FIT3162
Final Project Report

Ensemble Deep Convolutional Neural Network for Cardiac Abnormality Detection using Synchronized Electrocardiogram and Phonocardiogram Signals

Bryan Ho Yung Kynn
30221455
bhoo0005@student.monash.edu

Chew Shen Min
29640938
sche0094@student.monash.edu

Ryan Chow Shiu Wei
29914124
rcho0018@student.monash.edu

Team: MA_B_7
Supervisor: Dr Ting Chee Ming
Word Count: 9290

Table of Contents

1 Introduction

Cardiovascular diseases have long been one of the deadliest forms of diseases recorded yearly. According to the World Health Organization (WHO, 2017), more people die annually from cardiovascular diseases than any other diseases with an estimated 17.9 million people died in 2016 that represents 31% of global deaths. 85% of those deaths are due to heart attacks and strokes. Therefore, detecting abnormalities and diseases early played a major role towards heart diseases diagnosis and future research for further improving heart treatment standards. Many hospitals and physician practices around the world rely on Electronic Health Records (EHR) systems for monitoring patients and detecting abnormalities. EHR data can be used to predict events including related diseases, but there is only little known regarding practical tradeoffs between collected data and data modelling. Hence, the application and usage of machine learning methods to EHR has increased in popularity as it has the benefit of predicting future patient outcomes and gives a better understanding of data modelling practices (Ng et al., 2016, p. 656). In addition to machine learning, deep learning as a subset of machine learning also brings the benefit of automating classification by studying and analyzing data using artificial intelligence or algorithms inspired by the structure of the brain called artificial neural networks.

The aim of our project is to develop an automatic normal-abnormal classification method using two synchronous physiological signals, the Electrocardiogram (ECG) and Phonocardiogram (PCG) measurements provided by the user. Prior to classification, data quality assessment would be performed on the provided synchronous data through the command-line interface to identify low quality data to the user. A deep learning algorithm will then execute automatic classification on the filtered data to classify normal-abnormal cardiac events using artificial neural networks. Once the deep learning algorithm is executed, benchmarking will then take place between the ensembled deep learning algorithm and conventional machine learning algorithms to provide a report to the user stating accuracy of each approach towards classifying the ECG-PCG data.

2 Background

2.1 Literature Review - Updated

In the following subsections we will discuss the changes we have adopted in our approach since it was first conceived in our proposal in FIT3161, in addition to the literature we have based these changes on and discuss potential avenues for future work.

2.2 Preprocessing

We detailed many different preprocessing methods for ECG and PCG signals in our proposal from FIT3161. For instance we intended to implement the 3 step preprocessing method used in Rawther and Cheriyan (2015) — first apply five-order moving average filtering, then Butterworth high-pass filtering and lastly Butterworth low-pass filtering

.

1) Five-order moving average filtering (Electromyographic noise removal).

$$y(n) = \sum_{k=0}^{N} b_k x(n-k) \quad (1)$$

2) Butter worth high-pass filtering with cutoff frequency, fc =0.5 Hz (drift suppression).

$$|H(\omega)^2| = \frac{1}{1+(\frac{\omega c}{\omega})^{2N}} \quad (2)$$

3) Butter worth low-pass filtering with fc = 45 Hz (power line interference removal).

$$|H(\omega)^2| = \frac{1}{1+(\frac{\omega}{\omega_c})^{2N}} \quad (3)$$

Figure 2.1. 3 step preprocessing method for ECG signals. Obtained from (Rawther & Cheriyan, 2015).

We also learned from the study by Li et al. (2020) that we referenced in our literature review for FIT3161 that PCG signals are typically very noisy due to sources such as electromagnetic interference, lung sounds and power frequency interference.

After consulting our supervisor, and due to lack of experience in implementing said ECG and PCG signal denoising methods, we have opted to only use standardization; subtracting the mean and dividing by standard deviation for preprocessing the ECG and PCG signals in our provided dataset. Moreover, it is suggested in literature to normalize signals using the z-score method before input to deep learning models to improve performance (Roy et al., 2019). Implementation of the denoising algorithms proposed in Li et al. (2020), as well as in Rawther and Cheriyan (2015) would certainly be one of the best improvements that could be made to our approach in the future.

2.3 Feature Extraction

2.3.1 Continuous Wavelet Transform (CWT)

Based on Wang et al. (2021) which performed classification on ECG signals with CWT and achieved 98.74% accuracy, they also suggested that their proposed method showed potential to be used as a clinical auxiliary diagnostic tool, we have opted to use CWT in our CNN model as well as LSTM model. CWT was also suggested by our supervisor.

2.3.2 Mel-Frequency Cepstrum Coefficients (MFCC), delta MFCCs and delta-delta MFCCs

To extract features from PCG signals, we have opted to use standard MFCCs alone for input into CNN model, while for LSTM model we used MFCCs with its first order derivative (delta MFCCs) and second order derivative (delta-delta MFCCs) appended to the end of it. This approach utilizing first and second order derivatives of MFCCs was also suggested by our supervisor.

Moreover we see from Deng et al. (2020) which used an improved MFCC extraction technique where the Mel-frequency cepstrums are calculated without dividing the heart sound signal. Deng et al. (2020) also used combined delta MFCCs and delta-delta MFCCs in their approach. From Deng et al. (2020), we also see a potential avenue for future work, that is to implement a combined 2D-CNN and LSTM model for feature extraction and classification, rather than CNN and LSTM separately as we do in our approach.

2.4 Model Implementation

2.4.1 CNN

The architecture of our CNN model was implemented based on the CNN architecture proposed in Fawaz et al. (2019). We added a batch normalization layer after each 1D convolution layer to prevent internal covariate shift across one mini-batch training of time series (Ioffe and Szegeddy, 2015). Also from Fawaz et al. (2019), we experimented with reducing the learning rate by 0.5 (new learning rate = old learning rate * 0.5) each time the model's training loss does not decrease for 50 epochs, with 0.0001 as the learning rate lower bound.

2.4.2 LSTM

Our LSTM model architecture consists of 3 stacked LSTM layers followed by a dropout layer, then a dense layer with ReLU activation function and lastly a dense layer with sigmoid activation function for output was implemented through a combination of suggestions from our supervisor and the approach proposed in Pandey and Janghel (2020). Our supervisor suggested the use of a deeper network architecture — stacked LSTM layers, dropout layer

between LSTM layer and dense layer with ReLU activation function to reduce over-fitting (Pandey and Janghel, 2020).

2.5 Performance Optimization

The implementation of our performance test/optimization methodology was based on Wilson et al. (2017), they found that while it is generally believed that the Adam optimization method does not require tuning, their results show that tuning the initial learning rate and decay rate result in significant performance gains. This applies to both our models as they both use the Adam optimization method, we have additionally chosen the hyperparameter batch size to vary along with learning rate in our performance test.

3 Methodology

3.1 System Architecture

The overall system architecture consists of 4 major components as shown in Figure 3.1. Although the approach to data preprocessing applies to both ECG and PCG, their corresponding feature extraction methods vary due to their unique characteristics, such that the former are the electrical impulses traversing through the heart while the latter are the mechanical murmurs resulting from heart auscultation (Ismail, Siddiqi, & Akram, 2018). The system also offers two modes of classification for a broader approach on evaluation. Each component will be discussed in detail in the following sections.

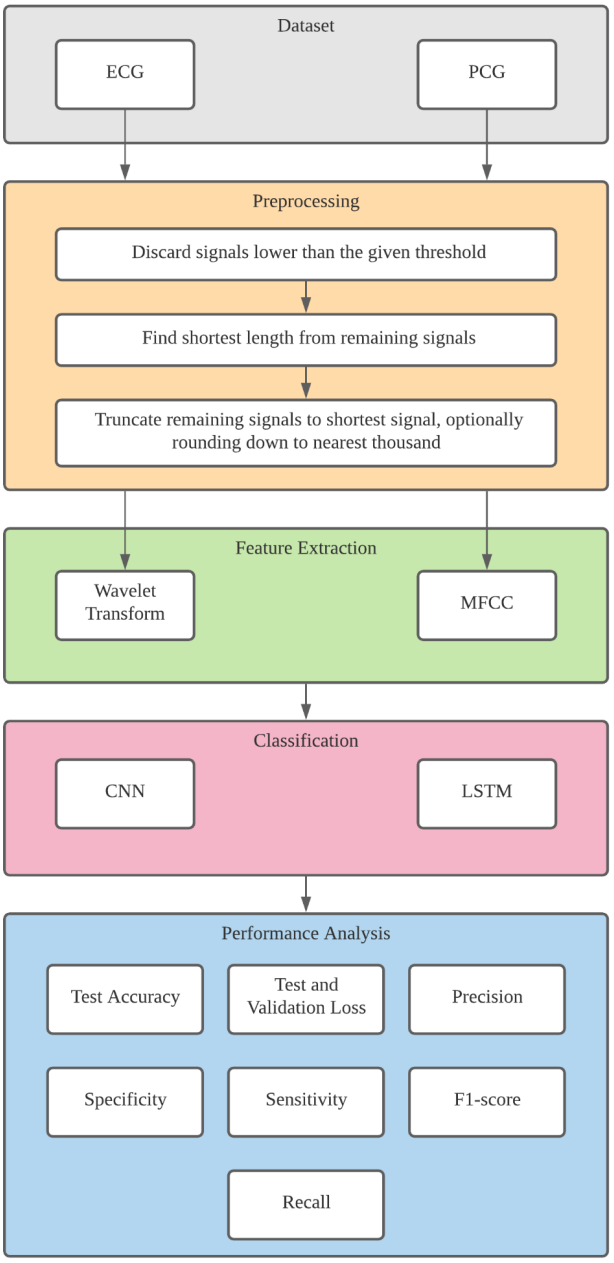


Figure 3.1. Overall architecture.

