregularities or non-cardiac signal patterns. This is a long and complex process that should be avoided if there is an alternative.

Additionally, there are several other factors that mean that traditional ML methods are unsuitable for this type of problem:

1. **High Dimensionality** - ECG signals are typically high-dimensional data which traditional ML models can struggle with, especially when the SNR is low.
2. **Temporal Dependencies** - ECG signals have significant temporal dependencies that need to be considered for effective denoising. Traditional ML approaches do not inherently model temporal relationships, making it challenging to capture the sequential nature of the data.
3. **Non-Stationarity** - ECG signals are non-stationary; Traditional ML models generally assume stationary datapoints or require the data to be pre-processed to remove non-stationarity. In this case, EM noise adds significant non-stationarity and thus, it is crucial that this is not removed.
4. **Variance** - The characteristic shape of EM noise in ECG signals can vary significantly in terms of frequency and amplitude, ML models require explicit modelling which can be complex and time consuming.

Goodfellow et al [46] concluded that traditional ML are inefficient for many modern applications, meaning that they require many observations for achieving generalizability, and imposing significant human labour to specify prior knowledge in the model. All these points provide evidence as to why traditional learning algorithms are not suitable for this type of problem and imply that a learning technique designed for non-linear, non-stationary and automatic feature extraction may be required.

3.3.2 DEEP LEARNING

Deep learning (DL), a subset of ML, has revolutionized biomedical signal processing by providing more sophisticated methods for analysing complex biomedical data. Unlike traditional ML techniques that require manual feature extraction, DL models automatically learn hierarchical representations of the data, which can capture intricate patterns and temporal dependencies. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are among the most popular architectures used in this domain. CNNs have shown exceptional performance in analysing spatial data and have been used to detect abnormalities in ECG signals, such as myocardial infarctions and atrial fibrillations [47]. RNNs, particularly Long Short-Term Memory (LSTM) networks, are effective in modelling sequential data, making them suitable for time-series analysis. These models have demonstrated superior accuracy and robustness in various biomedical applications, from

disease diagnosis to brain-computer interfaces. **Figure 5** depicts the structure of a Deep Neural Network (DNN) with 3 hidden layers, the full connections of nodes here are referred to as fully connected layers.

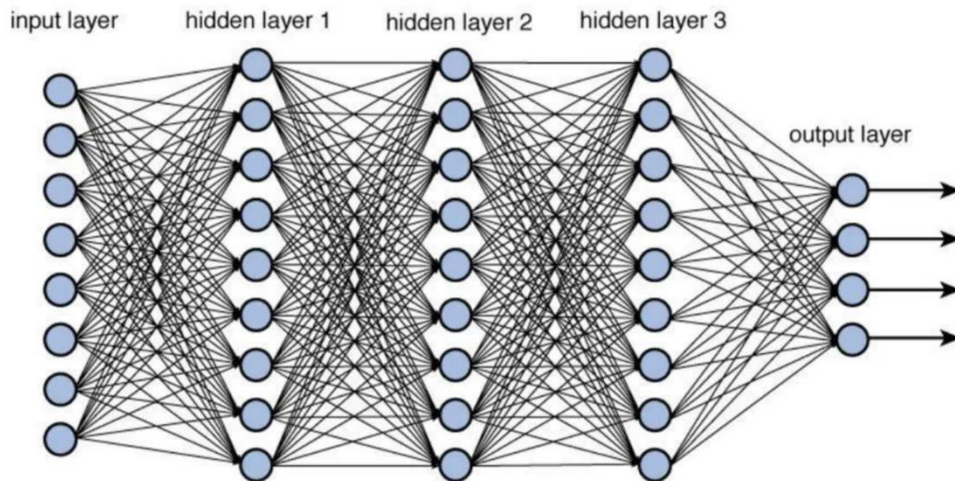


FIGURE 5 ILLUSTRATION OF DNN STRUCTURE.

A DNN processes input data through multiple layers, each consisting of interconnected nodes. These layers consist of an input layer, two or more hidden layers and an output layer. The input layer receives the raw data which we can represent as a vector $x \in R^n$ where n is the number of features in the data. This vector is then passed to the first hidden layer. Each hidden layer performs a linear transformation followed by a non-linear activation function. We can describe the transformation at the i^{th} hidden layer as:

$$z_i = W_i a_{i-1} + b_i \quad (3)$$

Where z_i is the input to the activation function of the i^{th} layer. W_i is the weight matrix of the i^{th} layer. a_{i-1} is the output from the prior layer and b_i is the bias vector for the i^{th} layer. It should be noted that W and b are parameters that the network wishes to optimise, meaning these values should be set at each node in a way that minimises some loss function.

As mentioned previously, traditional ML algorithms were solely based on the assumption that the relationship connecting the input and output labels is linear. This is suitable if the problem is linear, or can be easily linearised, however for most engineering problems the input and output data will be non-linear [48]. Thus, the introduction of a non-linear component in the networks is required. This component is called an activation function and has several benefits, including helping the DNN converge, improving the computation efficiency and increasing the accuracy of a model. The output of the i^{th} hidden layer is obtained by applying the activation function σ to z :

$$a_i = \sigma(z_i) \quad (4)$$

The choice on the activation function depends on the network requirements, however there are four primary activation functions that are used in practice: 1) Tanh, 2) ReLu, 3) Sigmoid and 4) Linear. These are depicted in **Figure 6** below:

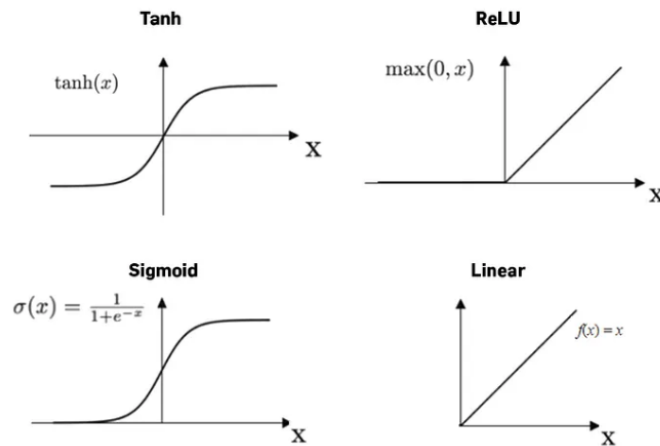


FIGURE 6 FOUR PRIMARY ACTIVATION FUNCTIONS USED IN DNNs.

Finally, the final hidden layer's output is passed to the output layer, which performs another linear transform to produce the networks output.

In DNN's, every layer of nodes trains on a different set of features based on the previous layers' output, the network recombines and learns from features of the previous layer. This means that the more layers in the network, the more complex the features it can extract, however this also increases computational complexity.

For time series data, CNN's and RNN's are the two most popular network architectures. However, Auto-Encoders (AE's) have also been employed.

3.3.2.1 CONVOLUTIONAL NEURAL NETWORKS

CNN's consist of multiple back-to-back layers connected in a feed-forward manner. The main layers are including a convolutional layer, pooling layer and a fully connected layer. The first two layers (convolutional and pooling) are responsible for feature extraction, while the fully connected layer performs the classification. **Figure 7** shows an example of a CNN.

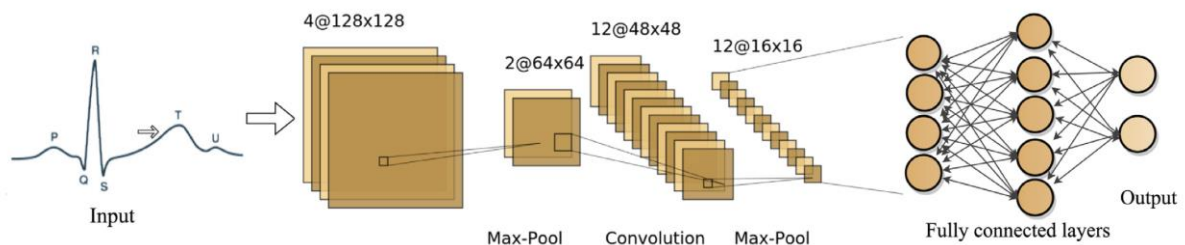


FIGURE 7 ILLUSTRATION OF CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE.

3.3.2.1.1 CONVOLUTIONAL LAYER

The convolutional layer is the core building block of a CNN and is where most of the computation occurs. It requires a few components; 1) Input data, 2) A filter and 3) A feature map. The input data is a numerical representation of the type of problem being solved, for example, if the input is an image the input will have 3 dimensions (height, width and depth) which correspond to red, green and blue (RGB) in an image. If the input data is a time-series signal, the input will be a 2-dimensional signal (value on Y-Axis and value on X-axis). The filter (herby referred to as 'kernel') is a 2-dimensional array of weights that represent a portion of the input, it is applied to an area of the overall input sequence and the dot product is computed between the input portion and the kernel which is stored in an output array. The kernel is then iteratively shifted by a 'stride' until the kernel has swept across the entire input signal. The resulting array from the dot product of each portion is known as the 'feature map'. This process of shifting the kernel across the entire signal and computation of the dot product is known as convolution. This process is illustrated in **Figure 8** below:

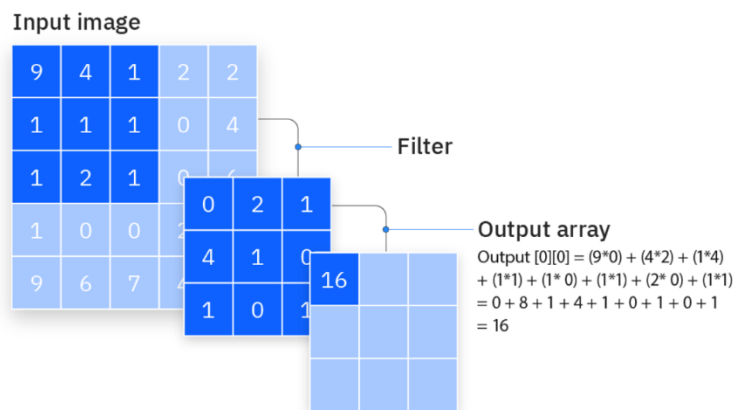


FIGURE 8 ILLUSTRATION OF CONVOLUTION PROCESS IN CNNs.

During this process, the weights in the kernel remain fixed as it shifts across the input in a process known as parameter sharing, however there are three hyperparameters (parameters that will affect the output of the CNN) that need to be 'tuned' before the training of a CNN. These are:

1. **Number of Kernels** – The number of kernels will affect the depth of the output, for example three distinct kernels will yield three different feature maps, creating a depth of three.
2. **Stride** – The distance the kernel moves over the input will affect the size of the output. In general, a larger stride will return less outputs.
3. **Zero-Padding** – This is required when the kernel/s do not fit the input. This sets all elements that fall outside the input matrix to zero, producing a larger output.

Note: There are three different types of padding; 1) Valid padding (drops the last convolution if dimensions do not align), 2) Same Padding (Ensures output layer has the same size as input layer) and 3) Full Padding (Increases the size of the output layer by adding zeros to the border of the input).

3.3.2.1.2 POOLING LAYER

The pooling layer is responsible for dimensionality reduction. This means it reduces the number of parameters in the input signal. The pooling operation has a similar mechanism to the convolutional layer in that it sweeps a kernel across the input, however this kernel does not have any weights. Instead, the kernel applies an aggregation function to the values within the input portion. There are two main types of aggregation functions used in this layer:

1. **Max Pooling** – As the kernel shifts across the input sequence, it selects the value with the maximum value to send to the output array.
2. **Average Pooling** – As the kernel shifts across the input sequence, it calculates the average value within the portion and sends it to the output array.

While this layer can lose a lot of information, the purpose of this layer is to reduce the complexity, improve efficiency and limit the risk of overfitting. Examples of both aggregation functions can be seen in **Figure 9** and **Figure 10** below:

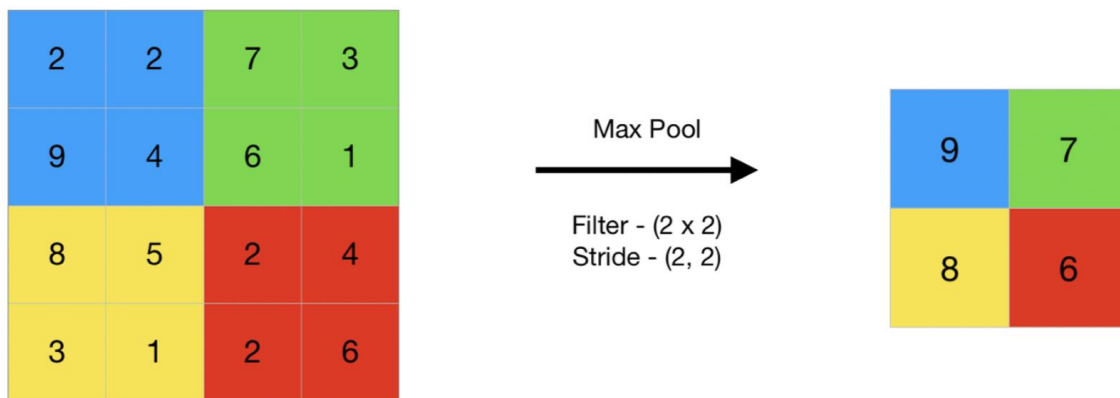


FIGURE 9 ILLUSTRATION OF MAX POOLING AGGREGATION FUNCTION USED IN THE POOLING LAYER OF A CNN.

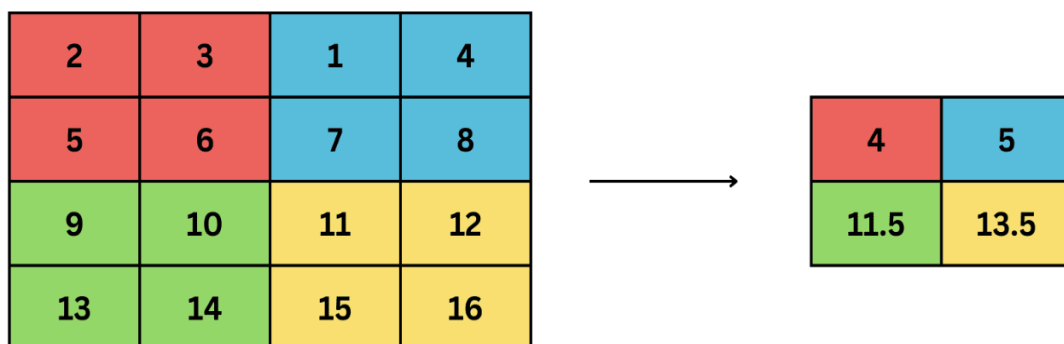


FIGURE 10 ILLUSTRATION OF AVERAGE POOLING AGGREGATION FUNCTION USED IN THE POOLING LAYER OF A CNN.