3.2 Data Preprocessing

3.2.1 ECG-PCG Dataset

The dataset was provided by our supervisor, consisting of 405 subjects in MATLAB file formats, each with their recorded ECG and PCG signals. An additional CSV file was included that classifies the abnormality and signal quality index (SQI) of each subject. In summary, the dataset consisted of 117 normal and 288 abnormal subjects. Each subject was sampled at a rate of 1000 Hz.

The removal of baseline wanders through the use of a notch filter of 0.01 Hz cut-off frequency results in the following visualizations of normal and abnormal ECG-PCG signals.

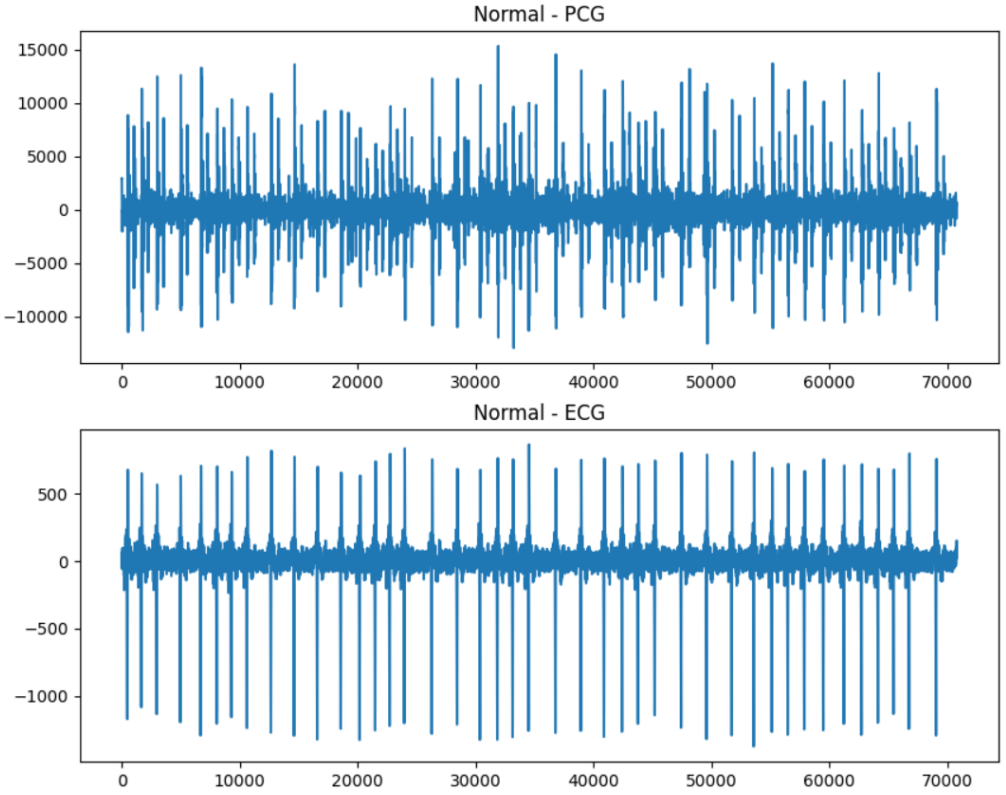


Figure 3.2. Normal PCG and ECG.

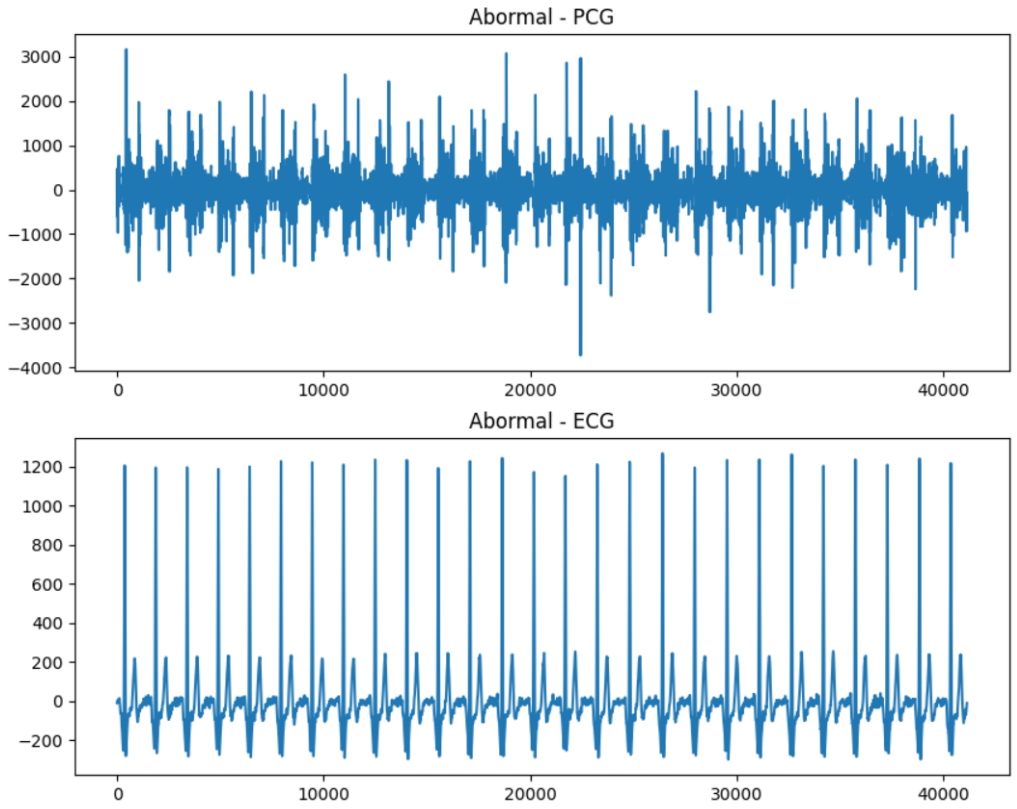


Figure 3.3. Abnormal PCG and ECG.

3.2.2 Preprocessing

Prior to any binary classification, data preprocessing is crucial in enhancing the overall quality of the dataset to improve the performance of the models built. This further allows for greater consistency in evaluating the reliability of the model's prediction accuracy. Hence, the following preprocessing steps are performed in order as shown in Figure 3.4.

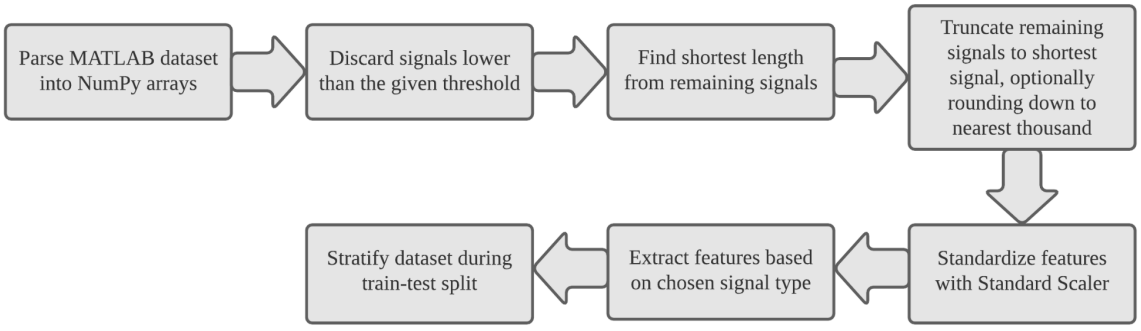


Figure 3.4. Summarized preprocessing steps.

Firstly, the MATLAB dataset needs to be parsed into a 2D NumPy array for readability and manipulation. However, we noticed that each subject recorded has a different wavelength of signals. This is an issue as the dimensions of the dataset to be passed into our deep learning algorithms must be fixed. Hence, we approached the problem with the following steps:

1. Discard all signals that have a length less than the given threshold: 20000. This is to remove any outliers for the following step of truncating the signals to minimize data loss. This resulted in 1 subject to be dropped who had a wavelength of 18530.

2. The remaining 404 signals are then iterated to find the newest shortest signal, which was 25955. Every other signal will now truncate their lengths to meet this value, with an optional process of truncating to the nearest thousand of 25000 as a suggested contingency if reshaping the array is required to avoid remainders.

An analysis on this entire preprocessing step was made to provide an overview of the results formulated. Figure 3.5 displays the summarized results of this process.

```
Total number of subjects: 405
Total number of used subjects: 404
Total number of discarded subjects: 1
Total number of normal signals: 116
Total number of abnormal signals: 288
Shortest length: 25955
Rounded down shortest length: 25000
```

Figure 3.5. Preprocessing results.

Next, the features are standardized with a Standard Scaler, where signals are subtracted by the mean and scaled to unit variance. Figure 3.6 shows the equation of the Standard Scaler.

$$z = \frac{x - \mu}{\sigma}$$

$$\mu = Mean$$
$$\sigma = Standard\ Deviation$$

Figure 3.6. Equation of Standard Scaler.

At this stage, we are still handling the dataset in its raw form. Therefore, features are extracted from the signals to better identify the characteristics for a more refined classification. It is a process of dimensionality reduction to reduce the initial raw dataset into more manageable groups to not only reduce computation resources required for processing, but also to reduce redundant data for a given analysis (DeepAI, 2019).

Mel-frequency Cepstrum Coefficients (MFCC) was applied for PCG signals because the features extracted are biologically inspired and resemble the resolution of a human's auditory system (Chowdhury, Poudel, & Hu, 2020). Meanwhile, ECG signals used the wavelet transform function to filter out noise interference such as myo-electrical and power frequency interferences (Tan & Du, 2009). Further details on the MFCC and wavelet transform implementations are detailed under the Feature Extraction subsection.

Finally, a 60:20:20 train-validation-test split was performed with a random state of 42. The split was also stratified due to the original dataset being unbalanced. Therefore, stratifying the split will preserve the proportions of examples for each normal-abnormal class. An evidential example that justifies this is shown in Figures 3.7 and 3.8 which indicates the proportions of normal and abnormal classes with stratification and without stratification when splitting the dataset.