3.3.2.1.3 FULLY CONNECTED LAYER

This layer is simply a fully connected layer as depicted in **Figure 5**. This layer performs classification based on the features extracted in the previous layers and their different kernels. While the convolutional and pooling layers use ReLu activation functions, this layer will leverage a SoftMax activation function to classify inputs appropriately, producing a probability from 0 to 1.

3.3.2.2 RECURRENT NEURAL NETWORKS

An RNN is a specific architecture of a deep learning model that is designed to process and convert a sequential data input into a specific sequential data output. Sequential data is data such as words, sentences or time-series data. RNNs are largely being replaced by transformer-based AI and Large Language Models (LLMs) [49], however for time-series data, these models are still relevant. The structure of a general RNN is shown below in **Figure 11**.

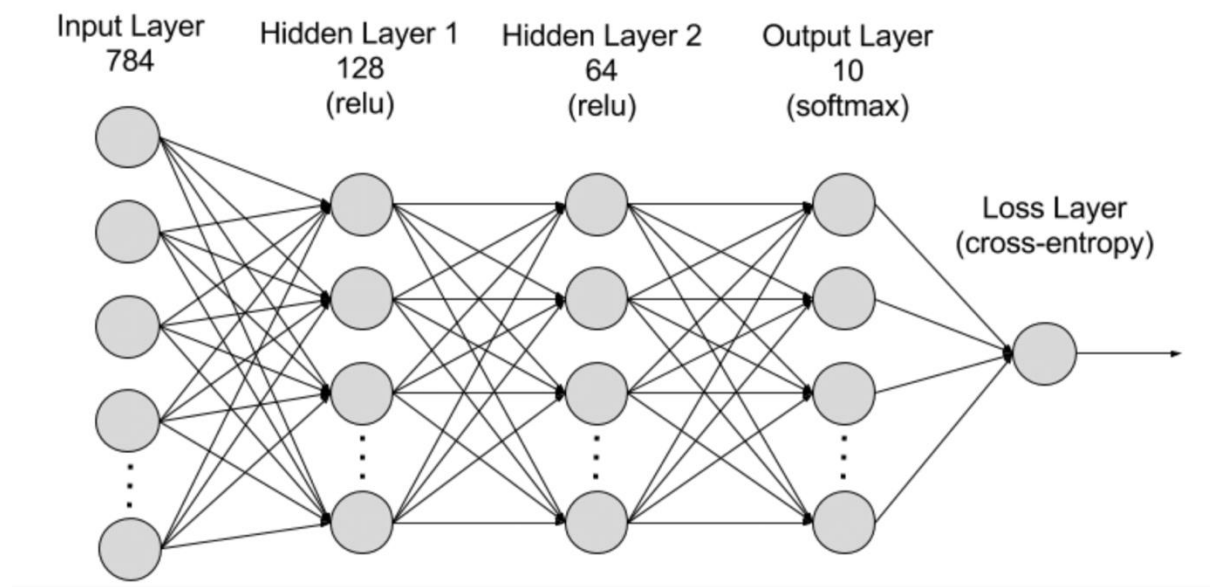


FIGURE 11 ILLUSTRATION OF RNN.

RNNs operate by sequentially passing data through their hidden layers, one step at a time. A key characteristic of RNNs is their recurrent structure, meaning the hidden layer retains information from previous inputs, allowing it to use past data for future predictions through a short-term memory component. The model uses both the current input and the stored memory to predict the next step in a sequence. There are several types of RNNs, including:

- One-to-Many, where a single input is used to produce multiple outputs,
- Many-to-Many, where multiple inputs are processed to generate multiple outputs, and
- Many-to-One, where multiple inputs are condensed into a single output.

The choice of RNN type depends on the specific application, and various RNN variants are available to address different tasks more effectively. The primary variants are as follows:

1. **Bidirectional RNNs** – Processed data sequences with forwards and backward layers of hidden neurons. The forward layer works like the general RNN where the previous input is stored in the hidden state and uses it to predict the subsequent output. Meanwhile, the backward layer works in the opposite direction by taking both the current input and future hidden state to update the present hidden state, thus improving accuracy.
2. **Long short-term memory (LSTM)** – This variant enables the model to expand its' memory capacity to accommodate a larger input sequence. This is important as a standard RNN can only remember the immediate past input and not inputs from several previous sequences.
3. **Gated Recurrent Units (GRU)** – A GRU is an RNN that enables selective memory retention. The model adds an update and forgets the gate to its' hidden layer, which can store or remove information in the memory.

3.3.2.3 AUTO-ENCODERS

Autoencoders are a type of DNN architecture designed for representation learning, meaning they automatically learn to extract and organize important features from raw input data. These learned features can then be used for tasks such as classification, regression, or clustering. Autoencoders accomplish this by introducing a bottleneck in the network, which compresses the input data into a smaller, more efficient representation. By forcing the input through this bottleneck, the network is encouraged to capture meaningful, correlated features. If the underlying structure of the input is well-represented, the autoencoder can effectively learn and leverage this information. The general architecture is illustrated in **Figure 12**.

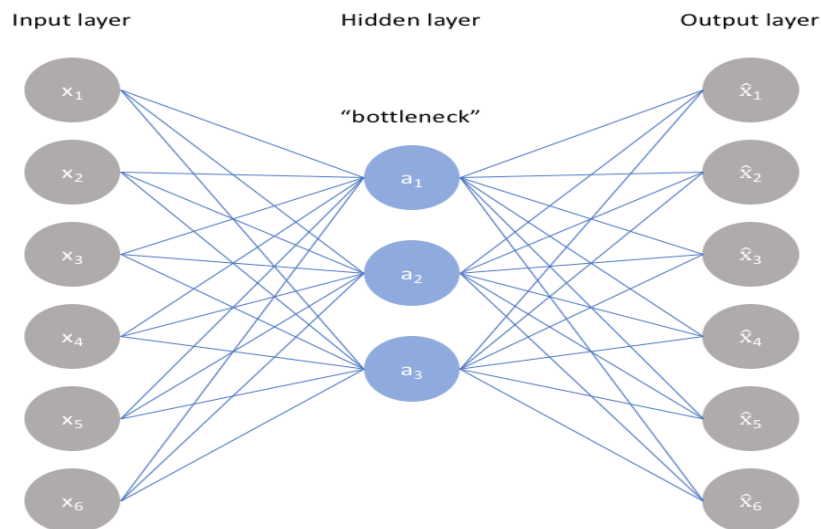


FIGURE 12 ILLUSTRATION OF AE.

As visualised in **Figure 12**, an autoencoder can be trained on an unlabelled dataset by framing the task as a supervised learning problem, where the objective is to reconstruct the original input x from the models' output x' . The network is trained by minimising the reconstruction error $L(x, x')$, which measures the difference between the original input and its' reconstruction. An ideal AE should be sensitive enough to capture essential features from the inputs for accurate reconstruction, while being robust enough to avoid simply memorising the data, which can lead to overfitting. To achieve this, the loss function must include two components: one that encourages the

model to capture relevant features from the input and another, known as the regularizer, that discourages memorisation and overfitting. The determination of the loss can be expressed as:

$$Loss = L(x, x') + regularizer \quad (5)$$

Where $L(x, x')$ is the reconstruction error.

There are number of variants of AE's, these include 1) Undercomplete AE's, 2) Sparse AE's, 3) Denoising AE's and 4) Contractive AE's.

3.3.2.3.1 UNDERCOMPLETE AUTOENCODER

The simplest architecture is known as an undercomplete AE which constrains the number of nodes present in the hidden layers of the network. By penalizing the network according to the reconstruction error, the model can learn most of the important attributes of the input data and how best to reconstruct the original signal. This model does not use a regularizer, and thus the only way to ensure the model isn't memorising the input, is to adequately constrain the number of nodes in the hidden layers.

3.3.2.3.2 SPARSE AUTOENCODERS

Sparse AE's offer an alternative method for introducing an information bottleneck, without requiring a reduction in the number of nodes. The loss function is constructed such that activations are penalised within a layer. For any given observation, the network is encouraged to learn an encoding and decoding which only relies on activating a small number of neurons. This is depicted below in **Figure 13** where the opacity of a node corresponds with the level of activation.

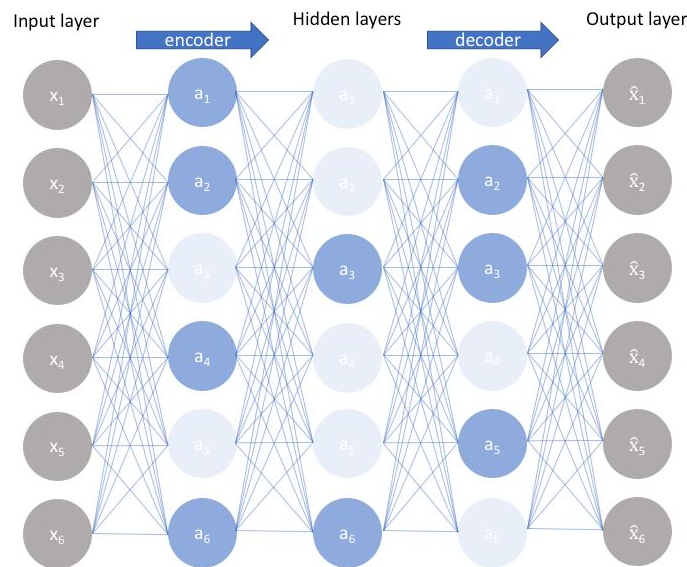


FIGURE 13 ILLUSTRATION OF SPARSE AE.

Unlike the undercomplete AE, the sparse AE can limit the networks capacity to memorise the input data without limiting the networks capability to extract features.

3.3.2.3.3 *DENOISING AUTOENCODERS*

Denoising AE's are designed to learn a vector field for mapping the input data towards a lower-dimensional manifold. If the manifold accurately describes the natural data, the added noise can be removed.

3.3.2.3.4 *CONTRACTIVE AUTOENCODERS*

Contractive AE's are similar to denoising AE's mentioned in **Section 2.3.2.3.3** however they require the derivative of the hidden layer activations to be small with respect to the input. Contractive AE's make the feature extraction function (encoder) resist small perturbations in the input.

3.3.2.4 *DNN TRAINING*

The training of DNNs is a complex process that involves various optimisation algorithms. To get an understanding of the process, it is important to be familiar with the following concepts:

- **Epoch** – Refers to one iteration where the model iterates across the entire training set to update its weights.
- **Mini-batch gradient descent** – During training, updating weights is not done in individual data points, rather small batches of data for more efficient computing. Gradient descent refers to the process that minimises the loss function.
- **Loss Function** – The loss function is a statistical measure used to compare the similarity of the predicted output against the true output.

The most fundamental algorithm behind DNN training is known as backpropagation (BP) and provides a means to iteratively update the weights and biases during training in a way that minimises the loss function. In general, it can be said that to train a DNN, the model parameters (weights and bias) should be initialised randomly on the first pass and the signal should be propagated through the network to obtain the corresponding output. The difference between the output and the actual output (Loss) should be computed and used to update the weights until the loss is minimised through BP. Whenever the loss is minimised, the parameters are 'locked' and used to make predictions on new unseen data. This process is illustrated below in **Figure 14**:

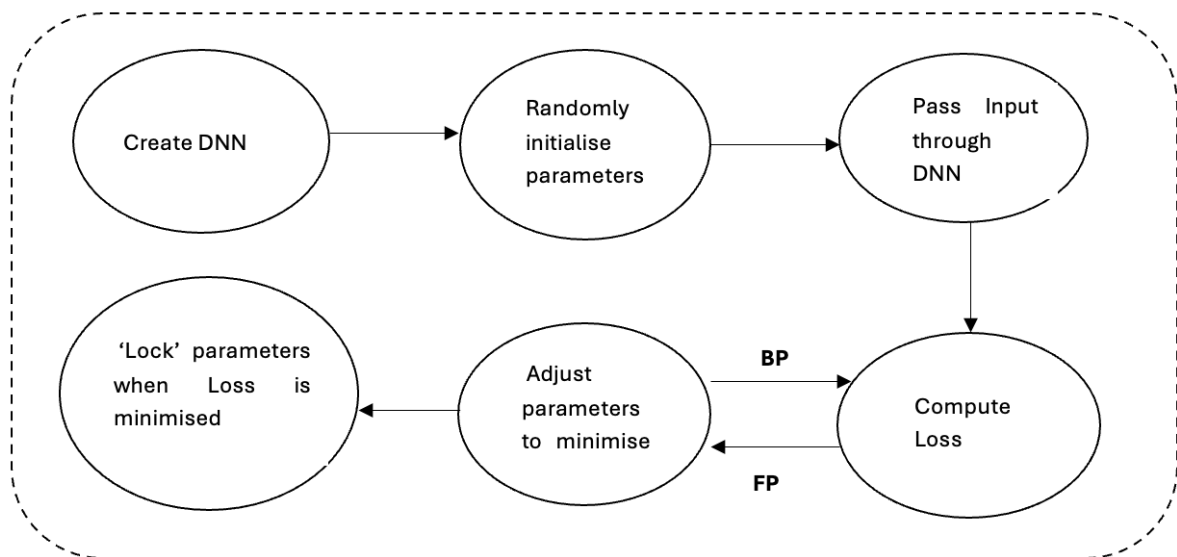


FIGURE 14 TRAINING PROCESS OF DNN.

3.3.2.4.1 BACKPROPAGATION

BPs' objective is to fine tune the weights and bias of all nodes contained within a DNN based on the loss obtained in the previous epoch. Effective tuning of weights ensures lower error rates, making the model reliable by increasing its generalization. There are four fundamental equations behind the BP algorithm, ultimately these try and underpin how changing the weights and bias at each individual neuron changes the loss function (With the aim to minimise loss). In other words, it is interested in computing two partial differential equations $\frac{\partial L}{\partial w_{jk}^l}$ and $\frac{\partial L}{\partial b_j^l}$ where L is the loss, W is the weight, b is the bias term and l, j, k represent layer number, neuron in current layer, and the neuron in the previous layer respectively. The equations of BP can be derived by looking at a single neuron j in l^{th} layer of the network as depicted below in **Figure 15**.

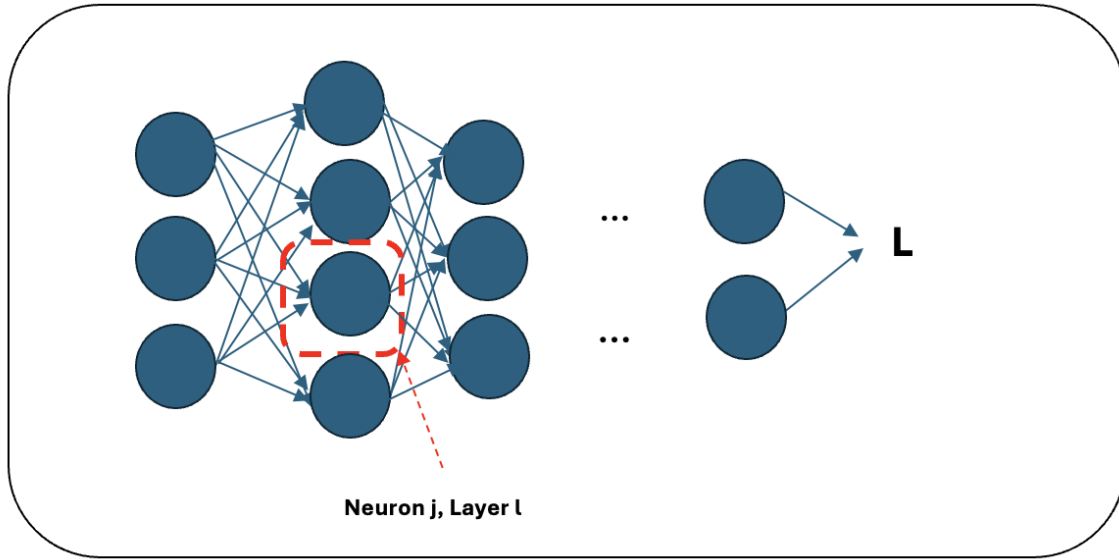


FIGURE 15 LLUSTRATION OF INDIVIDUAL NEURON ISOLATION FOR BP DERIVATION.

To get an appreciation of the importance of the BP algorithm, we can investigate its' effect on a single neuron. Neuron j in layer l in **Figure 15** above will have an associated weight and bias value, these terms will be updated such that the loss L is minimised. As an input comes into the neuron, we can add a small change to the weight input Δw_j^l . Instead of outputting $\sigma(w_j^l)$, the neuron will now output $\sigma(w_j^l + \Delta w_j^l)$ which will be propagated through later layers in the network causing an overall change of $\frac{\partial C}{\partial w_j^l} \Delta w_j^l$. Thus, it is evident that $\frac{\partial C}{\partial w_j^l}$ is a measure of error in the neuron:

$$\delta_j^l = \frac{\partial C}{\partial w_j^l} \quad (6)$$

Where δ_j^l is the error in the j^{th} neuron in the l^{th} layer. BPs' objective is to compute δ_j^l and relate it back to $\frac{\partial C}{\partial w_{jk}^l}$ and $\frac{\partial C}{\partial b_j^l}$. The BP algorithm can be explained by four primary equations, in which each are a consequence of the chain rule. Applying the chain rule to equation (6), we get:

$$\delta_j^l = \sum_k \frac{\partial C}{\partial a_k^l} \frac{\partial a_k^l}{\partial w_j^l} \quad (7)$$

Where a_k^l is the output activation function of the l^{th} layer. Since a_k^l only depends on the weighted input w_j^l for the j^{th} neuron with $k = j$ and so $\frac{\partial a_k^l}{\partial w_j^l}$ vanishes when k does not equal j :

$$\partial_j^l = \frac{\partial C}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \quad (8)$$

If we recall that $a_j^l = \sigma(w_j^l)$, the 2nd term above can be written as $\sigma'(w_j^l)$, thus:

$$\delta_j^l = \frac{\partial C}{\partial a_j^l} \sigma'(w_j^l) \quad (9)$$

Equation (9) is the 1st of the BP equations which will be referred to as BP1.

Next, we will apply the chain rule to find the error in terms of the error in the next layer ($l + 1$). We can rewrite $\delta_j^l = \frac{\partial C}{\partial w_j^l}$ as:

$$\delta_j^l = \sum_k \frac{\partial C}{\partial w_k^{l+1}} \frac{\partial w_k^{l+1}}{\partial w_j^l} \quad (10)$$

$$\delta_j^l = \sum_k \frac{\partial w_k^{l+1}}{\partial w_j^l} \delta_k^{l+1} \quad (11)$$

$$w_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} \quad (12)$$

$$w_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(w_j^l) + b_k^{l+1} \quad (13)$$

Differentiating this and subbing back into equation (6) gives:

$$\frac{\partial w_k^{l+1}}{\partial w_j^l} = w_{kj}^{l+1} \sigma'(w_j^l) \quad (14)$$

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(w_j^l) \quad (15)$$

This equation is known as BP2. Still using the chain rule, we can show that BP3 and BP4 are given by:

$$\delta_j^l = \frac{\partial C}{\partial b_j^l} \quad (16)$$

$$\delta_j^l a_k^{l-1} = \frac{\partial C}{\partial w_{jk}^l} \quad (17)$$

BP1, BP2, BP3 and BP4 form the basis of the BP algorithm.

Despite significant advancements, several gaps remain in the application of machine learning and deep learning to biomedical signal processing. One of the primary challenges is the lack of large, high-quality annotated datasets, which are crucial for training robust and generalizable models [50]. Many existing studies rely on small or proprietary datasets, limiting the reproducibility and comparability of results. Moreover, most current research focuses on specific types of biomedical signals, such as ECG or EEG, with less attention given to multimodal data integration [51], which could provide a more holistic understanding of physiological conditions. Another gap is the limited exploration of model interpretability and explainability. In clinical settings, it is vital to understand how a model arrives at a decision to ensure its reliability and to gain the trust of healthcare professionals [52]. Additionally, there is a need for more research on the real-time implementation of these models in clinical practice,

considering computational efficiency and scalability [53]. Addressing these gaps requires a concerted effort to develop open-access datasets, improve model interpretability, and focus on translational research that bridges the gap between algorithm development and clinical application.

4 METHODOLOGY

The Physionet database [54] is a commonly used publicly available database with large amounts of annotated ECG signals that can be used to both develop and validate biomedical algorithms. It is commonly used for the development of classification algorithms where the aim is to classify various cardiac abnormalities present. Only a few databases have also been used to develop and validate intelligent denoising algorithms, for example, Brophy et al [55] used two Physionet databases. The first of which was the MIT-BIH Arrhythmia database [56] which is comprised of 48 half hour records of two-channel ambulatory ECG. The second database was the MIT-BIH Noise Stress Tests Database (NSTDB) [57] which includes 12 half hour ECG recordings and three half hour recordings of noise typical in ambulatory ECG recordings, only the noise recordings were used.

The MIT-BIH database was selected as the ECG recordings are relatively clean, and thus provide a suitable ground truth. However, this database is small and only contains 48 records and so will miss much of the variability that ECG waveforms can exhibit. Furthermore, although the signals are relatively clean, they will still contain a small amount of noise due to movement or another source. Finally, the NSTDB database only contains 30 minutes of noise signal, recorded from a single device which means that models would be trained against a very small number of possible shapes of noise. This would fail to capture the wide range of EM artefacts observed in real life.

These factors were deemed enough to pursue the development of a much larger, clean reference database that could be used to develop and validate intelligent signal denoising algorithms. Due to limitations on using real ECG signals, it was necessary to simulate clean ECG signals using an established model. The ECGSYN [58] tool was used to simulate clean signals with different waveform morphologies and heart rates.

It was important that each signal had a different morphology to maximise variability into the database. The amplitude, width and slope of the P, QRS and T waves are features that were changed on each clean signal generation. Signals of different heart rates were also generated within a range that might be seen in the event of EM corruption. It is critical that the training database for DNNs contains a high variability of data, as it helps the model generalise better and improves robustness [59].

4.1 DATABASE GENERATION

4.1.1 NOISE EXTRACTION

The EM noise signal was extracted from the NSTDB with a sampling frequency of 360Hz. To ensure sampling frequency was consistent across the entire experiment, this was immediately up sampled to 500Hz. The raw noise signal after being up sampled to 500Hz can be seen in **Figure 16** below.

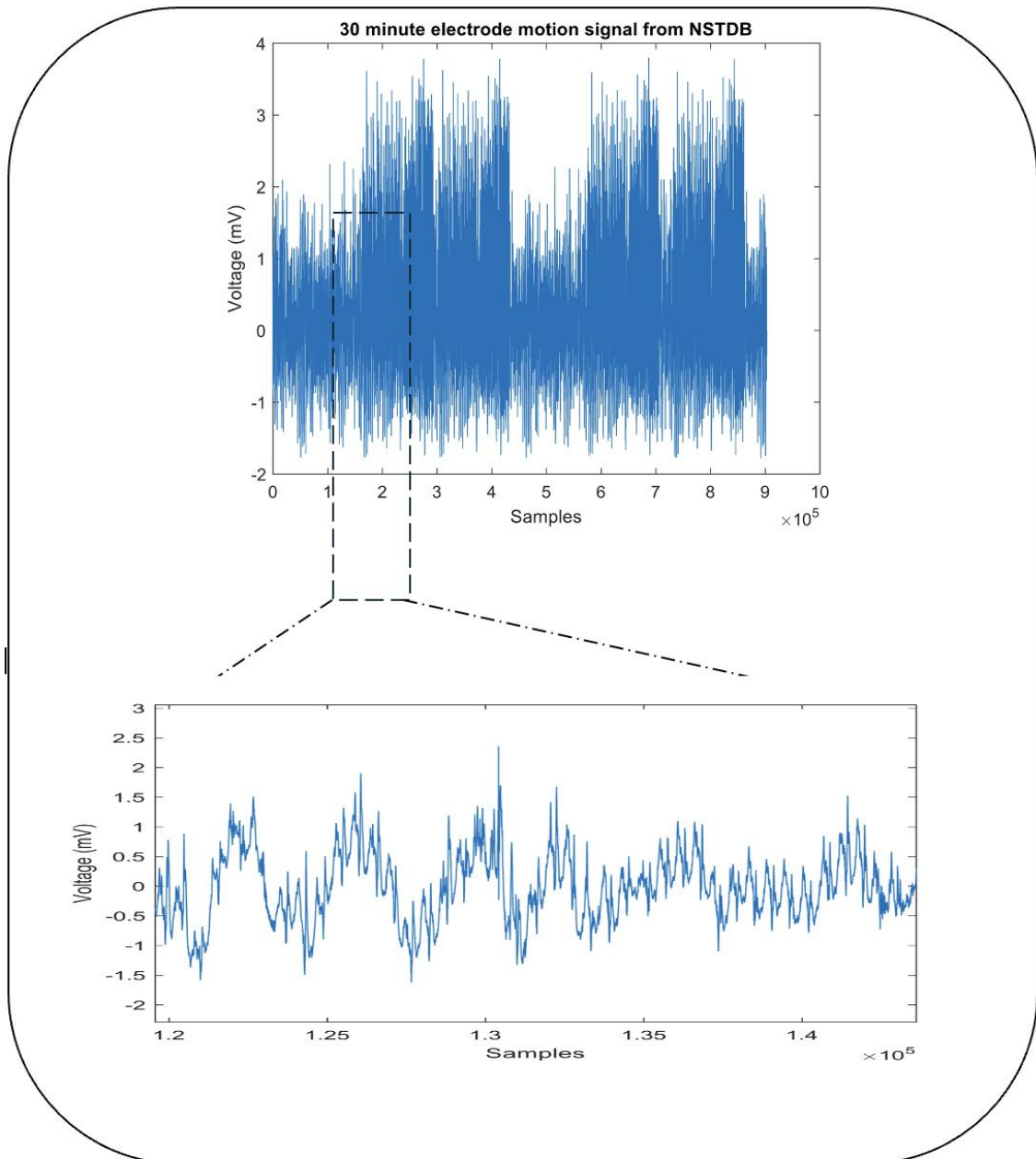


FIGURE 16 DEPICTION OF NOISE EXTRACTED FROM PHYSIONET.

There was a small amount of BW noise identified within this signal. The removal of this noise source is out of scope for this report and thus it was deemed necessary to remove this low-frequency component. The raw signal shown above was passed through a 20th order type II Chebyshev bandpass filter with 60dB attenuation. The cut-off frequencies were set at 1Hz and 50Hz to ensure a maximum amount of EM noise was retained. The final prepared noise signal can be seen in **Figure 17** below:

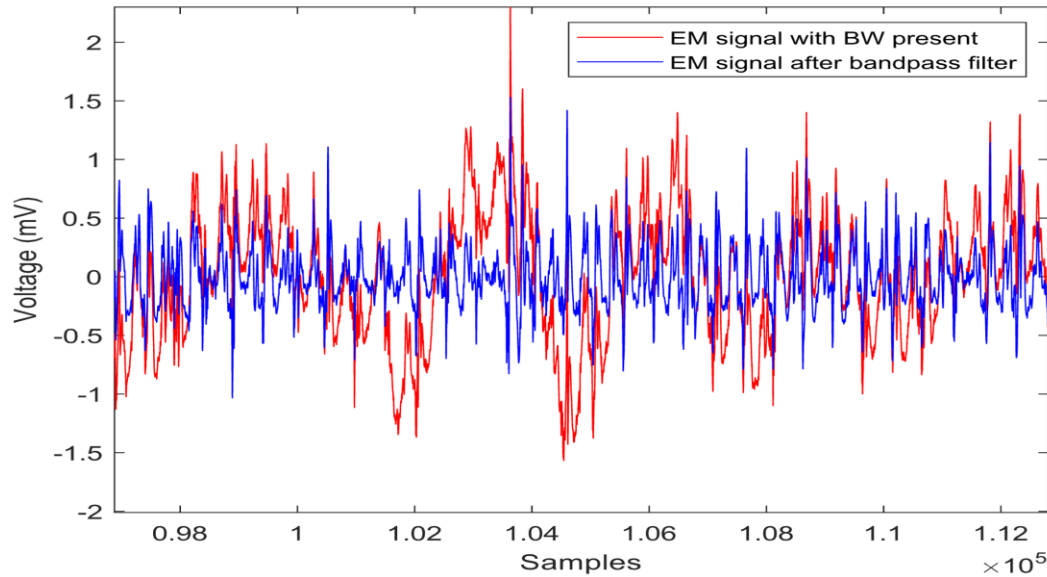


FIGURE 17 RESULTS OF REMOVING BASELINE WANDER FROM NOISE.

4.1.2 CLEAN SIGNAL DATASET GENERATION

Synthetic clean signals were generated using ECGSYN and an algorithm was developed to generate a large dataset that met the following requirements:

1. **Signal Variability** - Each clean signal would have different waveform parameters. The width, amplitude and slope would be different in each signal.
2. **Sample Frequency** - All clean signals were generated with a sampling frequency of 500Hz.
3. **Representative** - Each clean signal would contain realistic waveform morphologies, that could be seen in real life measurements.
4. **Heart Rate Coverage** - The clean signals would be generated at 50, 60, 70, 80, 90 and 100 Beats Per Minute (BPM)

To meet **requirement (1)**, The algorithm utilised a sampling method known as Latin Hypercube Sampling (LHS) to randomly sample parameter values between defined limits. These limits refer to the minimum and maximum widths, amplitudes and slopes of each waveform. The minimum and maximum parameters set in this study can be seen in **Table 2** below:

Wave	Angles of Extrema [Min, Max]	Z Position of Extrema [Min, Max]	Gaussian Width [Min, Max]
P	[-60, -80]	[0.9, 1.5]	[0.25, 0.75]
Q	[-12, -25]	[-7, -2]	[0.1, 0.4]
R	[0, 0]	[20, 45]	[0.1, 0.8]
S	[12, 25]	[-10, -5]	[0.1, 0.6]
T	[80, 120]	[0.5, 1]	[0.1, 0.6]

TABLE 2 WAVEFORM LIMITS USED IN LHS.