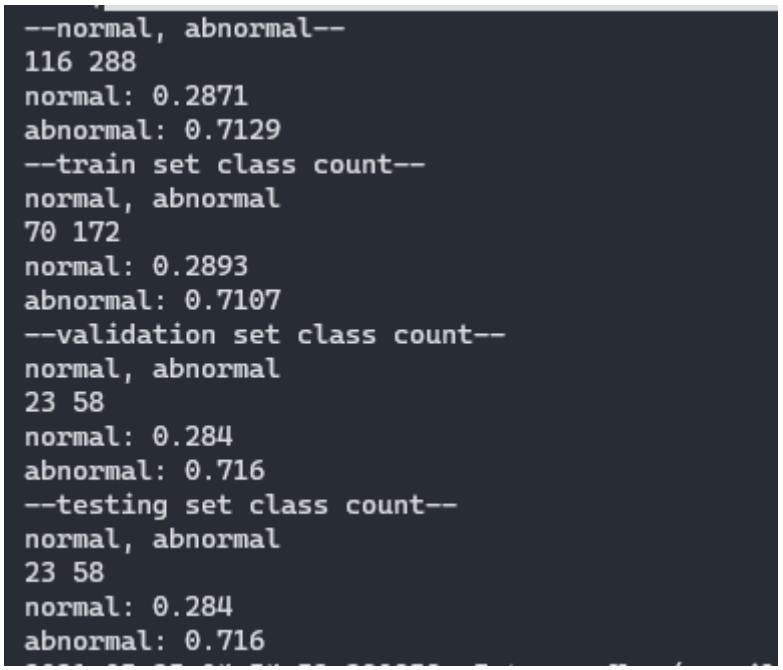


Figure 3.7. Stratified train-test split results in consistent proportions of normal and abnormal classes across training, testing and validation sets.

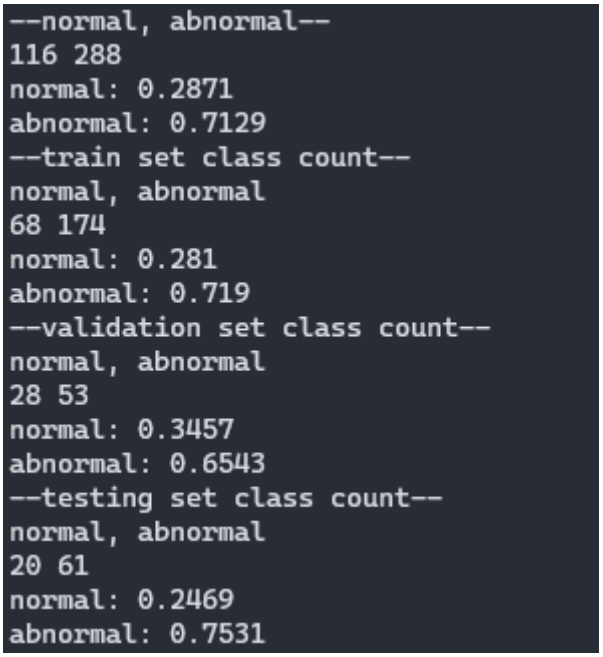


Figure 3.8. Train-test split without stratification results in varying proportions of normal and abnormal classes across training, testing and validation sets.

3.3 Feature Extraction

3.3.1 Mel-frequency Cepstrum Coefficients (MFCC)

For our CNN model, a standard MFCC is used with a sliding window length of 256 samples alongside returning 20 MFCCs. For LSTM however, we return 12 MFCCs using a hop length of 128 samples. In addition, the MFCCs have their first order and second order derivatives (delta and delta-delta MFCCs) appended to the end of their arrays. These delta features are computed using the Savitzky-Golay filter smoothing due to the difference operators of second derivatives being sensitive to noise. A spectrogram of a sample MFCC is visualized below.

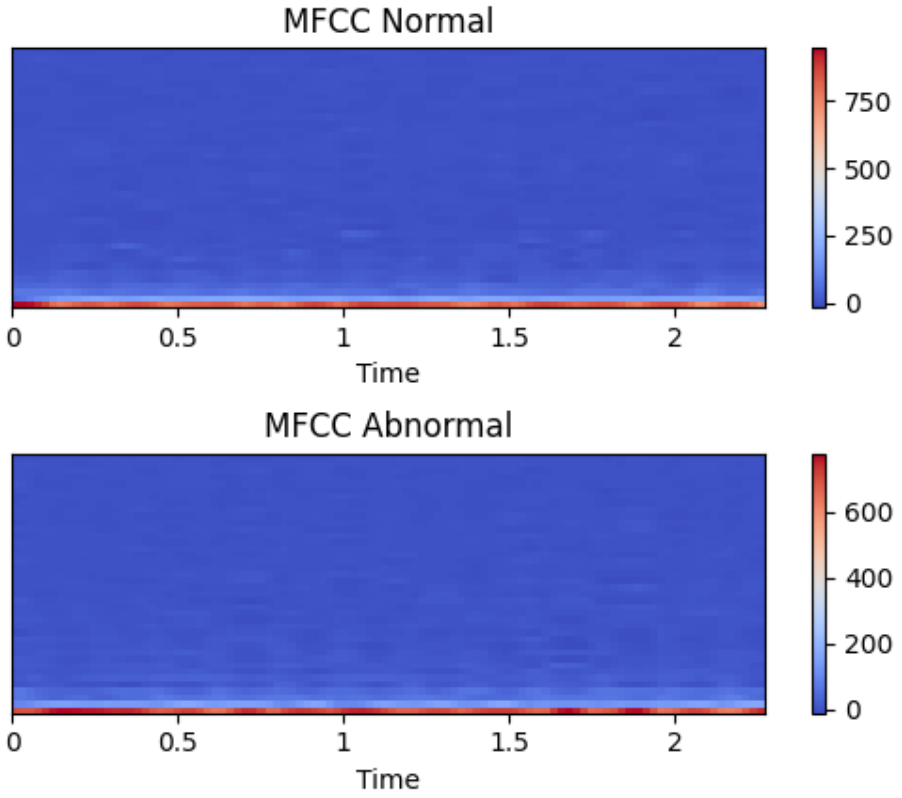


Figure 3.9. MFCC of a normal and abnormal sample.

3.3.2 Continuous Wavelet Transform (CWT)

Initially, the team intended to use R-peak detection for ECG signals with BioSPPy's library. However, several difficulties were encountered in the management of returned features, hence the team opted to use CWT instead with our supervisor's advice. Both CNN and LSTM models utilize a similar continuous wavelet transform function for extracting ECG features. The CWT performs a convolution on the ECG signals using a Ricker wavelet, also known as the Mexican Hat wavelet. Each frequency band returned is scaled to a factor of 10 as a provided parameter. An image visualization of a sample wavelet is shown below.

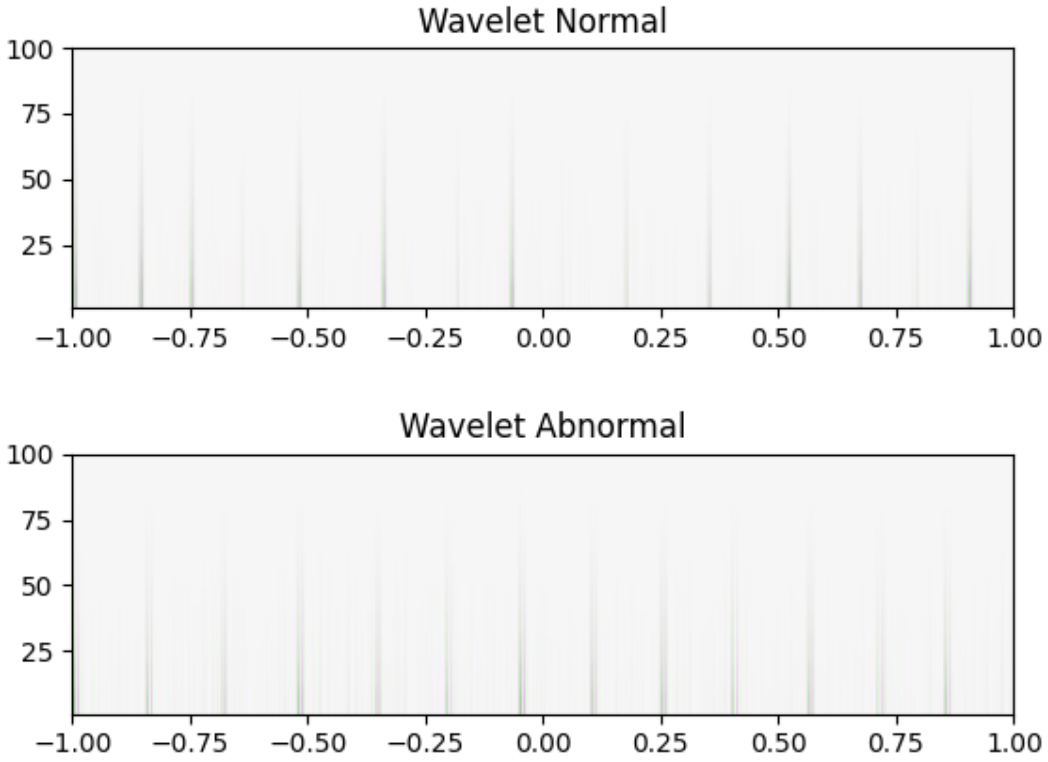


Figure 3.10. Wavelet of a normal and abnormal sample.

3.4 Convolutional Neural Network (CNN)

The architecture of the CNN model was implemented following the proposed design by Fawaz et al. (2019) as mentioned in our Literature Review in section 2.4.1 CNN. An illustration of this framework is shown in Figure 3.11.