Note: The selection of these minimum and maximum boundaries was arbitrary, however were selected such that the generation of unrealistic ECG waveforms was unlikely.

LHS is utilised due to its' ability to control sample points in the parameter. That is, we can ensure that the sampling point distribution is close to the probability density function (PDF). In this case, we assume that each parameter has an equal probability of falling in any region between the defined boundaries, thus, LHS avoids the probability that all sampling points come from the same local region. This is depicted in **Figure 18** below:

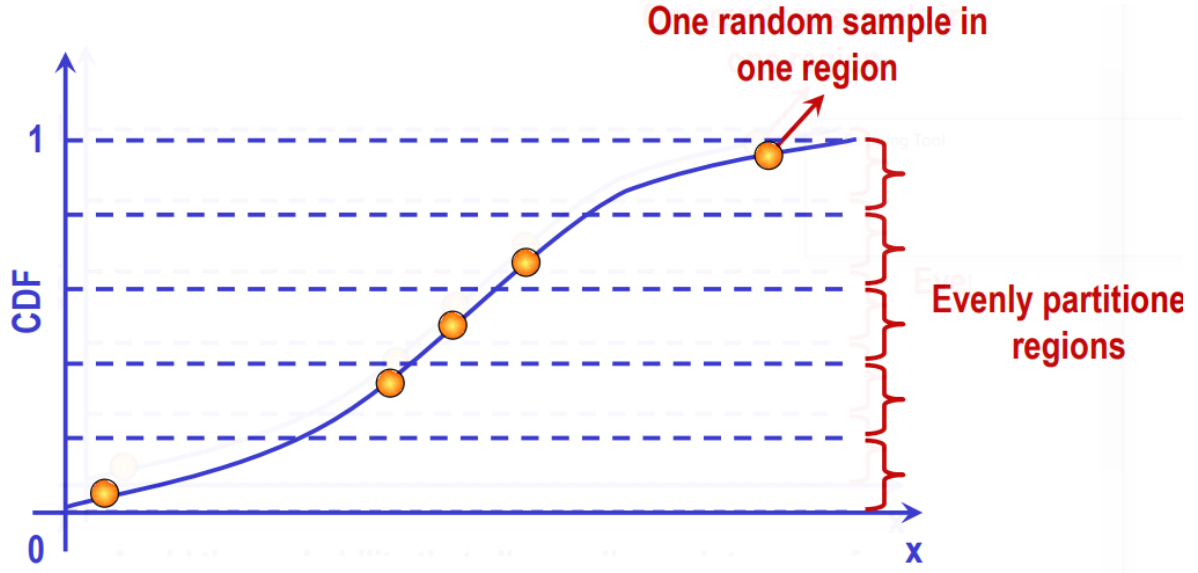


FIGURE 18 LHS METHODOLOGY.

Practically, and in the context of this work, this means that feature waves (P, QRS, T) will have a uniform distribution of sample values that covers the entire range.

When each parameter value is set based on the above, the ECG signal is generated using Physionets ECGSYN algorithm. ECGSYN generates synthetic ECG signals by modelling the electrical activity of the heart as a sequence of quasi-periodic waveforms. These waveforms are constructed from a set of characteristic points corresponding to the P, Q, R, S, and T waves in a typical ECG cycle. The tool uses a combination of differential equations and Gaussian functions to produce these waveforms. The derivation of each ECG can be characterized by the following equation:

$$ECG(t) = \sum_i A_i \exp\left(-\frac{(t-t_i)^2}{2\sigma_i^2}\right) \quad (18)$$

Where A_i is the amplitude of the i -th wave (P, Q, R, S, T), t_i is the time position of the i -th wave, and σ_i^2 is the width (standard deviation) of the i -th wave.

A clean ECG signal with 256 beats is generated using this method. As stated above, HR and Waveform morphology are the only parameters that change in this experiment. A validation step is performed after each clean signal is generated to ensure the signal is realistic.

To meet **requirement (3)**, after a clean signal was generated, it went through a validation stage. An algorithm was developed to ensure that the P wave, QRS complex and T wave all had dimensions that are reasonable. The algorithm validates this by determining the onsets, peaks and offsets of each waveform. These characteristics can be seen below in **Figure 19**:

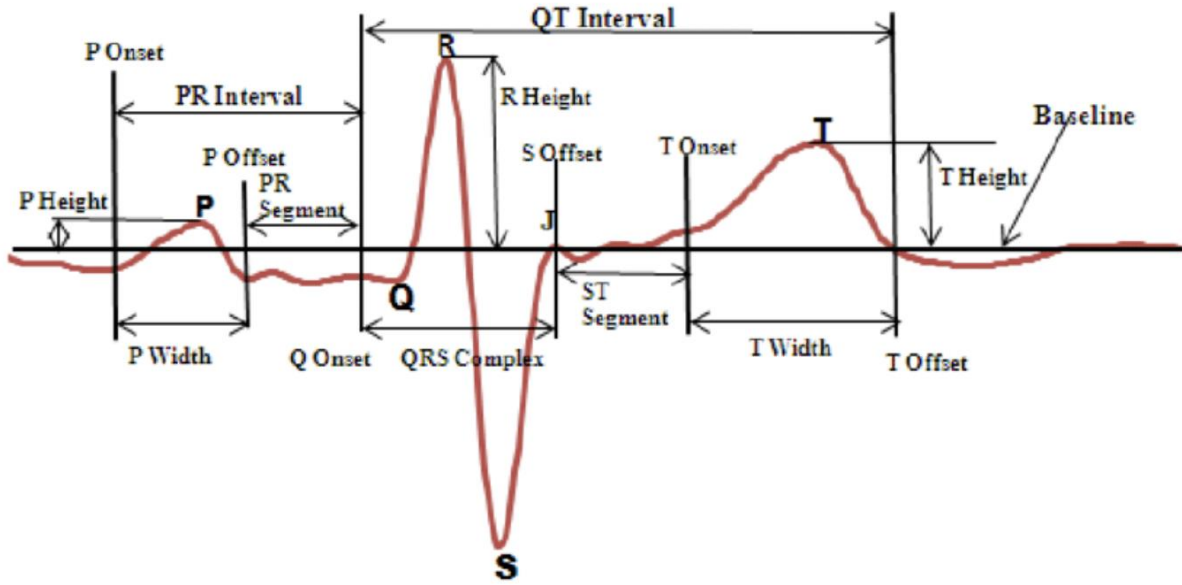


FIGURE 19 FULLY ANNOTATED ECG BEAT.

In this study, seven primary measurements were used to filter out unrealistic ECG signals. These were 1) P Wave Duration, 2) QRS Duration, 3) T Wave Duration, 4) P Wave Amplitude, 5) QRS Amplitude, 6) T Wave Amplitude and 7) QT Interval. The calculations of these measurements are as follows:

$$P_{Duration} = P_{Offset} - P_{Onset} \quad (19)$$

$$QRS_{Duration} = QRS_{Offset} - QRS_{Onset} \quad (20)$$

$$T_{Duration} = T_{Offset} - T_{Onset} \quad (21)$$

$$P_{Amplitude} = P_{Peak} - Signal_{Baseline} \quad (22)$$

$$QRS_{Amplitude} = R_{Peak} - Signal_{Baseline} \quad (23)$$

$$T_{Amplitude} = T_{Peak} - Signal_{Baseline} \quad (24)$$

$$QT_{Interval} = T_{Offset} - QRS_{Onset} \quad (25)$$

For the signal to be deemed valid, all measurements must fall within the bounds listed in **Table 3** which are clinically defined ranges for normal ECG beats [60].

Measurement	Lower Bound	Upper Bound
P Duration (seconds)	0.02	0.4
QRS Duration (seconds)	0.01	0.3
T Duration (seconds)	0.08	0.5
P Amplitude (mV)	0.05	0.5
QRS Amplitude (mV)	0.06	2
T Amplitude (mV)	0.01	0.9
QT Interval (seconds)	0.15	0.66

TABLE 3 UPPER AND LOWER BOUNDS TO VALIDATE ECG SIGNALS.

Examples of clean signals generated for each different HR can be seen in **Figure 20** below:

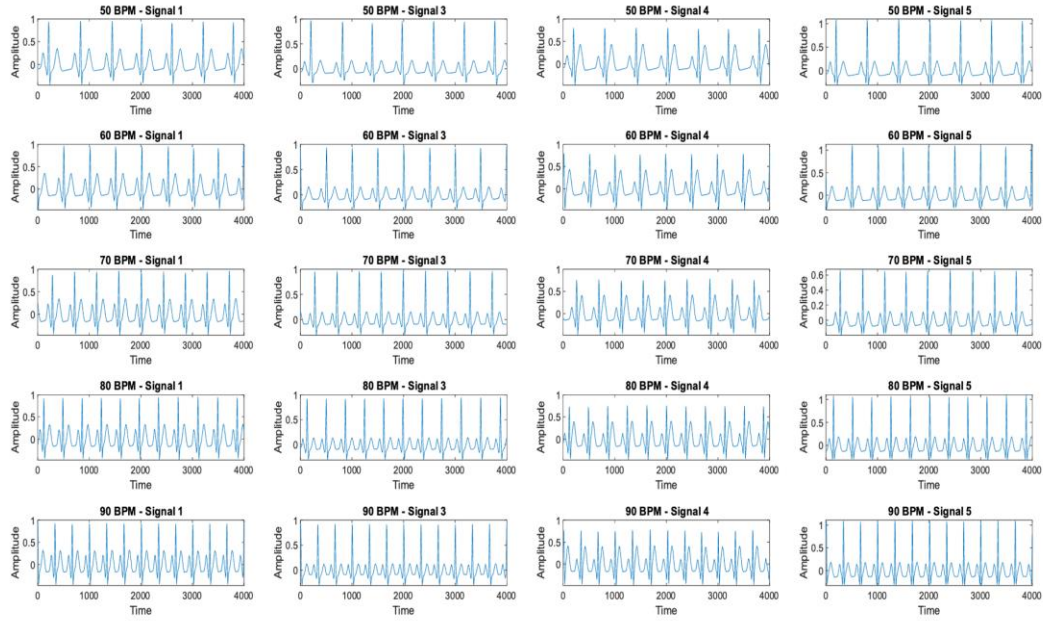


FIGURE 20 EXAMPLE CLEAN SIGNALS GENERATED.

4.2 CORRUPTING CLEAN ECGs WITH NOISE

The final step of the database generation involved corrupting each clean ECG signal with various amounts and types of noise. This involved extracting 30 second segments from the noise signal mentioned in **Section 4.1.1** and employing an auto-regressive (AR) model to predict ten new noise signals with similar statistical properties. Finally, the extracted and predicted noise signals were scaled to 0dB, 6dB, 12dB, 18dB and 24dB and linearly added to each clean ECG signal.

4.2.1 NOISE MODELLING

4.2.1.1 AUTO-REGRESSIVE MODELLING

To expand the dataset, auto-regressive (AR) modelling was employed to predict new characteristic noise signals with the same time-frequency content, meaning they are still representative of EM noise. In general, AR models are a class of ML models that automatically predict the next component of a sequence by taking measurements from previous inputs in the sequence. This means the predicted sequence will be a function of the past input sequence. AR is a common technique used in Data Augmentation (DA) where researchers have limited data available and need to expand the database, while maintaining data integrity [61].

AR models are trained on the data that is available, in which it learns the underlying distribution of the data by analysing how the current value in a sequence depends on its' preceding values. Like NN's, the model learns parameters (like weights) that define how past data points influence future ones. These parameters are adjusted to minimize prediction errors on the training set.

Once the AR model is trained, it can be used to generate new data. Starting with an initial small sequence, the model predicts the next data point, this prediction is then appended to the sequence and the process continues, eventually generating a sequence that resembles the training set. This process can be repeated to generate as many new sequences as needed, effectively creating synthetic data that augments the original dataset.

This approach is valid due to its' ability to capture temporal dependencies and maintain statistical properties of the original dataset.

The general form of an AR model of order p is given by:

$$X_t = c + \sum_{i=1}^p \Phi_i X_{t-i} + E_t \quad (26)$$

Where X_t is the value of the time series at time t, c is a constant, Φ_i are the coefficients of the model which represent the influence of the past p values X_{t-1} on the current value X_t , p is the order of the model, which indicates how many past values are used to predict the current value and E_t is the noise.

For example, if p = 1, the model uses only the immediately preceding value to predict the current value and the equation becomes:

$$X_t = c + \Phi_1 X_{t-1} + E_t \quad (27)$$

If p = 2, the model uses the two most recent past values to predict the current value and the equation becomes:

$$X_t = c + \Phi_1 X_{t-1} + \Phi_2 X_{t-2} + E_t \quad (28)$$

This model order relationship continues, and for a given dataset, there exists an optimal model order that provides the best predictive performance.

4.2.1.2 AR MODELLING APPLIED TO NOISE SIGNALS

Section 4.2.2.1 gives an overview on the mathematical principles behind AR models. This section will provide details on how the AR models were applied to the noise signal in this experiment and ultimately, added to the clean signals to return a noisy ECG signal.

Following the extraction of the 30-minute noise signal detailed in **Section 4.1.1**, ten segments were extracted from the 30-minute file each with a length of 30-seconds. As such, only 5-minutes from the original file were used, the reason for this is to reduce the computational complexity involved when scaling the database using AR modelling. Due to both hardware constraints, storage limitations and time constraints, 5-minutes was deemed to be sufficient.

The mean of each noise segment is obtained and subtracted from the original signal to obtain the zero-mean signal. The AR model coefficients are then obtained using Burge's method which fits an AR model to the input data by minimising the forward and backward prediction errors while constraining the AR parameters to satisfy the Levinson-Durbin recursion [58]. Random Gaussian noise signals are then generated which are used to generate new noisy signals with a similar Power Spectral Density (PSD) to the original noise signal. Finally, the initial

state of the filter is obtained, and the AR model is applied to the input signal to generate a new noisy copy. The result is re-scaled by the square root of the variance and the mean of the original signal is added back. This produces new noisy signals that have a similar PSD to the original signal. To ensure no hardware or storage constraints are exceeded, ten copies of noise per segment are generated, ultimately this meant that there were 11 (original and 10 copies) noise signals for each of the 10 segments, giving 110 unique noise signals per clean ECG signal.