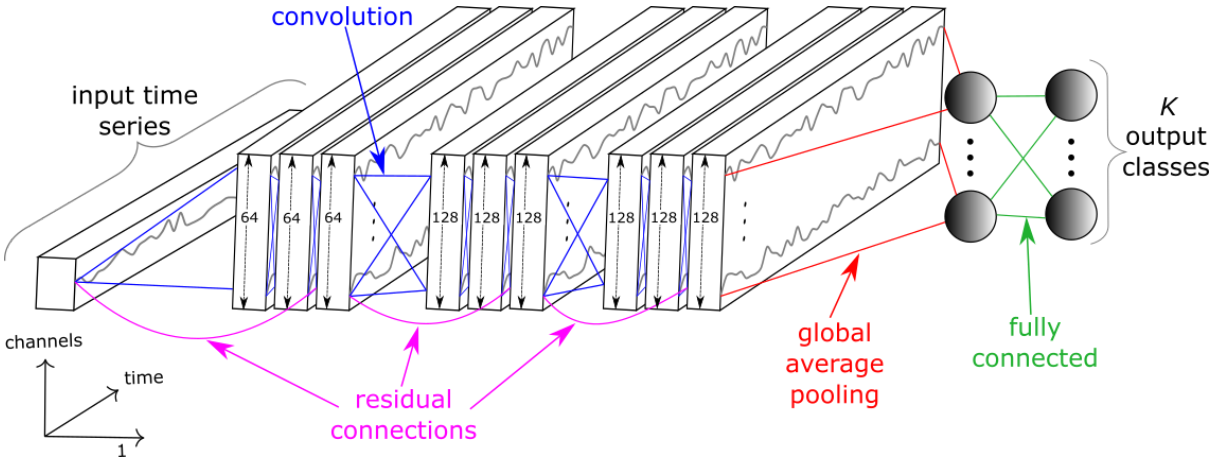


Figure 3.11. Residual network architecture proposed by Fawaz et al. (2019).

The network is composed of 3 residual blocks followed by a 1D global average pooling layer and a final sigmoid classifier of 2 classes (normal and abnormal). While Fawaz et al. (2019) applied a softmax dense layer instead, the results determined by the team suggested that the differences in performance were unnoticeable. Furthermore, the softmax function is used for multivariate classification tasks, while the sigmoid activation function is mainly used as the output unit in binary classification due to its ability on mapping probability-based output (Nwankpa et al., 2018), making it highly suitable for our case.

Each residual block consists of three 1D convolutions fixed to 64 filters whereby its output is added as the input to be fed into the next layer. This is coupled with a ReLU activation function preceded by a batch normalization layer to prevent internal covariate shifts, furthermore promoting a faster learning rate with lesser caution towards parameter initialization (Ioffe and Szegeddy, 2015).

In addition by Fawaz et al. (2019), experimentations were conducted to reduce the learning rate by a factor of 0.5 during callbacks each time the model's validation loss does not improve after 50 iterations, with a lower bound of 0.0001.