4.2.1.3 CORRUPTING CLEAN SIGNALS WITH EM NOISE

The final step in the database generation was the corruption of clean ECG signals with relative amounts of the noise copies generated in **Section 4.2.1.2**. Five Signal to Noise Ratio's (SNRs) were selected for this study, these were 0dB, 6dB, 12dB, 18dB and 24dB. The SNR represents the amount of the amount of noise relative to signal and can be expressed mathematically as:

$$SNR = 10\log\left(\frac{S}{N}\right) \quad (29)$$

Where S is the signal vector in Volts and N is the noise vector in Volts.

For each clean ECG signal, the mean peak to peak amplitude was calculated where the peak-to-peak amplitude refers to the difference between the minimum and maximum locations within the QRS complex. The peak-to-peak amplitude was then converted to a power through the following transformation:

$$PeakToPeakPower = \frac{PeakToPeakAmplitude^2}{8} \quad (30)$$

This transformation is based off sinusoidal signals [59] and is necessary to estimate SNR levels. Following the transformation to clean signal power, the noise signal power is computed and used to determine a scaling factor through:

$$scalingFactor = \sqrt{\frac{PeakToPeakPower}{noisePower \times 10^{\frac{desiredSNR}{10}}}} \quad (31)$$

Where *desiredSNR* is the specified SNR level in decibels (IE 0dB), and noisePower is the Root Mean Square (RMS) of the noise signal squared.

The noise signal is then scaled to all SNR levels through:

$$noiseSignal_{scaled} = noiseSignal * scaleFactor \quad (32)$$

And the clean ECG signal is corrupt through:

$$ECG_{Noisy} = cleanSignal + noiseSignal_{scaled} \quad (33)$$

This method enables a means to scale a number of noisy signals to desired SNR levels and corrupt the original ECG signal. After this method is followed, each clean ECG signal has 110 noise corrupt versions at each SNR level (IE 110 X SNR0, 110 SNR6 etc)

4.3 DEEP LEARNING MODELS

4.3.1 REGION BASED CONVOLUTIONAL NEURAL NETWORK

The Region Based Convolutional Neural Network (RCNN) proposed by Brophy et al [54] was trained on the improved database. The network consists of 4 1-D convolutional layers with batch normalisation and ReLu followed by a fully connected layer. The network architecture can be seen in **Figure 21** below:

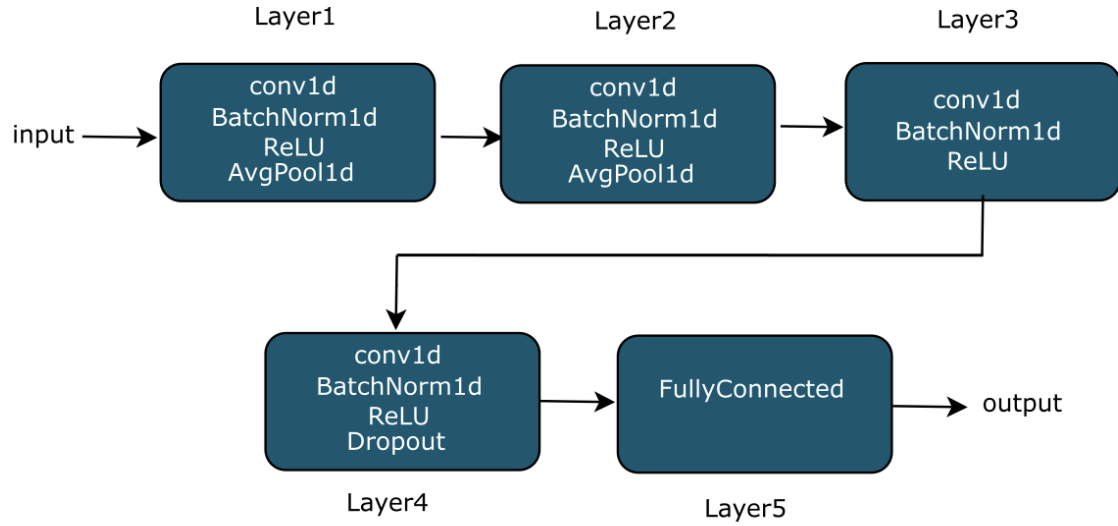


FIGURE 21 NETWORK ARCHITECTURE FOR RCNN [51].

4.3.2 CONVOLUTIONAL DENOISING AUTO ENCODER

The convolutional denoising autoencoder proposed by Chiang et al [64] was implemented and trained on the improved database. The Encoder consisted of 5 layers each with convolutional layers, batch normalisation and ReLu activation function. The decoder again consisted of 5 layers with convolutional layers and both ReLu and Sigmoid activation functions. The architecture is show below in **Figure 22**:

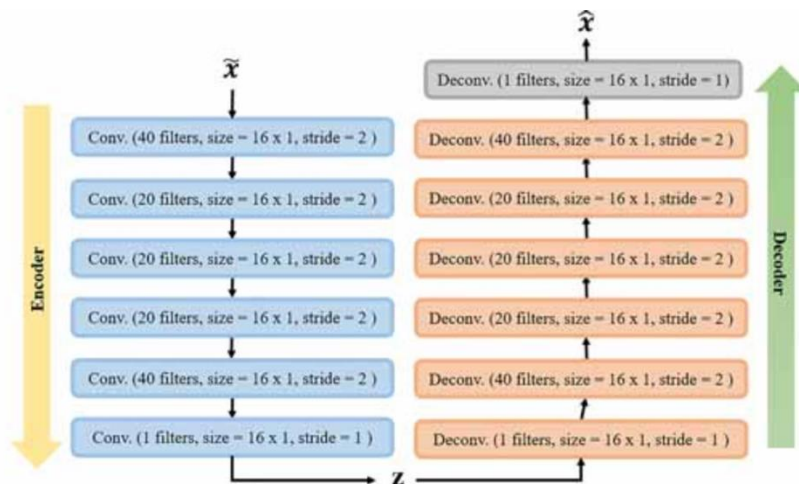


FIGURE 22 NETWORK ARCHITECTURE FOR CDAE.

4.3.3 DEEP RECURRENT NEURAL NETWORK

The deep recurrent neural network proposed by antczak et al [65] was implemented and trained on the improved database. The NN consisted of 64 LSTM units followed by 2 ReLU layers with 64 units and a single liner layer. The structure can be seen inf **Figure 23** below:

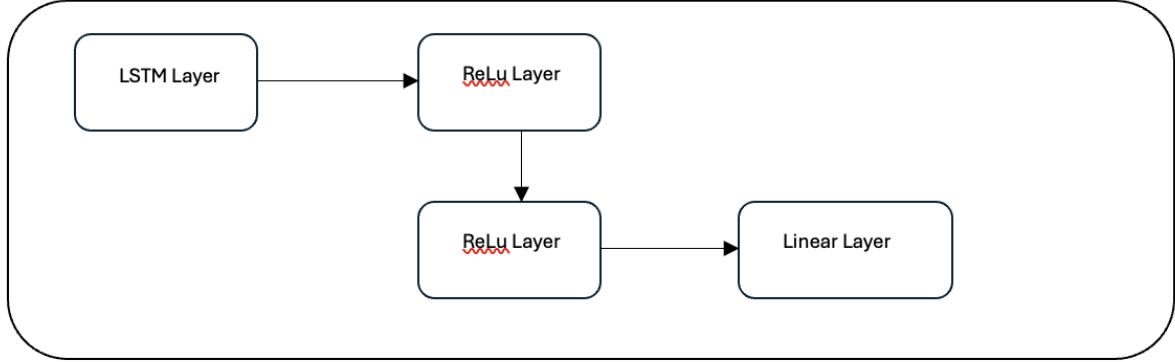


FIGURE 23 NETWORK ARCHETECTURE OF DRNN.

4.3.4 LOSS FUNCTION

The loss function used in all networks was employed to capture areas of each ECG cycle that contained useful information. The loss was computed around the PQRST complex only by utilising the following equations.

$$L = MSE(X_N, X) + \alpha * \sum_{i=1}^j MSE(RX_n i, R_{Xi}) \quad (34)$$

Where j is the total number of QRS peaks, R_{Xi} is the i^{th} QRS complex in the signal X and $RX_n i$ is the i^{th} QRS complex in the signal X_n .

4.4 PERFORMANCE METRICS

Standard performance metrics are used to compare the denoised ECG signals to the clean reference signal. Three metrics were chosen for their combined ability to capture the similarity of the denoised signal to the clean signal, these were the RMSE, NCC and SNR improvement.

The RMSE captures the global error between the two signals, mathematically it can be expressed as:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - y'_i)^2} \quad (35)$$

Where y_i is the actual value (clean signal) at sample I, y'_i is the predicted (denoised) value at sample I and n is the number of samples.

The NCC captures the phase relationship and similarity of the two signals, it can be expressed as:

$$NCC = \frac{1}{n} \sum_{n=1}^n \frac{(y-x)(z-m)}{\sigma_y \sigma_z} \quad (36)$$

Where y is the clean signal, z is the denoised signal, x is the mean of the clean signal, m is the mean of the denoised signal and σ_y σ_z are the standard deviation of the clean and denoised signal respectively.

The SNR improvement simply shows the amount of noise that has been removed from the noise corrupt signal. It can be expressed as:

$$SNR_{imp} = 10 \log_{10} \frac{SNR_{processed}}{SNR_{noisy}} \quad (37)$$

Two additional metrics were employed to validate the performance of the synthetic ECG signals generated. It was important to show that both time and frequency domain characteristics are similar and thus, the standard deviation and power spectral density (PSD) were used.

Standard deviation measures the amount of variation or dispersion in the ECG signals and is given by:

$$SD = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - y'_i)^2} \quad (38)$$

Where y' is the mean of the signal y .

PSD provides insights into the signal's frequency content and helps assess how much noise resides in different frequency bands. It can be expressed as:

$$PSD(f) = \frac{|FFT(y)|^2}{T} \quad (39)$$

Where T is the duration of the signal.

5 IMPLEMENTATION

A range of software tools were employed in this project to facilitate: 1) Project management, 2) Database generation, 3) Data augmentation, 4) Data storage, 5) Algorithm development (deep learning), and 6) GPU access.

Git was used for project management, with the graphical user interface (GUI) SourceTree employed for more efficient version control of code and documents. Due to storage limitations, Git was not used for database versioning.

MATLAB 2023b was utilized to generate the reference database. MATLAB offers an efficient platform for generating time-series signals undergoing various transformations, and its built-in toolboxes include digital signal processing algorithms relevant to this study. MATLAB was also used for data augmentation, with all simulations and autoregressive modelling performed using open-source MATLAB functions.

Initially, Google Drive was chosen as the cloud storage platform due to its free access and ability to interface with NVIDIA GPUs, which are crucial for deep learning development. However, corruption issues when writing HDF5 files from MATLAB to Google Drive necessitated a shift to local storage. The files were compressed into .ZIP format and stored locally, imposing constraints on the volume of data generated, but this approach was deemed necessary under time constraints.

For deep learning implementation, Python with the PyTorch framework was employed. PyTorch accelerates the development process by providing pre-built deep learning algorithms that can be easily invoked, while also offering flexibility to modify key model parameters, such as learning rate, batch size, and network architecture.

The overall workflow of the framework can be seen below in **Figure 24**.

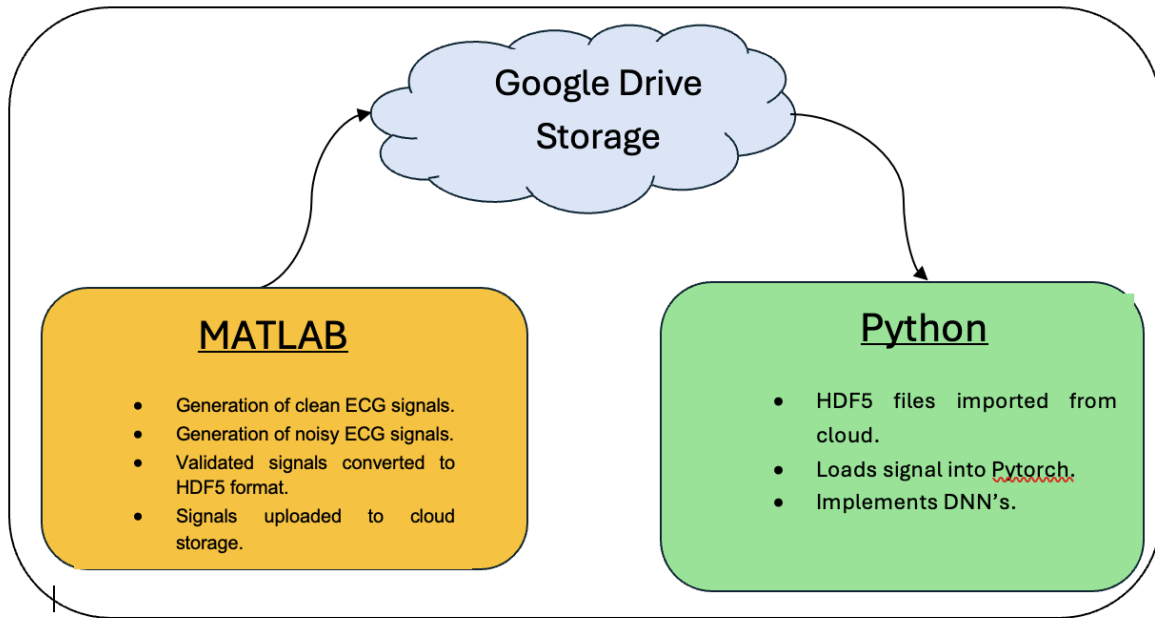


FIGURE 24 OVERALL FRAMEWORK WORKFLOW.

Access to GPUs in this project presented significant challenges. Initially, cloud platforms such as Google Colab were utilized to access NVIDIA Tesla V100 GPUs. However, the limitations of the free version prevented the completion of training, making these solutions viable only for smaller networks. The Metal Performance Shaders (MPS) framework offers a highly optimized interface for GPU utilization on MacBook systems, providing a powerful tool for machine learning and deep learning tasks. By leveraging the MPS framework, significant acceleration in training and inference of neural networks is achieved by harnessing the computational capabilities of the MacBook's GPU. Notably, this framework is integrated with PyTorch, facilitating ease of setup. While the GPU on MacBook systems offers a considerable speed improvement over Central Processing Units (CPUs), it remains notably slower than NVIDIA GPU systems.

Reflecting on the progress of this research, there has been a substantial alignment with the initial project plan, albeit with several necessary adjustments due to unforeseen complexities. In the early stages, the development of the reference database and noise extraction proceeded as scheduled, and the implementation of the DNNs followed the outlined plan closely. However, during training and validation of the deep learning models it became evident that the variability in noise characteristics required a more complex approach to data augmentation, leading to the incorporation of AR modelling. This adjustment significantly improved performance.