3.5 Long Short-Term Memory (LSTM)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 36, 128)           166400
_________________________________________________________________
lstm_1 (LSTM)                (None, 36, 128)           131584
_________________________________________________________________
lstm_2 (LSTM)                (None, 256)               394240
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 32)                8224
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 700,481
Trainable params: 700,481
Non-trainable params: 0
```

Figure 3.12. The architecture of our LSTM model.

The LSTM model was implemented as described in our updated literature review in section 2.4.2 LSTM. The addition of a dropout layer between the 3rd LSTM layer and the dense layer with ReLU activation function was inspired by the work done in Pandey and Janghel (2020).

In hindsight, after analyzing our performance test/optimization results and observing overfitting, we try adding another dropout layer followed by a dense layer with ReLU activation function before the final dense layer, which is the output layer in order to reduce overfitting.

3.6 Performance Optimization

Our performance optimization and benchmarking approach consisted of tuning the learning rate and batch size used when training the CNN and LSTM models. This hyperparameter tuning was only done for models which took in extracted features as input; CWT for ECG signals, standard MFCCs alone were used in the CNN model and MFCC with first and second order derivatives of the coefficients appended to the end of it were used in the LSTM model. We then compared the results of the best performing models against those of the same model except using raw features as input.

For the CNN model we varied the learning rate through the following set of values: 0.001, 0.0005, 0.0001, 0.00005 and 0.00001. While batch size was varied between: 28, 30, 32, 34 and 36.

For the LSTM model, learning rate was varied through: 0.0001, 0.00005, 0.00001, 0.000005 and 0.000001. Note that these values were chosen because when larger values were used, such as those tested for the CNN model, we observed that validation loss stopped improving and started increasing very early on during training. Batch size was varied between: 14, 16, 18, 20 and 22.

4 Project Management

4.1 How was the Project Conceived

During the first part of this project, we were given a list of deep learning topics from various fields including cybersecurity, medicine, and document classification. As the team was allowed to freely select any of the topics given, we were initially interested in facial and audio deep fakes proposed by Professor Raphael. However, upon enquiry, we were told that there were no available slots for us, so we had to opt for another topic.

We decided to email Ms Prabha on document classification and Dr Ting on heart signal classification respectively. After some consultation between the two supervisors, we were more interested in Dr Ting's proposal as he had provided us with a couple of examples on the composition of heart signals, their differences in behaviours in relation to the features intended to be extracted, and how the team could approach the given topic. In addition, Dr Fuad would be his assistant to guide us on any uncertainties we face in regards to the technical implementations of deep learning classification on ECG and PCG signals. While the team initially felt intimidated by the project scope as we had little to no experience in deep learning, nonetheless we thought it would be a worthwhile learning experience. As a result, the team decided to choose Dr Ting's topic and began planning out the project design and prototype.

The team's initial approach on the prototype was to classify ECG and PCG signals with simple machine learning algorithms such as the decision tree and random forest classification to get a basic understanding on the structure of how machine learning works before proceeding to deep learning. Once we successfully implement it for our project design, the team would proceed on investigating CNN and LSTM to improve the prototype further as discussed with Dr Ting. The team began planning out the work breakdown agreement and communication platforms, where the team will have weekly meetings with Dr Ting to update on the overall progress, feedback on our software, and enquire for any clarification of our deliverables.

4.2 How was the Project Managed

Due to the COVID-19 pandemic, our project was held completely online, but this was already expected from the team and we had planned the necessary procedures written in our project proposal.

We had agreed to communicate through WhatsApp as our main platform of communication, consisting of two groups: One where team members can discuss the project's development and tasks, while the other included Dr Ting and Dr Fuad for questions on the project as well as weekly meeting reminders. The team also communicated on Discord as a secondary platform as planned to share literature review resources, code implementations, and conduct

informal voice call meetings. Our weekly formal meetings are held on Zoom with Dr Ting and the meeting minutes are documented.

As for any document-related deliverable (i.e., test report, meeting minutes, user and technical guide), they are maintained on our shared Google Drive folder to allow for collaborative writing. As for the project's codes, we used GitLab for version control, storage, and continuous development and integration of the project.

4.3 Project Management Approach

The development of this project adheres to the Waterfall model where project activities are broken into linear sequential phases (Lewis, 2019). The main reason for this approach is because requirements have been clearly defined and well understood by the team. Furthermore, it is unlikely that the requirements will change severely during the entire life cycle of the project, which includes the dataset used, feature extraction methods performed, and the deep learning algorithms to be implemented. In the case where requirements do change however, the team has discussed with our supervisor, determined the potential areas that pose the risks and have derived alternatives to resolve them. The proposed structure of the development plan is shown in Figure 4.1.

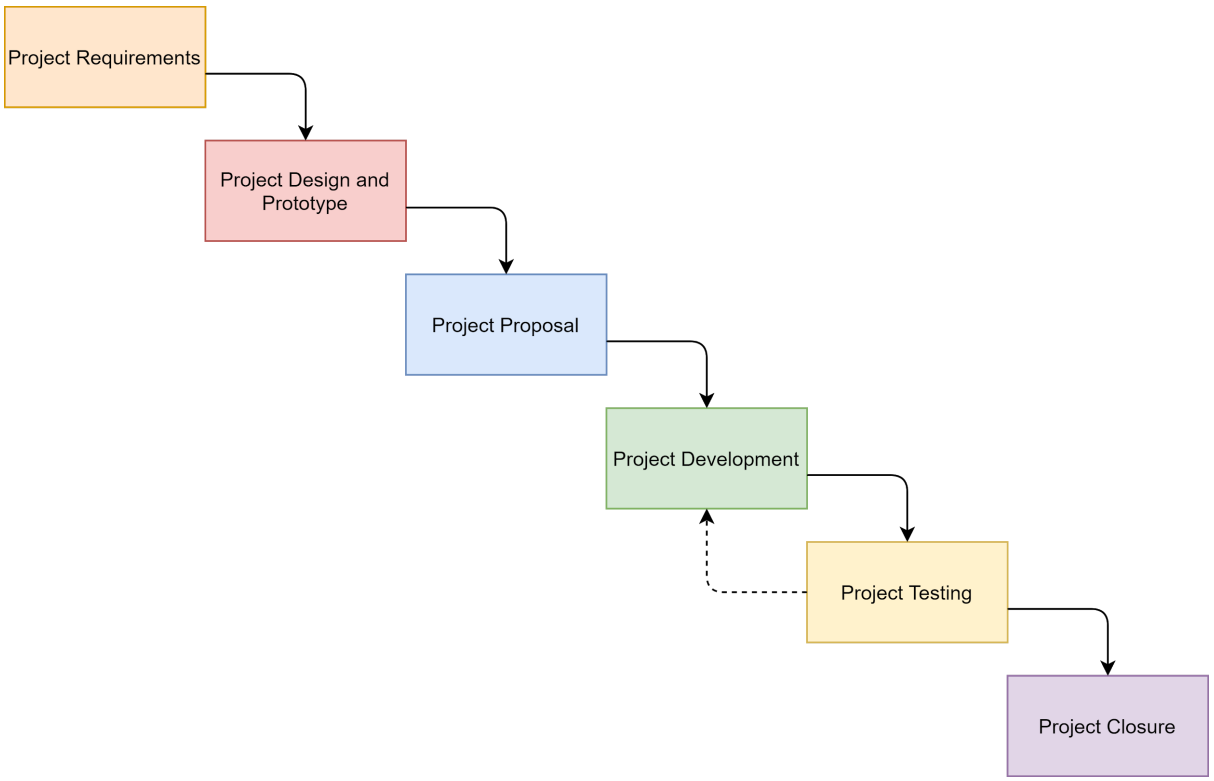


Figure 4.1. Waterfall Model.

The project was initialized in Semester 2, 2020 where requirements were first gathered from our supervisor. An interview was carried out remotely through a Zoom meeting where our supervisor explained the specifications and expectations of the project to the team while providing a document of the project description to further clarify the scope of the project. After the project's requirements were identified and comprehended, the project entered the design and prototype phase. During this phase, the team described the architectural design of the system and extrapolated it with a basic proof of concept code of a decision tree classifier as agreed by our supervisor.