One notable challenge was the computational limitations encountered when training large networks. This led to a shift from using cloud-based GPUs to local GPU acceleration via the MPS framework, an adaption that was not initially planned. This shift required some time to setup, however resulted in gaining control over training

processed and reduced dependence on external resources. Overall, the research has not only adhered to the key milestones but has demonstrated a capacity to adapt and improve upon the original methodologies.

6 RESULTS AND DISCUSSION

The following section shows results from each DNN model on the generated database. A comparison of these results to traditional time-frequency based filters is made. Any implications surrounding the results is discussed and limitations are implementation of the algorithms in a real world setting are covered.

Note: 70% of the database was used to train the algorithms, and 30% was used to validate their performance.

6.1 DATABASE PERFORMANCE

To first show that the clean ECG signals being generated only contained features that were realistic, the percentage of signals that contained a P wave duration, QRS duration, T wave duration, P wave amplitude, QRS amplitude, T wave amplitude and QT interval within the specified range mentioned in **Table 3** was calculated. As is evident from **Table 4** and **Table 5** below, all clean signals used in this study were deemed suitable for further processing.

6.1.1 CLEAN SIGNAL FEATURE VALIDATION

Heart Rate (BPM)	P Duration (%)	QRS Duration (%)	T Duration (%)	QT Interval (%)
50	100	100	100	100
60	100	100	100	100
70	100	100	100	100
80	100	100	100	100
90	100	100	100	100
100	100	100	100	100

TABLE 4 PERFORMANCE OF CLEAN SIGNAL GENERATION (DURATION).

Heart Rate (BPM)	P Amplitude (%)	QRS Amplitude (%)	T Amplitude (%)
50	100	100	100
60	100	100	100
70	100	100	100
80	100	100	100
90	100	100	100
100	100	100	100

TABLE 5 PERFORMANCE OF CLEAN SIGNAL GENERATION (AMPLITUDE)

6.1.2 COMPARISON WITH STANDARD ECG DATABASE.

Table 6 shows a statistical comparison of the generated clean signals to that of the MIT-BIH Arrhythmia database. Results show that similar statistical properties indicate valid ECG signals.

Database	Standard Deviation (mV)	PSD Correlation
Generated Reference database	0.33	0.87
MIT-BIH Arrhythmia	0.34	

TABLE 6 SHOWS THE STATISTICAL PROPERTIES OF THE GENERATED DATABASE COMPARED TO MIT-BIH ARRHYTHMIA

The mean standard deviation of both databases was computed, the generated database returned a mean standard deviation of **0.33mV** whereas the real ECG database returned **0.34mV**. This difference of **2.94%** was deemed tolerable and provided evidence that the synthetic ECG signals have similar temporal components to real signals.

A high PSD correlation of **0.87** provided evidence that the frequency content of both databases was similar. Since both frequency domain statistical properties and time domain statistical properties were similar, the clean signal generation can be considered a valid technique.

1.1.1 NOISE SIGNAL VALIDATION

The statistical properties of the generated noise signals compared to statistical properties from the MIT-BIH Arrhythmia database with added noise is shown in **Table 7**.

Database	SNR	Standard Deviation (mV)
Generated Reference database	0	2.20
	6	1.65
	12	1.22
	18	0.91
	24	0.39
MIT-BIH Arrhythmia	0	2.41
	6	1.72
	12	1.40
	18	0.88
	24	0.42

TABLE 7 STATISTICAL PROPERTIES OF NOISE SIGNALS GENERATED.

As is evident, noise signals generated in this experiment inhibit similar standard deviations to EM noise signals used in similar experiments, validating their use for training. Furthermore, high PSD correlation values for each SNR level at **0.82, 0.82, 0.84, 0.79** and **0.86** respectively was returned. Both metrics provide evidence that the AR noise signals inhibit similar time and frequency properties to that of real-world noisy ECG signals.

6.2 DNN PERFORMANCE

Table 8 below shows the RMSE, NCC and SNR improvement of each DNN at each SNR level. It is evident that the RCNN performs the best across all SNR levels.

RCNN					
SNR (dB)	RMSE (Noise)	RMSE (Processed)	NCC (Noise)	NCC (Processed)	SNR Improvement (dB)
0	0.24	0.17	0.32	0.48	2.33
6	0.23	0.14	0.38	0.50	2.59
12	0.22	0.07	0.49	0.82	4.19
18	0.17	0.11	0.56	0.90	5.31
24	0.12	0.07	0.58	0.92	3.45

CDAE					
SNR (dB)	RMSE (Noise)	RMSE (Processed)	NCC (Noise)	NCC (Processed)	SNR Improvement
0	0.24	0.27	0.32	0.32	-0.15
6	0.23	0.22	0.38	0.48	1.58
12	0.22	0.20	0.49	0.61	1.18
18	0.17	0.14	0.56	0.72	3.12
24	0.12	0.09	0.58	0.81	2.23

RNN					
SNR (dB)	RMSE (Noise)	RMSE (Processed)	NCC (Noise)	NCC (Processed)	SNR Improvement
0	0.24	0.25	0.32	0.48	-0.04
6	0.23	0.20	0.38	0.50	0.80
12	0.22	0.17	0.49	0.82	1.12
18	0.17	0.11	0.56	0.90	1.45
24	0.12	0.07	0.58	0.92	1.88

TABLE 8 RESULTS FROM EACH DNN ON TEST DATA.

Significant denoising effects are highlighted green to easier visualise the results. It can be seen that the RCNN performs positively across all SNR levels, however it should be noted that at 0dB, while positive the performance is still low with a low correlation value of 0.48, indicating a low similarity between the denoised and clean signal. The RCNN performed well across SNR levels 24, 18 and 12dB however at noise levels exceeding 12dB, the algorithm returned negligible performance difference between the denoised and noisy signal. At 0dB, the RMSE value actually increased in the processed signal. CDAE performed the worst, however was still able to have a positive impact on the noise signal at 24dB and 18dB.

Figure 25, 26, 27, 28 and 29 show visual examples of the denoised signals alongside the clean and noisy signal for the RCNN model for each SNR level.

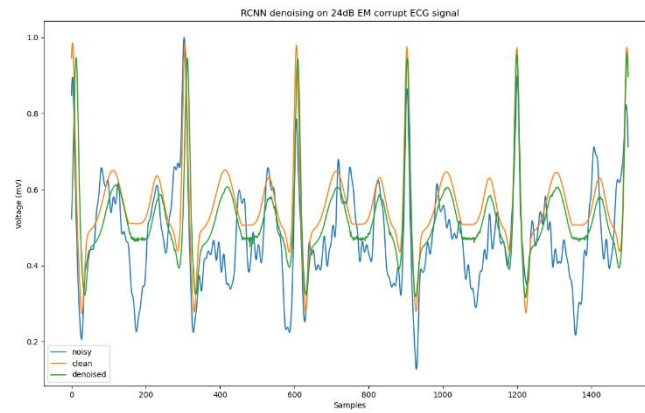


FIGURE 25 RCNN DENOISED SIGNAL AT 24DB.

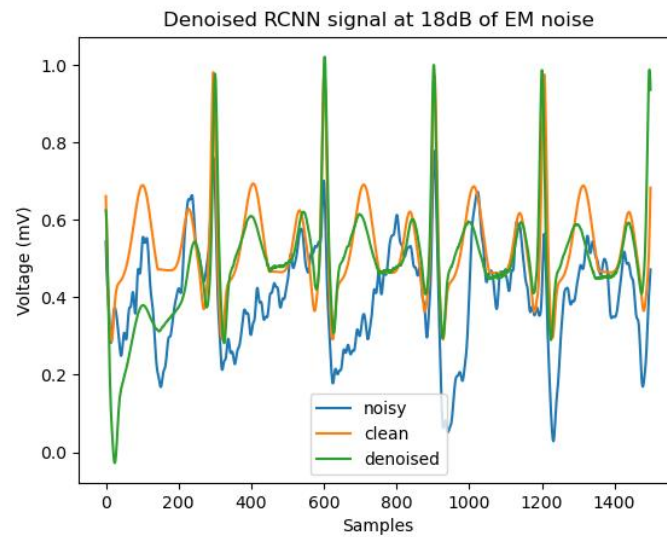


FIGURE 26 RCNN DENOISED SIGNAL AT 18DB.

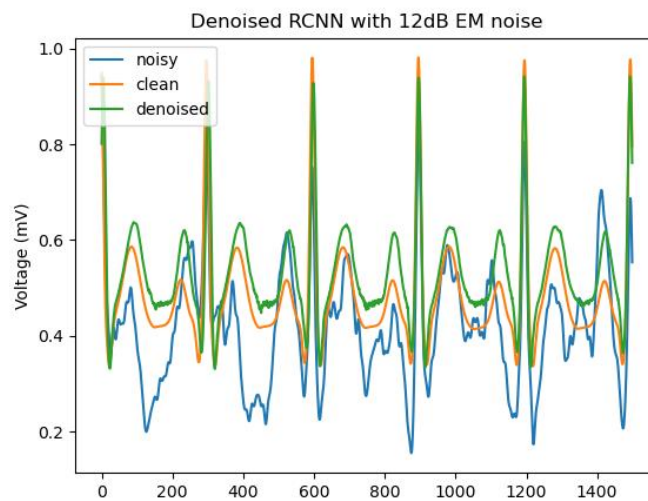


FIGURE 27 RCNN DENOISED SIGNAL AT 12DB.

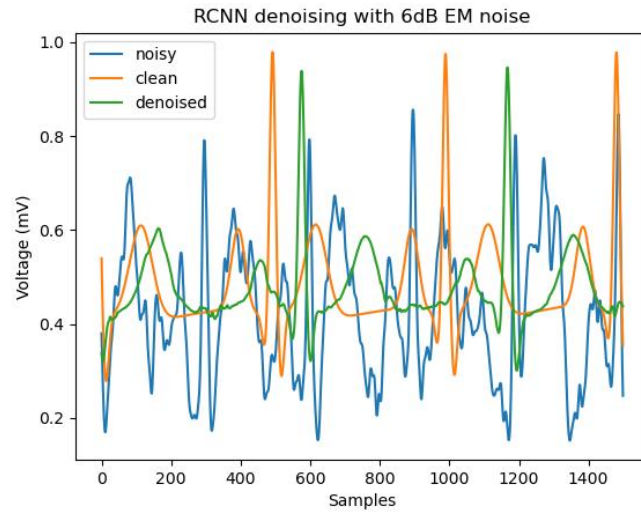


FIGURE 28 RCNN DENOISING AT 6DB.

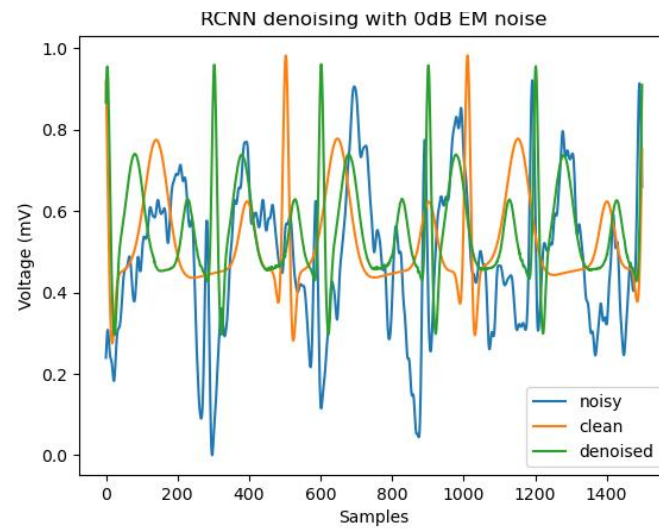


FIGURE 29 RCNN DENOISING AT 0DB.

It is evident at the lower SNR levels (0dB and 6dB), the model can incorrectly predict QRS causing a large difference between the signals and subsequently, poor performance metrics. The large spikes in noise at these levels is likely the cause, tricking the model into thinking that a QRS peak was present before noise corruption. This behaviour was present on all three models.

6.3 COMPARISON TO TRADITIONAL FILTERS

The noisy signals were filters with EMD and wavelet methods and the RMSE, NCC and SNR improvement is shown below in **Table 9**.

Wavelet					
SNR (dB)	RMSE (Noise)	RMSE (Processed)	NCC (Noise)	NCC (Processed)	SNR Improvement
0	0.24	0.22	0.32	0.33	0.46
6	0.23	0.19	0.38	0.39	1.42
12	0.22	0.18	0.49	0.83	3.78
18	0.17	0.14	0.56	0.86	3.88
24	0.12	0.09	0.58	0.89	3.23

EMD					
SNR (dB)	RMSE (Noise)	RMSE (Processed)	NCC (Noise)	NCC (Processed)	SNR Improvement
0	0.24	0.20	0.32	0.38	0.98
6	0.23	0.18	0.38	0.44	2.10
12	0.22	0.16	0.49	0.77	3.92
18	0.17	0.13	0.56	0.90	5.26
24	0.12	0.07	0.58	0.92	3.52

TABLE 9 PERFORMANCE OF WAVELET AND EMD FILTERS.

It can be seen that EMD generally performs better than the wavelet method for removing EM noise from ECG signals, with improved metrics across all SNR levels.

Comparing **Table 8** with **Table 9**, it is evident that the RCNN model outperforms both traditional methods of removing EM noise from ECG signals across all SNR levels.

6.4 ANALYSIS OF RESULTS

Results in Section 6.3 indicate that DNN's have the potential to outperform current gold standard techniques for EM removal (EMD and wavelet). While the CDAE and DRNN models did not perform as well as either traditional methods, the cause of this is likely caused by inadequate tuning of the networks. Further optimisation of these networks may increase performance beyond what is seen in this report. Looking at the two best performing models (RCNN and EMD) it can be seen that the RCNN outperforms EMD in RMSE across all SNR levels with a reduction in RMSE of **15%, 22%, 56% 7% and 0%**. The NCC values increased in the RCNN model by **21%, 12%, 6%, 0% and 0%**. The SNR improvement increased in the RCNN by **58%, 19%, 6%, 1% and 1%** relative to EMD. It is evident that the difference in filters is negligible at the lower noise levels (18dB and 24dB) and DL may only offer a real benefit as noise levels increase.

All filters provide a denoised signal that is better than the noise signals, with improvements up to **68%** reduction in RMSE obtained by the RCNN at 12dB noise. It should be highlighted that the CDAE and RNN models removed little to no noise at 0dB and in fact, made the signal worse in some cases.

Overall, the results suggest that deep learning models offer a promising alternative to conventional methods, providing superior denoising capabilities, particularly in moderate to low-noise environments. The RCNN's ability to outperform traditional methods across key metrics like RMSE, NCC, and SNR improvement indicates

that it can preserve the integrity of ECG signals while significantly reducing EM noise. Nonetheless, future work may explore the use of hybrid models or novel architectures to further improve denoising performance in high-noise scenarios.