The next phase was the project proposal to outline the project's core value propositions. This established the project's objectives with supporting literature reviews, project management plans and proposed methodologies. After the acceptance of the project proposal, the project entered the development phase in Semester 1, 2021. During this phase, the entire software is developed according to the requirements specified with emphasis on continuous integration and weekly progress reports within the team. Every week, incremental improvements were made from the last, beginning with the modifications of raw data preprocessing, followed by CNN and LSTM model implementations, MFCC and wavelet transform feature extractions, and finally the addition of performance metrics.

After development, the testing phase succeeded which consisted of unit testing (whitebox), integration testing (blackbox) and performance testing on classification results. Due to the rigidity of the Waterfall model, the team proposed a slight modification to the Waterfall model to allow some flexibility between the development and testing phases of the project. This was made in conjunction with the planned weekly meetings we had with our supervisor for feedback loops to enable the team to make further development enhancements. This proved helpful as it prompted us to refactor clunky modules into more testable components while we wrote the test report.

The final stage of the project was then reached in the final week of the semester. A final report was produced wrapping up the entire software development life cycle and details of the end-product, and a software demonstration was performed to our supervisor. This report will be submitted alongside the finished product when the project reaches its closure. As there are currently no plans to provide long-term support or hypercare, the model does not include a maintenance phase. Overall, the team was satisfied to adopt this model as it fitted well into our expectations on commitments and helped us focus on each phase individually to produce organized, incremental results.

4.4 Project Resources, Execution, Management and Planning

4.4.1 Project Resources

Hardware Specifications

Type	Minimum Requirements
Processor	Intel Core i5 or equivalent
GPU	NVIDIA GeForce GT1030
RAM	8 GB
Storage	10 GB of available space

Table 4.1. Hardware Specifications.

Software Specifications

Requirement	Selection	Description
Classification Model Programming Language	Python	Python is a high-level general purpose programming language that emphasizes code readability. It is the language used for back-end development.
Integrated Development Environment	Pycharm	Pycharm is an IDE specifically for the Python language.
Project Management Tools	GitLab	GitLab is a version control tool to allow continuous integration and continuous development of software code.
	Google Drive	Google Drive is a cloud storage for storing the meeting minutes, team management report, test report, and final report.
	Discord	Discord is a secondary communication platform used during the development phase on resource sharing and code.
	WhatsApp	WhatsApp is used as a secondary communication tool.
	ClickUp	ClickUp is used for requirements and issue tracking during development.

Table 4.2. Software Specifications.

Software Libraries

- TensorFlow 2.4.1
- SciPy 1.5.2
- tqdm 4.49.0
- Keras 2.4.3
- scikit-learn 0.24.1
- NumPy 1.19.5
- Matplotlib 3.3.2
- pandas 1.1.2
- librosa 0.8

4.4.2 Project Planning, Management and Execution

The components of the project were broken down into segmented activities that reflected the requirements gathered. This was listed in a work breakdown structure (WBS) documented in Appendix B. The WBS constructed consists of distinguishable branches where each branch is equivalent to the phases of the Waterfall model the team is following. Each branch contains the deliverables to be completed for that phase which was evenly delegated to each member.

A Gantt chart was also prepared to illustrate the entire project schedule. This schedule spanned from the topic selection till the project's closure. The schedule was also arranged in consideration of the Waterfall model such that clearly defined phases are outlined with their corresponding deliverables scheduled, but phases do not overlap with each other. The Gantt chart can be found in Appendix C. Microsoft Project was used to design the following documents.

In addition, the high-level responsibilities of every member were determined from the project proposal in Table 4.3 while Appendix D described the responsibility assignment matrix whereby responsibilities are defined in greater detail and roles are categorized based on the RACI (Responsible, Accountable, Consulted, Informed) model.

High-Level Responsibility	Member
Project Manager	Bryan Ho Yung Kynn
Technical Lead	Ryan Chow Shiu Wei
Quality Assurance	Chew Shen Min

Table 4.3. High-level Responsibilities.

During the development of the project, the timeline was slightly revised from the project proposal where some modules took greater precedence than others due to new findings (i.e., performance analysis on raw data to be carried out first before feature extraction for results comparison), and a command-line interface was opted instead of a web-based user interface due to it being an out-of-scope requirement, which was agreed by our supervisor. These changes fortunately did not affect the overall duration of the project and was well within the controlled risks assessed from the project proposal.

During the testing phase, aside from unit testing and system integration testing, we also evaluated our software in terms of its performance and scalability. This is in consideration of the CNN and LSTM models' binary classification performance during runtime and handling of unbalanced datasets. These tests were documented in the test report.

4.5 Risk Management

Since the team was venturing into the field of deep learning and studies on the heart which were topics that everyone was inexperienced with, the team determined that the potential risks in the implementation process could be high due to unfamiliarity. As such, the team adopted a risk analysis approach illustrated by Ingrid (2020) to prepare for unexpected outcomes. The approach consisted of five steps: Identify, Analyse, Evaluate, Treat and Monitor.

1. Identifying Potential Risks