6.5 IMPLICATIONS AND APPLICATIONS

The implications of these results are significant for both clinical applications and broader healthcare technology. The development of deep learning models that can effectively reduce EM noise in ECG signals can directly enhance the accuracy of ECG-based diagnoses, particularly in ambulatory and remote monitoring settings where noise contamination is common. This improvement would be especially impactful in long-term cardiac monitoring, where wearable devices frequently encounter motion-induced noise. By reducing noise, these models can help avoid misdiagnoses and ensure more reliable monitoring of heart conditions like arrhythmias, ischemic heart disease, and myocardial infarctions.

Beyond immediate clinical applications, this work demonstrates the broader potential of machine learning in biomedical signal processing, paving the way for its application to other noisy physiological signals such as electroencephalograms (EEGs) and electromyograms (EMGs). Moreover, the models developed in this study could be integrated into real-time monitoring systems, enhancing the capabilities of wearable devices and telemedicine platforms by providing continuous, accurate, and non-invasive diagnostics without the need for repeated measurements or manual intervention.

From a technical perspective, the approach also highlights the potential of deep learning models in signal denoising across various engineering and scientific domains. The methodologies used in this research can be adapted for other types of time-series data that suffer from noise contamination, such as environmental monitoring, industrial process control, and financial forecasting.

7 CONCLUSION

The research conducted in this thesis explored the application of advanced machine learning algorithms for the mitigation of electrode motion (EM) artifacts in electrocardiogram (ECG) signals. Through the development of a synthetic ECG database and the comparison of multiple deep learning architectures, several key conclusions have been drawn. Firstly, the superiority of deep learning models such as CNN's significantly outperforms traditional time-frequency based methods such as EMD and WT in reducing EM noise from ECG signals. CNN's offered a reduction in RMSE of up to 56%, an increase in NCC of up to 21% and an increase in SNR improvement by up to 56% compared to the best performing traditional method.

The importance of training data is highlighted in this study, where the creation of a diverse, expansive ECG database that incorporates a wide range of heart rates, signal morphologies and noise conditions played a crucial role in enabling models to generalise and perform well across varying levels of noise. The success of the denoising algorithms compared to reported results from literature highlights how this study has performed a benchmark for training signal conditioning algorithms for optimal performance.

Deep learning models show promise not only for ECG signal processing but also for broader biomedical signal applications, especially where noise reduction is a challenge. The demonstrated flexibility of the models suggests potential use in various real-world scenarios where patient movement or external conditions affect the quality of ECG readings.

Although the CDAE and RNN models did not perform well in this experiment, a likely cause is the network optimisation and with further research, these could offer promising solutions for noise removal in ECG signals.

Future research could build upon this work by exploring more advanced techniques for both data generation and noise reduction. Specifically, the integration of Physics-Informed Neural Networks (PINNs) and Generative AI could provide groundbreaking advancements in the denoising process and in the overall analysis of ECG signals.

PINNs can be integrated into the ECG signal denoising process by embedding the physical laws governing cardiac electrophysiology into the model architecture. This could enable the network to better differentiate between physiological ECG patterns and noise, potentially reducing the need for large amounts of training data. By incorporating known differential equations related to heart function, PINNs can enhance the model's ability to generalize, leading to improved accuracy when dealing with EM noise, especially in complex, real-world conditions. This would allow the model to enforce physiological constraints, thus improving the interpretability and robustness of the outputs.

Generative Adversarial Networks (GANs) could be used to generate more realistic ECG signals for training models, particularly in scenarios where large, diverse datasets are unavailable. By generating synthetic ECG signals that mimic real-world variations and noise conditions, GANs could improve the robustness of denoising models. A GAN framework can learn the distribution of clean ECG signals and EM-corrupted signals, allowing it to generate highly realistic clean signals even when trained on noisy datasets. This could significantly enhance the capability of the denoising models, particularly in edge cases where traditional noise reduction methods struggle.

Finally, this report only investigated the performance of the algorithms on the database generated in this work. To further explore the benefits of using this database, the algorithms should be validated on real life signals. Specifically, the databases used to validate each model in previous work would be used and a comparison could be made.

Note : All development code used in this work can be located on GitHub [66].

8 REFERENCES

- [1] Khater, H.M. and Suliman, A., 2023. *Deep Learning-Based ECG Analysis for Myocardial Infarction Detection*.
- [2] Arooj, M., Ul-Haq, I., Ansari, S.A., Zeb, A. and Zaman, S., 2022. *An overview of electrocardiography (ECG) signals processing and classification for arrhythmias detection*. *Journal of Intelligent & Fuzzy Systems*, 43(5), pp.4945-4960.
- [3] Yu, K., Feng, L., Chen, Y., Wu, M., Zhang, Y., Zhu, P., Chen, W., Wu, Q., and Hao, J., 2023. *Accurate wavelet thresholding method for ECG signals*. *Computers in Biology and Medicine*
- [4] Abbaspour, S., Fallah, A. and Yaghmaee, F., 2021. *A new ECG denoising structure based on an improved hybrid neural network model*. *IET Signal Processing*, 15(5), pp. 337-348
- [5] Saini, S., Singh, N., Khosla, A. and Kaur, H., 2020. *A novel method for muscle artifact removal from single-channel ECG signal using variational mode decomposition*. *Biomedical Engineering Online*, 19(1), pp.1-18.
- [6] Kalra, A., Anand, G., Lowe, A., Simpkin, R., and Budgett, D., 2024. *A smart idea to reject motion artefacts from ECG measurements due to sensor-body impedance*
- [7] HM Government. (2021). *Net Zero Strategy: Build Back Greener*. Department for Business, Energy & Industrial Strategy. Available at: <https://www.gov.uk/government/publications/net-zero-strategy>
- [8] HM Government. (2017). *The Clean Growth Strategy: Leading the way to a low carbon future*. Department for Business, Energy & Industrial Strategy. Available at: <https://www.gov.uk/government/publications/clean-growth-strategy>
- [9] NHS England. (2019). *The NHS Long Term Plan*. Available at: <https://www.longtermplan.nhs.uk/>
- [10] UK Government. (2012). *Health and Social Care Act 2012*. Available at: <https://www.legislation.gov.uk/ukpga/2012/7/contents/enacted>
- [11] Oppenheim, A. V., Willsky, A. S., & Nawab, S. H. (1996). *Signals and Systems* (2nd ed.). Prentice Hall.
- [12] Proakis, J. G., & Manolakis, D. G. (1996). *Digital Signal Processing: Principles, Algorithms, and Applications* (3rd ed.). Prentice Hall.
- [13] Cooley, J. W., & Tukey, J. W. (1965). *An algorithm for the machine calculation of complex Fourier series*. *Mathematics of Computation*, 19(90), 297–301.
- [14] Oppenheim, A. V., & Schafer, R. W. (2009). *Discrete-Time Signal Processing* (3rd ed.). Prentice Hall.
- [15] Mallat, S. (1999). *A Wavelet Tour of Signal Processing* (2nd ed.). Academic Press.
- [16] Huang, N. E., Shen, Z., Long, S. R., Wu, M. C., Shih, H. H., Zheng, Q., ... & Liu, H. H. (1998). *The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis*. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 454(1971), 903-995.
- [17] Romero, I., et al. (2011). *Motion artifact reduction in ambulatory ECG monitoring: An integrated system approach*. In **Proceedings of the 2nd Conference on Wireless Health**, San Diego, California.

- [18] Sörnmo, L., & Laguna, P. (2005). *Bioelectrical Signal Processing in Cardiac and Neurological Applications*. Academic Press.
- [19] **Tam, H. T., & Webster, J. G. (1977).** *Minimizing Electrode Motion Artifact by Skin Abrasion*. IEEE Transactions on Biomedical Engineering, BME-24(2), 134-139.
- [20] **Edelberg, R. (1968).** *Electrical properties of the skin*. In C. C. Brown (Ed.), **Methods in Psychophysiology** (pp. 1-53). Baltimore: Williams & Wilkins.
- [21] An X., Stylios G.K. Comparison of Motion Artefact Reduction Methods and the Implementation of Adaptive Motion Artefact Reduction in Wearable Electrocardiogram Monitoring. *Sensors*. 2020;**20**:1468.
- [22] Łęski J.M., Henzel N. ECG baseline wander and powerline interference reduction using nonlinear filter bank. *Signal Process*. 2005;**85**:781–793.
- [23] Chouhan V.S., Mehta S.S. Total removal of baseline drift from ECG signal; Proceedings of the International Conference on Computing: Theory and Applications (ICCTA'07); Kolkata, India. 5–7 March 2007; pp. 512–515.
- [24] Hao W., Chen Y., Xin Y. ECG baseline wander correction by mean-median filter and discrete wavelet transform; Proceedings of the 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society; Boston, MA, USA. 30 August 2011–3 September 2011; pp. 2712–2715.
- [25] El-Dahshan E.-S.A. Genetic algorithm and wavelet hybrid scheme for ECG signal denoising. *Telecommun. Syst*. 2011;**46**:209–215. doi: 10.1007/s11235-010-9286-2.
- [26] Karthikeyan P., Murugappan M., Yaacob S. ECG signal denoising using wavelet thresholding techniques in human stress assessment. *Int. J. Electr. Eng*.
- [27] Singh B.N., Tiwari A.K. Optimal selection of wavelet basis function applied to ECG signal denoising. *Digit. Signal Process*. 2006;**16**:275–287.
- [28] Alfaouri M., Daqrouq K. ECG signal denoising by wavelet transform thresholding. *Am. J. Appl. Sci*. 2008;**5**:276–281.
- [29] Strasser F., Muma M., Zoubir A.M. Motion artifact removal in ECG signals using multi-resolution thresholding; Proceedings of the 20th European Signal Processing Conference (EUSIPCO); Bucharest, Romania. 27–31 August 2012; pp. 899–903.
- [30] Zhang D. Wavelet approach for ECG baseline wander correction and noise reduction; Proceedings of the 27th Annual Conference on Engineering in Medicine and Biology; Shanghai, China. 17–18 January 2006; pp. 1212–1215.
- [31] Blanco-Velasco M., Weng B., Barner K.E. ECG signal denoising and baseline wander correction based on the empirical mode decomposition. *Comput. Biol. Med*. 2008.
- [32] Pan N., Mang V., Un M.P., Hang P.S. Accurate removal of baseline wander in ECG using empirical mode decomposition; Proceedings of the 2007 Joint Meeting of the 6th International Symposium on Noninvasive Functional Source Imaging of the Brain and Heart and the International Conference on Functional Biomedical Imaging; Hangzhou, China. 12–14 October 2007; pp. 177–180.
- [33] Zhao Z.-D., Chen Y.-Q. A new method for removal of baseline wander and power line interference in ECG signals; Proceedings of the Machine Learning and Cybernetics; Dalian, China. 13–16 August 2006; pp. 4342–4347.

- [34] Kabir M.A., Shahnaz C. Denoising of ECG signals based on noise reduction algorithms in EMD and wavelet domains. *Biomed. Signal Process. Control.* 2012;**7**:481–489. doi: 10.1016/j.bspc.2011.11.003.
- [35] Thakor N.V., Zhu Y.S. Applications of adaptive filtering to ECG analysis: Noise cancellation and arrhythmia detection. *IEEE Trans. Biomed. Eng.* 1991;**38**:785–794. doi: 10.1109/10.83591.
- [36] Zhang Z., Silva I., Wu D., Zheng J., Wu H., Wang W. Adaptive motion artefact reduction in respiration and ECG signals for wearable healthcare monitoring systems. *Med. Biol. Eng. Comput.* 2014;**52**:1019–1030. doi: 10.1007/s11517-014-1201-7.
- [37] Iyer V., Ploysongsang Y., Ramamoorthy P. Adaptive filtering in biological signal processing. *Crit. Rev. Biomed. Eng.* 1990;**17**:531–584.
- [38] Correa A.G., Lacia E., Patino H.D., Valentinuzzi M.E. Artifact removal from EEG signals using adaptive filters in cascade. *J. Phys. Conf. Ser.* 2007;**90**:012081. doi: 10.1088/1742-6596/90/1/012081.
- [39] Romero I., Geng D., Berset T. Adaptive filtering in ECG denoising: A comparative study; *Proceedings of the 2012 Computing in Cardiology; Krakow, Poland. 9–12 September 2012; pp. 45–48.*
- [40] Liu S.-H. Motion artifact reduction in electrocardiogram using adaptive filter. *J. Med. Biol. Eng.* 2011;**31**:67–72. doi: 10.5405/jmbe.676.
- [41] Ko B., Lee T., Choi C., Kim Y., Park G., Kang K., Bae S., Shin K. Motion artifact reduction in electrocardiogram using adaptive filtering based on half cell potential monitoring; *Proceedings of the 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society; San Diego, CA, USA. 28 August 2012–1 September 2012; pp. 1590–1593.*
- [42] Clifford, G. D., Azuaje, F., & McSharry, P. E. (2006). *Advanced methods and tools for ECG data analysis.* Artech House
- [43] Abagaro, A.M., Barki, H., Ayana, G. *et al.* Automated ECG Signals Analysis for Cardiac Abnormality Detection and Classification. *J. Electr. Eng. Technol.* **19**, 3355–3371 (2024). <https://doi.org/10.1007/s42835-024-01902-y>
- [44] Rashidi, S., Asl, B.M. Strength of ensemble learning in automatic sleep stages classification using single-channel EEG and ECG signals. *Med Biol Eng Comput* **62**, 997–1015 (2024). <https://doi.org/10.1007/s11517-023-02980-2>
- [45] **Ling, Y., Zhang, Y., & Hu, S. (2019).** *Biomedical Signal Processing and Machine Learning for Cardiovascular Diseases.* Journal of Healthcare Engineering, 2019, 1-10. <https://doi.org/10.1155/2019/1910285>
- [46] **Goodfellow, I., Bengio, Y., & Courville, A. (2016).** *Deep Learning.* MIT Press.
- [47] Hannun et al. (2019) in "Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks"
- [48] Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation* (2nd ed.). Prentice Hall.
- [49] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention is All You Need.* Advances in Neural Information Processing Systems, 30.
- [50] Liu, Y., Zeng, X., Li, R., Wang, J., & Xie, H. (2021). *Challenges and opportunities in deep learning for biomedical data.* *Nature Reviews Bioengineering*, 3(1), 35-51.