The team identifies the risks by reviewing the various categories that could potentially impact and be relevant to the project. Such categories determined are scope, technical, human, budget, training and schedule risks.

2. Analyzing Risks
The team discusses the likelihood of the identified risks occurring and the consequences of them. In addition, the root causes and triggers are discussed for each risk to understand the underlying steps that led to the risk. Positive risks are also taken into account to determine what could benefit the team.

3. Evaluating Risks
The team ranks the risks based on their tallied score of probability and severity. This allows the team to better address and prioritise critical risks to avoid unpredictable results.

4. Treating Risks
The team explores the potential responses to address the risk in hopes that the strategy would resolve the issue or prevent repeated occurrences of it. This allows the team to take preemptive strides to control the risks and mitigate them in the future.

5. Monitor and Review Risks
In certain cases, some risks could not be avoided. Therefore, risk owners amongst the team are assigned to manage and inform one another to ensure continuity in the development of the project.

The risk register can be found documented in Appendix E.

4.6 Limitations Encountered During Project Management

In this section, we discuss the limitations encountered on the project from a project management perspective. The limitations are as follows:

1. Time Constraints
    - All team members had challenging units during the two semesters. As a result, time management was proven difficult. There were weeks where certain members had to pause their work due to immense workload and strict deadlines to meet, impacting their dedication to the project.
    - The pandemic had also contributed to the burden of putting the team through physical and mental stress as we had to adapt to the new online learning environment and pick up new techniques, including online collaborative writing and editing recorded presentations which were less efficient than in-person collaborations.

2. Communication
    - The team was completely unable to meet in-person. This forced us to utilize communication platforms for all discussions, meetings, and code reviews. As a result, it made monitoring on team progress difficult. Furthermore, we faced progress stagnation on certain modules due to lack of communication or unavailability.
    - This also affected the expectations between team members due to miscommunication and had ultimately affected our prioritizations on various deliverables in the later weeks.
    - Sharing of technical knowledge was also difficult due to every member being at different wavelengths. Some of us were lacking behind while others made considerable research in specific areas of the project. This not only affected our initial assignment of responsibilities, but also our understanding of each other.

5 Outcomes

5.1 Results Achieved

In this section we present the results and objectives achieved by our project, divided according to the objectives and scope outlined in the project description from FIT3161.

5.1.1 Signal Quality Assessment

Firstly, during the preprocessing phase, we ensure consistency among the signals that are obtained for all subjects by specifying a threshold such as discarding signals lesser than a specified length to be used for our model. To further assess the signal quality to be used for classification by either the CNN or LSTM deep learning algorithms, we performed MFCC and CWT feature extraction to highlight features of PCG with MFCC and ECG with CWT to improve signal quality.

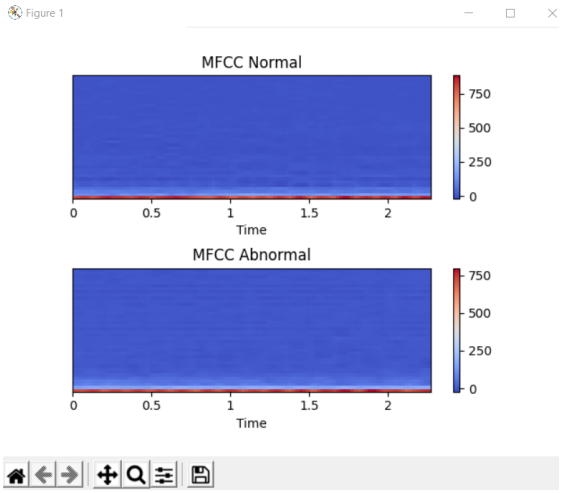


Figure 5.1. MFCC data visualization.

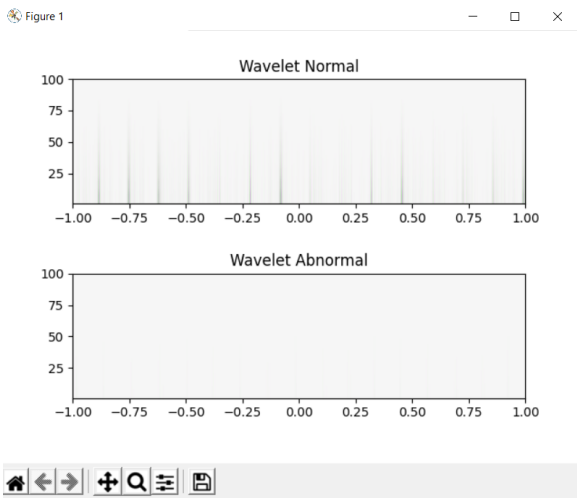


Figure 5.2. CWT data visualization.

5.1.2 Deep Learning Approaches

We applied two deep learning approaches to automatically classify abnormal and normal heartbeats — one using CNN and another using LSTM. Both approaches were able to process homogenous data as input, that is either ECG or PCG signals alone, and then perform automatic classification on them.

5.1.3 Performance Optimization

Before going over the performance results of our best models, we will first explain the structure of the Excel files where we have compiled all our results for each respective model.

33								
34		Test accuracy	Test loss				Average test acc	Average test loss
35	Training run nunber	1		2		3		
36	lr = 0.000005, bs=14	0.6543	0.643	0.679	0.6373	0.6296	0.6615	0.6543	0.6473
37	lr = 0.000005, bs=16	0.6914	0.6376	0.6296	0.6595	0.5802	0.6906	0.6337	0.6626
38	lr = 0.000005, bs=18	0.5556	0.6898	0.6667	0.6743	0.642	0.6867	0.6214	0.6836
39	lr = 0.000005, bs=20	0.5679	0.6611	0.679	0.6308	0.321	0.7017	0.5226	0.6645
40	lr = 0.000005, bs=22	0.5062	0.6943	0.6173	0.6567	0.6543	0.6673	0.5926	0.6728
41								

Figure 5.3. Snippet from the Excel file named "performance-test-results-LSTM-PCG.xlsx".

Figure 5.3 shows the resulting accuracy and loss on the test dataset when our LSTM model was trained with a learning rate of 0.000005 and 5 varying batch sizes. Each training run and evaluation on the test dataset was repeated 3 times, each repetition is labeled by a training run number from 1 to 3. Each training run produces a saved model file, and each saved model file is named according to model, input signal type, learning rate (lr), batch size (bs) and training run number (i1, i2, i3). Thus each tuple of test accuracy and loss corresponds to a saved model file in the Google Drive folder linked in each Excel file. The green highlighted cell represents the learning rate, batch size and training run that produced the best accuracy with minimum loss out of all combinations. Each training run is also accompanied by two plots — training accuracy against validation accuracy and training loss against validation loss. Link to the Google Drive folder is also provided here: https://drive.google.com/drive/folders/1gu8DmXAwX2FNsh_omZ3RHEXxIj9f7o_X?usp=sharing

Model, input signal	Learning rate	Batch size
CNN, ECG	0.0005	36
CNN, PCG	0.0001	36
LSTM, ECG	0.00005	18
LSTM, PCG	0.000005	14

Table 5.1. Best combination of learning rate and batch size for CNN and LSTM with feature extraction for PCG and ECG, selected based on average test accuracy and loss as described in the performance section of our test report.

Now, using the learning rates and batch sizes shown in Table 5.1, we selected the best model out of the 3 training runs for each respective hyperparameter combination. In addition, we also present the corresponding accuracy and loss plots.

Using extracted features:

For CNN model with CWT feature extraction on ECG signals:

Best saved model filename	Test accuracy	Test loss
`cnn_best-ecg-lr5e-05-bs36-i1`	0.7037	0.6359

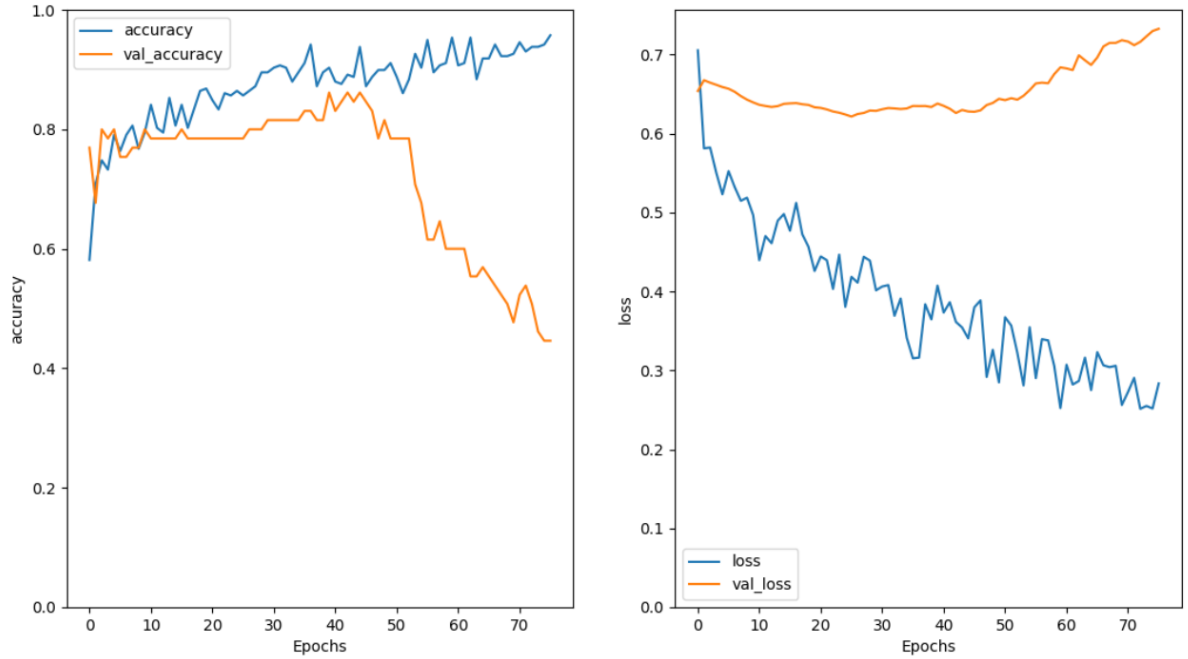