- [51] Shickel, B., Tighe, P. J., Bihorac, A., & Rashidi, P. (2018). *Deep EHR: A survey of recent advances in deep learning techniques for electronic health record (EHR) analysis*. *IEEE Journal of Biomedical and Health Informatics*, 22(5), 1589-1604.
- [52] Samek, W., Wiegand, T., & Müller, K. R. (2017). *Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models*. *ITU Journal: ICT Discoveries*, 1(1), 1-10.
- [53] Jiang, F., Jiang, Y., Zhi, H., Dong, Y., Li, H., Ma, S., ... & Wang, Y. (2017). *Artificial intelligence in healthcare: past, present and future*. *Stroke and Vascular Neurology*, 2(4), 230-243.
- [54] Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., ... & Stanley, H. E. (2000). *PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals*. *Circulation*, 101(23), e215-e220. <https://doi.org/10.1161/01.CIR.101.23.e215>.
- [55] Brophy, Eoin & Hennelly, Bryan & de Vos, Maarten & Boylan, Geraldine & Ward, Tomas. (2022). Improved Electrode Motion Artefact Denoising in ECG Using Convolutional Neural Networks and a Custom Loss Function. *IEEE Access*. 10. 1-1. 10.1109/ACCESS.2022.3176971.
- [56] Moody GB, Mark RG. The impact of the MIT-BIH Arrhythmia Database. *IEEE Eng in Med and Biol* 20(3):45-50 (May-June 2001). (PMID: 11446209)
- [57] Moody GB, Muldrow WE, Mark RG. A noise stress test for arrhythmia detectors. *Computers in Cardiology* 1984; 11:381-384.
- [58] McSharry PE, Clifford GD, Tarassenko L, Smith L. A dynamical model for generating synthetic electrocardiogram signals. *IEEE Transactions on Biomedical Engineering* 50(3): 289-294; March 2003
- [59] Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding the Impact of Dataset Quality on Deep Learning Model Performance. *arXiv preprint arXiv:1611.03530*. Retrieved from <https://arxiv.org/abs/1611.03530>.
- [60] Wagner GS (2014), Marriott's Practical Electrocardiography.
- [61] **Shorten, C., & Khoshgoftaar, T. M. (2019).** *A survey on image data augmentation for deep learning*. *Journal of Big Data*, 6(1), 1-48. DOI: 10.1186/s40537-019-0197-0
- [62] **Hayes, M. H. (1996).** *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons.
- [63] **Oppenheim, A. V., & Schaffer, R. W. (1999).** *Discrete-Time Signal Processing* (2nd ed.). Prentice Hall.
- [64] **Chiang, W. Y., Yen, S. J., & Lee, Y. S. (2019).** *Denoising autoencoder with convolutional layers for domain adaptation*. *IEEE Access*, 7, 75043-75055. DOI: 10.1109/ACCESS.2019.2921279
- [65] **Antczak, K. (2019).** *Deep recurrent neural networks for ECG signal denoising*. *IEEE Transactions on Signal Processing*, 67(20), 5344-5354. DOI: 10.1109/TSP.2019.2940995
- [66] [MSc-Project/Electrode Motion Denoising at main · ben120-web/MSc-Project · GitHub](#)

Appendices

ALGORITHM 1: ALGORITHM TO CONSTRUCT CLEAN ECG SIGNALS.

Input:

HR_TO_GENERATE
MIN_ANGLES_OF_EXTREMA
MAX_ANGLES_OF_EXTREMA
MIN_Z_POSITION_OF_EXTRAMA
MAX_Z_POSITION_OF_EXTRAMA
MIN_GAUSSIAN_WIDTH
MAX_GAUSSIAN_WIDTH
NUM_OF_SIGNALS_PER_HR

Output: A database of clean ECG signals with different morphologies and Heart Rates.

```
1  Sampling //Generate 200 random sample parameters of our inputs.
2  For (iHeartRate = 1 : numberOfHeartRates) // Loop through all heart rates defined.
3      Define the HR to generate 200 signals for.
4      for (iSignal = 1 : numberOfSignalsPerHR) // Loop through each of the 200
        signals
5          Define parameter settings: Waveform angle, waveform amplitude,
            width size.
6          Try:
            Call ECGSYN function with specific parameter settings.
          Catch:
            Continue // If defined values are invalid for the function, skip.
          end
7          Validate : Validate that the waveform amplitudes and widths are
            within a realistic range.
8  Save : Save the parameter setting matrix and signals.
```

TABLE 8 ALGORITHM DESIGN TO GENERATE CLEAN SIGNAL DATABASE.

ALGORITHM 2 : ALGORITHM TO CONSTRUCT NOISY ECG SIGNALS.

Input: Path to noise signal
Path to clean ECG signals
Sampling frequency
ECG Length
Number of noise sections
SNR Levels
Number of generated noise signals.

Output: A database of noisy ECG signals with different morphologies and Heart Rates.

```
1 Load Noise Data □ Read in 30 minute noise signal
2 Segment the noise signal into 30 second strips.
3   for (iSegment = 1 : Number of noise sections)
4       Predict a new 30 second noise strip using auto-regressive modelling.
5       Save : Save each noise signal to a table on MATLAB.
6   for (iCleanEcgSignal = 1 : numberOfEcgSignal)
7       Load ECG Data □ Read in 30 second record.
8       Load QRS Locations □ Load in the QRS locations.
9       Calculate the peak to peak amplitude of QRS peaks.
10      Convert peak to peak amplitude to power
11      Scale the noise signal to each required SNR level.
12      Add the noise signal to the clean ECG record.
13      Save : Save the table which contains all SNR levels and noise corrupt
          Signals for one clean ECG record.
14  end
```

TABLE 9 ALGORITHM DESIGN TO CONSTRUCT NOISY ECG SIGNALS.

DNN Models

Region Based Convolution Neural Network

```
##### Region Based Convolutional Neural Network #####
class RCNN(nn.Module):
    def __init__(self, input_size):
        super(RCNN, self).__init__()

        self.input_size = input_size

        self.cv1_k = 3
        self.cv1_s = 1
        self.cv1_out = int(((self.input_size - self.cv1_k)/self.cv1_s) + 1)

        self.cv2_k = 3
        self.cv2_s = 1
        self.cv2_out = int(((self.cv1_out - self.cv2_k)/self.cv2_s) + 1)

        self.cv3_k = 5
        self.cv3_s = 1
        self.cv3_out = int(((self.cv2_out - self.cv3_k)/self.cv3_s) + 1)

        self.cv4_k = 5
        self.cv4_s = 1
        self.cv4_out = int(((self.cv3_out - self.cv4_k)/self.cv4_s) + 1)

        self.layer_1 = nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=3, kernel_size=(3)),
            nn.BatchNorm1d(num_features=3),
            nn.ReLU(inplace=True),
            nn.AvgPool1d(kernel_size=1)
        )

        self.layer_2 = nn.Sequential([
            nn.Conv1d(in_channels=3, out_channels=5, kernel_size=(3)),
            nn.BatchNorm1d(num_features=5),
            nn.ReLU(inplace=True),
            nn.AvgPool1d(kernel_size=1)
        ])

        self.layer_3 = nn.Sequential(
            nn.Conv1d(in_channels=5, out_channels=3, kernel_size=(5)),
            nn.BatchNorm1d(num_features=3),
            nn.ReLU(inplace=True)
        )

        self.layer_4 = nn.Sequential(
            nn.Conv1d(in_channels=3, out_channels=1, kernel_size=(5)),
            nn.BatchNorm1d(num_features=1),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5, inplace=False)
        )

        self.layer_5 = nn.Sequential(
            nn.Linear(self.cv4_out, 1500), # FC Layer
            nn.Linear(1500, 1500) # Regression
        )

    def forward(self, x):
        x = self.layer_1(x)
        x = self.layer_2(x)
        x = self.layer_3(x)
        x = self.layer_4(x)
        x = x.view(x.size(0), -1)
        x = self.layer_5(x)

        return x
```


Convolution Denoising Auto Encoder

```
##### Convolutional Denoising Auto-Encoder #####
class CDAE(nn.Module):
    def __init__(self):
        super(CDAE, self).__init__()

        # Encoder part
        self.encoder = nn.Sequential(
            nn.Conv1d(1, 16, 25, stride=2, padding=12), # Downsample by 2
            nn.BatchNorm1d(16),
            nn.ReLU(),

            nn.Conv1d(16, 32, 25, stride=2, padding=12), # Downsample by 2
            nn.BatchNorm1d(32),
            nn.ReLU(),

            nn.Conv1d(32, 64, 25, stride=2, padding=12), # Downsample by 2
            nn.BatchNorm1d(64),
            nn.ReLU(),

            nn.Conv1d(64, 128, 25, stride=2, padding=12), # Downsample by 2
            nn.BatchNorm1d(128),
            nn.ReLU(),

            nn.Conv1d(128, 256, 25, stride=2, padding=12), # Downsample by 2
            nn.BatchNorm1d(256),
            nn.ReLU()
        )

        # Decoder part
        self.decoder = nn.Sequential(
            nn.ConvTranspose1d(256, 128, 25, stride=2, padding=12, output_padding=1), # Upsample by 2
            nn.ReLU(),

            nn.ConvTranspose1d(128, 64, 25, stride=2, padding=12, output_padding=1), # Upsample by 2
            nn.ReLU(),

            nn.ConvTranspose1d(64, 32, 25, stride=2, padding=12, output_padding=1), # Upsample by 2
            nn.ReLU(),

            nn.ConvTranspose1d(32, 16, 25, stride=2, padding=12, output_padding=1), # Upsample by 2
            nn.ReLU(),

            nn.ConvTranspose1d(16, 1, 25, stride=2, padding=12, output_padding=1), # Final ConvTranspose1d layer
            nn.Sigmoid()
        )

    def forward(self, x):
        # Encode the input
        x = self.encoder(x)

        # Decode the encoded features
        x = self.decoder(x)

        return x
```

Recurrent Neural Network

```
class DRNN(nn.Module):  
  
    def __init__(self, input_size=1, lstm_hidden_size=64, fully_connected_size=64, output_size=1, seq_length=1500):  
        super(DRNN, self).__init__()  
  
        # LSTM layer: input_size = 1 (assuming time series input of 1D signal), hidden size = 64 units  
        self.lstm = nn.LSTM(input_size, lstm_hidden_size, batch_first=True)  
  
        # Fully connected layers  
        self.fc1 = nn.Linear(lstm_hidden_size * seq_length, fully_connected_size) # Multiplied by sequence length for  
        self.fc2 = nn.Linear(fully_connected_size, fully_connected_size)  
  
        # Output layer (1D regression task)  
        self.output = nn.Linear(fully_connected_size, output_size)  
  
        # Activation function  
        self.relu = nn.ReLU()  
  
    def forward(self, x):  
        # LSTM forward pass  
        lstm_out, (hn, cn) = self.lstm(x)  
  
        # Flatten the LSTM output  
        lstm_out = lstm_out.contiguous().view(x.size(0), -1) # Flatten to feed into fully connected layers  
  
        # Fully connected layers with ReLU  
        x = self.fc1(lstm_out)  
        x = self.relu(x)  
        |  
        x = self.fc2(x)  
        x = self.relu(x)  
  
        # Output layer  
        out = self.output(x)  
        return out
```

Training Procedure

```
##### Training #####
if userSelectTrain:
    for epoch in range(10):
        print(f"===== EPOCH {epoch} =====")
        running_loss = 0.0

        for batch_idx, (clean_signals, noisy_signals) in enumerate(dataloader):
            print(f"Batch {batch_idx}: clean_signal shape = {clean_signals.shape}, noisy_signal shape = {noisy_signals.shape}")

            # Continue with the rest of the processing
            clean_signal = clean_signals.float().to(device)
            noisy_signal = noisy_signals.float().to(device)

            for i in range(noisy_signal.size(1)):
                noisy_segment = noisy_signal[:, i, :]
                optimizer.zero_grad()
                outputs = net(noisy_segment)
                peaks = torch.tensor(detectors.hamilton_detector(clean_signal.cpu().numpy().flatten())).to(device)
                loss = lossfcn(clean_signal.squeeze(), outputs.squeeze(), peaks, a=20)
                #loss = criterion
                loss.backward()
                optimizer.step()
                running_loss += loss.item()

            print(f'Epoch [{epoch + 1}/10], Loss: {running_loss / len(dataloader)}')
            torch.save(net.state_dict(), './model_weightsCDAE12dB.pt')

else:
    net.load_state_dict(torch.load('./model_weightsCustom.pt'))
    net.eval()
```

Data Preparation

```
class ECGDataset(Dataset):
    def __init__(self, clean_signals, noisy_signals, segment_length=1500):
        self.clean_signals = clean_signals
        self.noisy_signals = noisy_signals
        self.signal_numbers = list(clean_signals.keys())
        self.segment_length = segment_length

    def __len__(self):
        total_segments = 0
        for signal_number in self.signal_numbers:
            signal_length = self.clean_signals[signal_number].size
            segments_per_signal = signal_length // self.segment_length
            total_segments += segments_per_signal * len(self.noisy_signals)
        return total_segments

    def __getitem__(self, idx):
        segments_per_signal = 10 # Assuming 15000 samples per signal and 1500 samples per segment

        signal_idx = idx // (segments_per_signal * len(self.noisy_signals))

        if signal_idx >= len(self.signal_numbers):
            raise IndexError(f"Signal index {signal_idx} out of range for available signals")

        segment_in_signal_idx = idx % (segments_per_signal * len(self.noisy_signals))
        snr_idx = segment_in_signal_idx // segments_per_signal
        segment_idx = segment_in_signal_idx % segments_per_signal

        signal_number = self.signal_numbers[signal_idx]
        snr = list(self.noisy_signals.keys())[snr_idx]

        start_idx = segment_idx * self.segment_length
        end_idx = start_idx + self.segment_length

        if signal_number in self.noisy_signals[snr]:
            clean_signal_data = self.clean_signals[signal_number][0]
            clean_signal_segment = clean_signal_data[start_idx:end_idx]
            noisy_signal_segments = []
            for noisy_signal_copy in self.noisy_signals[snr][signal_number]:
                if noisy_signal_copy is None:
                    print(f"Warning: Noisy signal copy is None for file {signal_number} at SNR {snr}. Skipping.")
                    continue # Skip this segment if it's None
                noisy_signal_segment = noisy_signal_copy[0][start_idx:end_idx]
                noisy_signal_segments.append(noisy_signal_segment)

            # Check if we have any valid segments after filtering
            if len(noisy_signal_segments) == 0:
                print(f"Warning: No valid noisy signal segments found for file {signal_number} at SNR {snr}.")
                return torch.tensor([]), torch.tensor([])

            clean_signal_tensor = torch.tensor(clean_signal_segment).unsqueeze(0)
            noisy_signal_tensor = torch.tensor(noisy_signal_segments)

            clean_signal_tensor = clean_signal_tensor.unsqueeze(0)
            noisy_signal_tensor = noisy_signal_tensor.unsqueeze(1)

            return clean_signal_tensor, noisy_signal_tensor

        return torch.tensor([]), torch.tensor([])
```