Figure 5.4. Training accuracy against validation accuracy plot, and training loss against validation loss plot from best training run of CNN model with CWT feature extraction.

For CNN model with standard MFCC extraction on PCG signals:

Best saved model filename	Test accuracy	Test loss
`cnn_best-pcg-lr0.0001-bs36-i1`	0.7284	0.5435

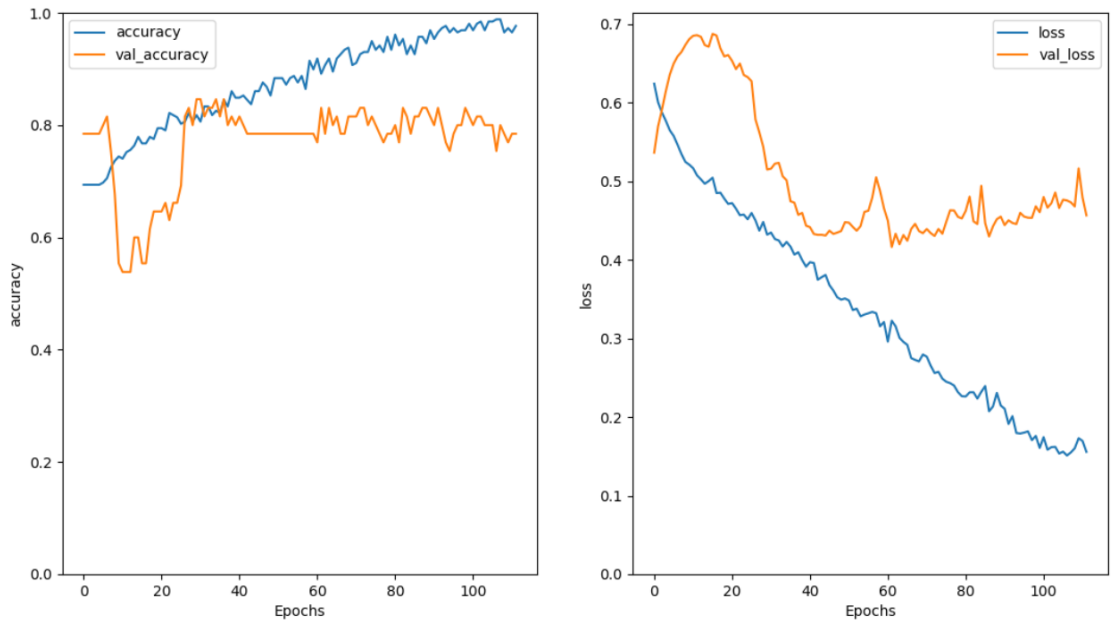


Figure 5.5. Training accuracy against validation accuracy plot, and training loss against validation loss plot from best training run of CNN model with standard MFCC feature extraction.

For LSTM model with CWT feature extraction on ECG signals:

Best saved model filename	Test accuracy	Test loss
`lstm_best-ecg-lr5e-05-bs18-i3`	0.6914	0.6035

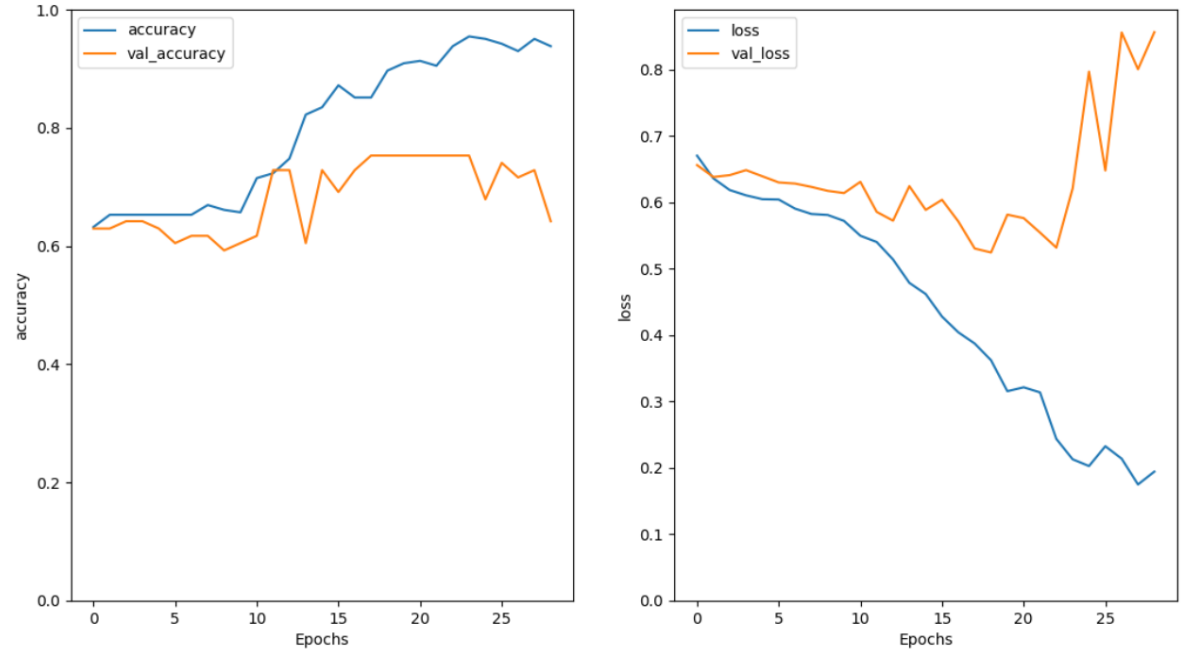


Figure 5.6. Training accuracy against validation accuracy plot, and training loss against validation loss plot from best training run of LSTM model with standard CWT feature extraction.

For LSTM model with MFCC combined with delta MFCC and delta-delta MFCC feature extraction on PCG signals:

Best saved model filename	Test accuracy	Test loss
`lstm_best-pcg-lr5e-06-bs14-i2`	0.6790	0.6373

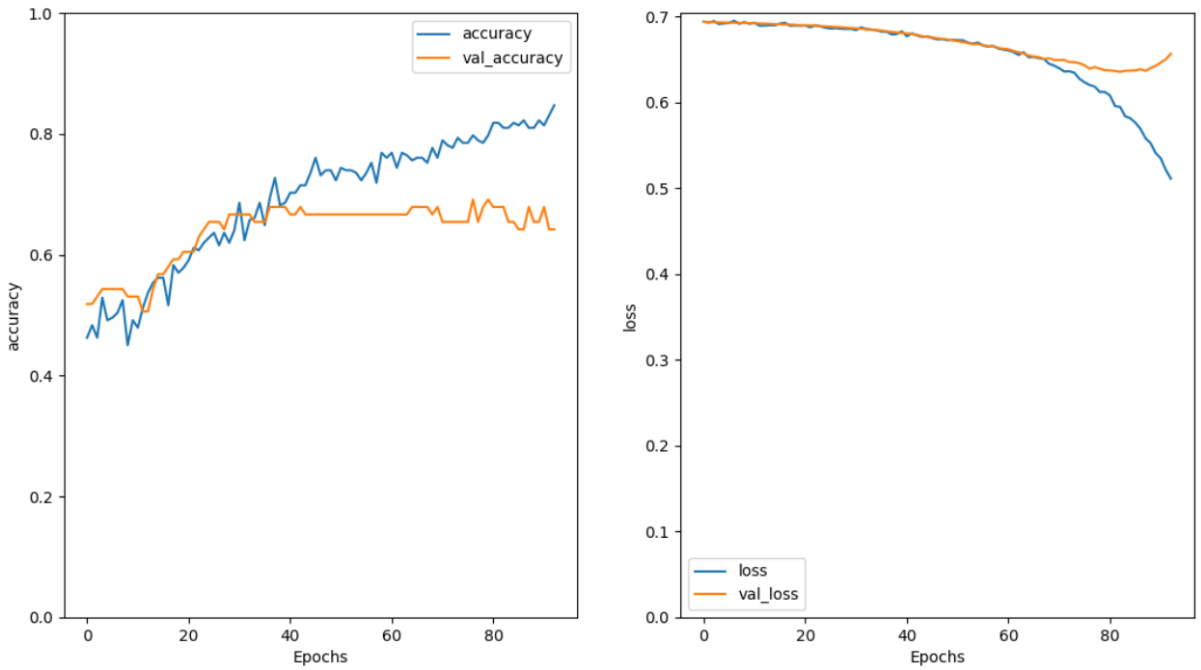


Figure 5.7. Training accuracy against validation accuracy plot, and training loss against validation loss plot from best training run of LSTM model with MFCC combined with delta MFCC and delta-delta MFCC feature extraction on PCG signals.

Using raw features:

For CNN model with raw features on ECG signals:

Best saved model filename	Test accuracy	Test loss
`cnn_raw_best-ecg-lr0.001-bs32-i8`	0.7778	0.5193

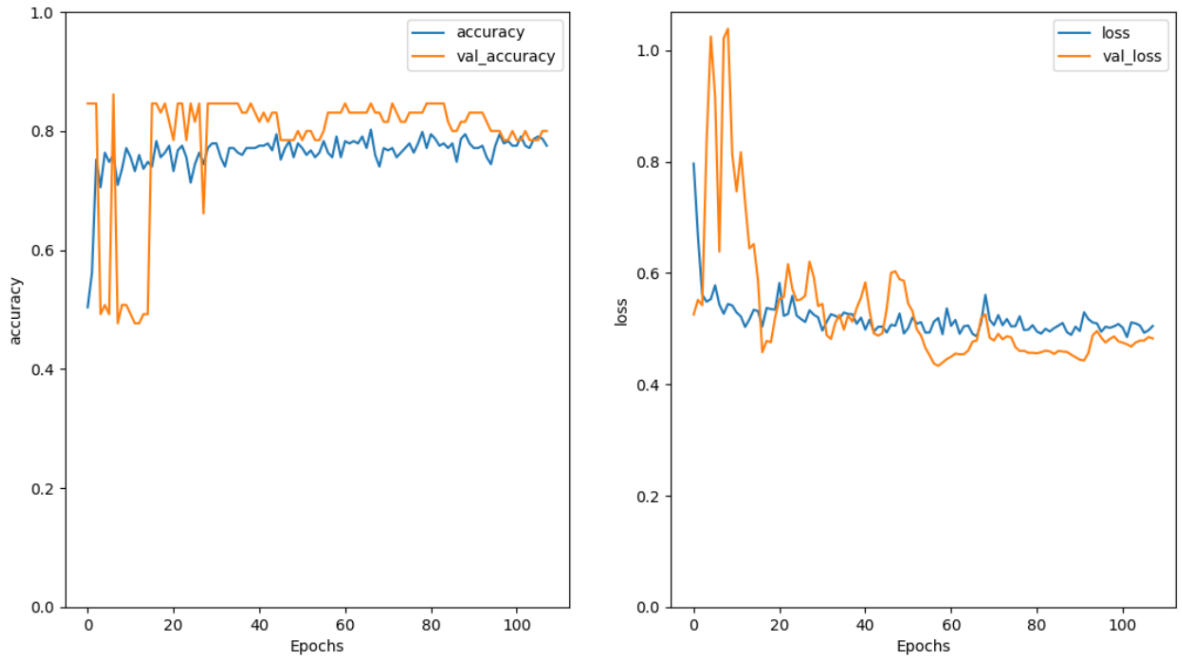


Figure 5.8. Training accuracy against validation accuracy plot, and training loss against validation loss plot from best training run of CNN model on raw features of ECG signals.

For CNN model with raw features on PCG signals:

Best saved model filename	Test accuracy	Test loss
`cnn_raw_best-pcg-lr0.001-bs32-i9`	0.7654	0.5856

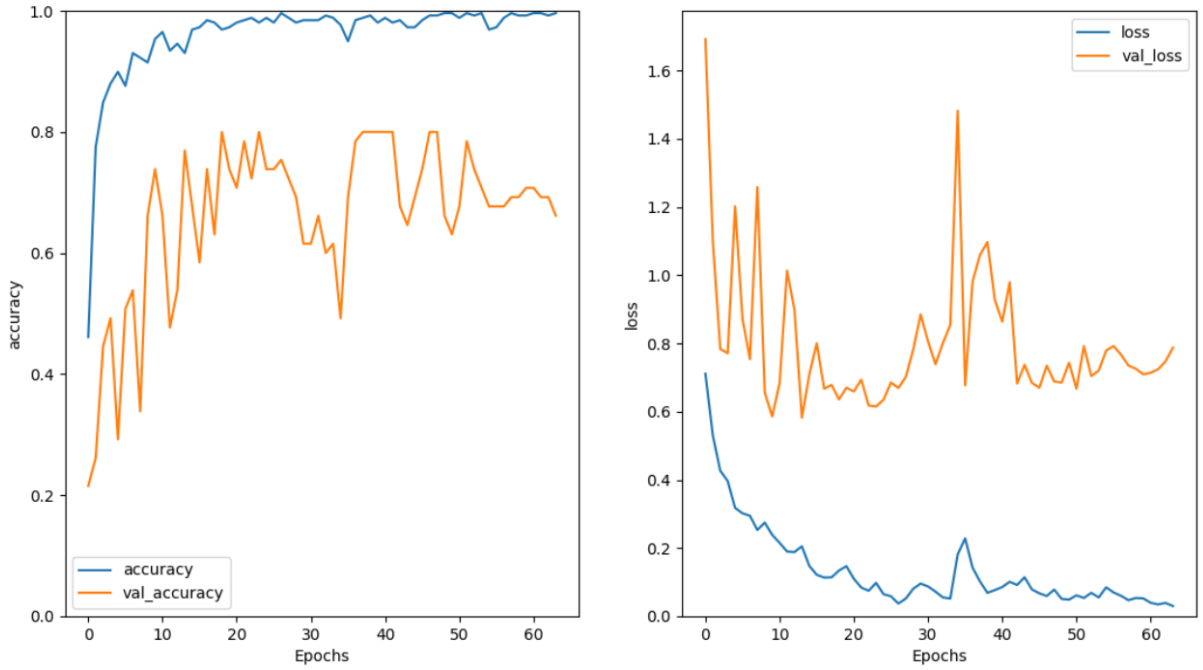


Figure 5.9. Training accuracy against validation accuracy plot, and training loss against validation loss plot from best training run of CNN model on raw features of PCG signal.

For LSTM model with raw features on ECG signals:

Best saved model filename	Test accuracy	Test loss
`lstm_raw_best-ecg-lr0.0005-bs16-i9`	0.7901	0.5589

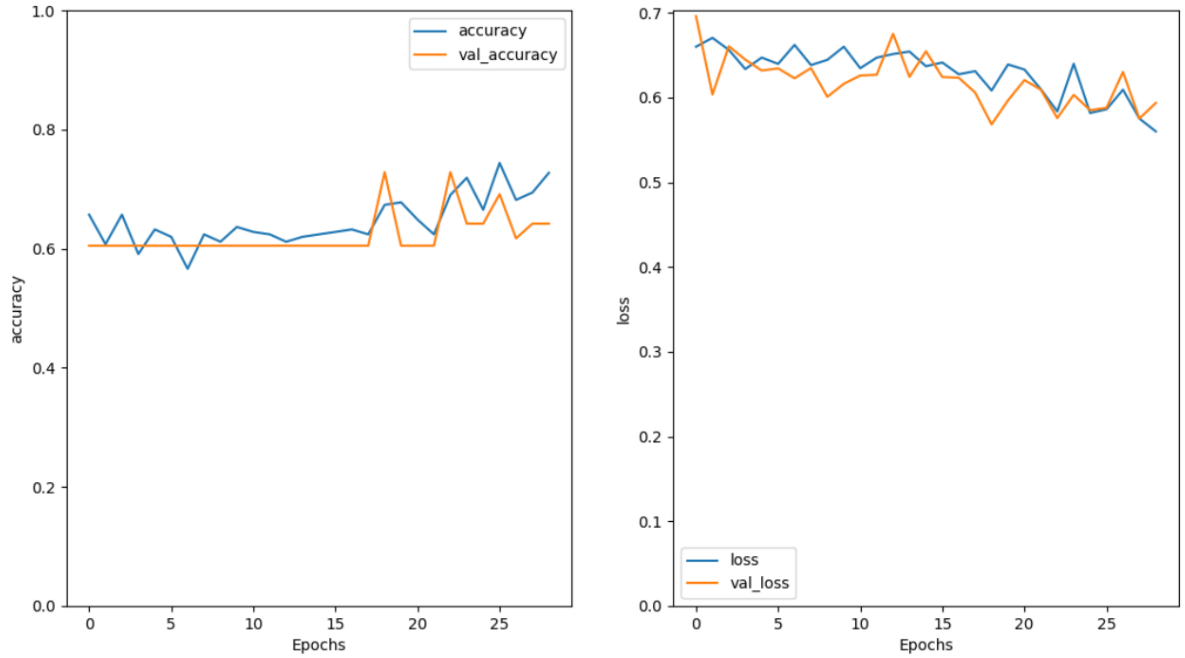


Figure 5.10. Training accuracy against validation accuracy plot, and training loss against validation loss plot from best training run of LSTM model on raw features of ECG signal.

For LSTM model with raw features on PCG signals:

Best saved model filename	Test accuracy	Test loss
`lstm_raw_best-pcg-lr0.0005-bs16-i9`	0.7284	1.1701

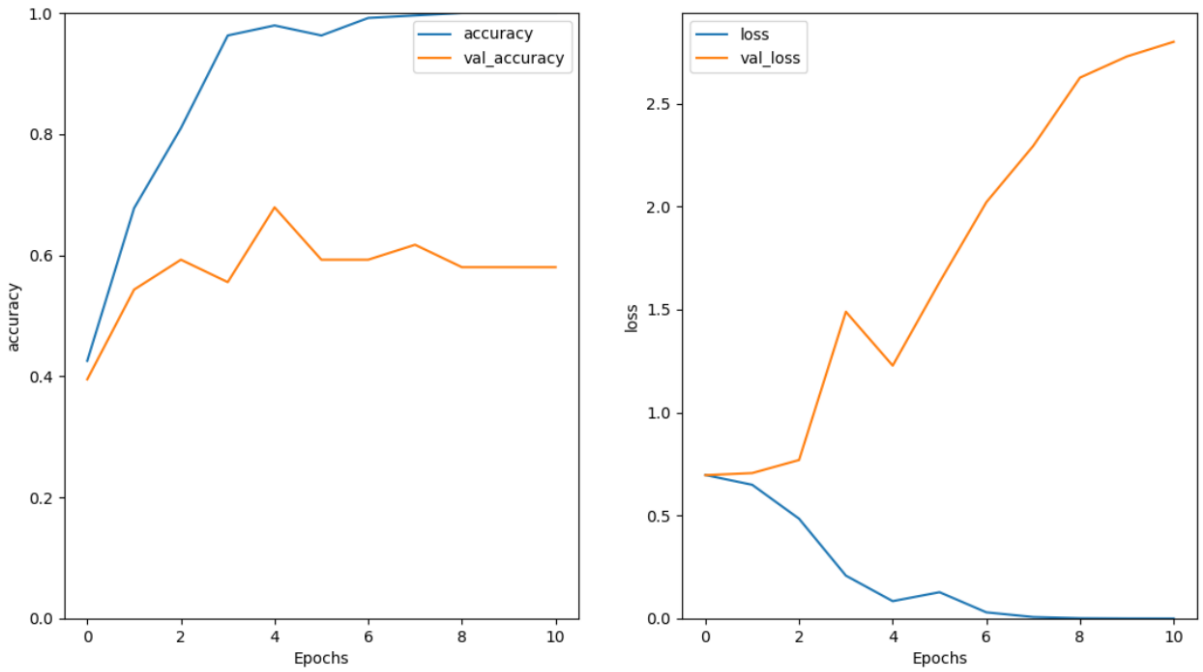


Figure 5.11. Training accuracy against validation accuracy plot, and training loss against validation loss plot from best training run of LSTM model on raw features of PCG signal.

5.2 Discussion on Results, Limitations and Future Improvements

In the following subsections, we will evaluate the results of our project based on the original objectives and scope — what objectives we did or did not achieve, as well as what we would have done differently and avenues of future improvements, given the knowledge and experience we've gained from carrying out this project.

5.2.1 Discussion on Signal Quality Assessment

The achievements that we have done for ensuring signal quality from our deep learning system reflects the requirements and objectives of our project. Inspection of data quality for both our deep learning approaches we have utilized the plotting feature from the Python library *matplotlib* as our GUI to visualize the 2D signals as well as visualizing the results of both deep learning algorithms. In addition, data standardization has been implemented using the z-score method to further improve signal quality by ensuring the signals are of the same dimensions. However, one method of signal quality improvement that we could have done is enhancements prior to discarding low quality data. This issue may prove a significant impact towards classification accuracy if the dataset was on a large scale as data enhancements such as noise reduction and filtering upon low quality data may allow these data to be usable that we have not implemented.

Among the limitations of our dataset and preprocessing methods are its small size after being split into training, validation and testing datasets at a ratio of 60:20:20. In hindsight, a method we should have used to combat this is to evaluate our models using stratified k-fold cross validation method. Doing so would have aided in reducing the overfitting issues our models

faced during training. Fortunately, we did apply stratification when splitting the original dataset, in addition to setting the class weights parameter when training our models so that it values the class with fewer examples more heavily, all to adjust for the imbalanced classes in the original dataset.

5.2.2 Discussion on Deep Learning Approaches

There are several immediately apparent limitations that apply to both our CNN and LSTM implementation approach. Firstly, while we were able to achieve automatic classification using PCG and ECG signals separately, we could not implement a model able to take in synchronous PCG and ECG signals — inputs from two modalities. Among the downsides of our approach is the need to optimize training hyperparameters twice, for CNN and LSTM. Moreover, it is known that there exists heart abnormalities that can be detected through PCG signals but not ECG signals, namely defects related to heart valves and heart murmurs. Even through performing offline analysis by averaging the prediction scores of two models; one from prediction on ECG and one from prediction on PCG to obtain a single prediction score, the averaging operation may result in a prediction score that indicates normal heartbeat (e.g., PCG signal model predicts abnormal with score 0.7, and ECG signal model predicts normal with score 0.1, averaging the two results in a score of 0.4 which is classified as normal if the abnormal/normal threshold is 0.5).

5.2.3 Discussion on Performance Optimization

Before discussing our performance testing results, we preface it by mentioning that each of us have minimal to no prior experience in tuning supervised machine learning model hyperparameters. Thus we will base the following discussion on a few basic understandings of the learning process for machine learning models and how their metrics change throughout training epochs. The loss metric is a measure of how often the model is wrong at learning the dataset after each epoch, training and validation loss measure the model's error on the training and validation datasets respectively. Therefore, a validation loss curve that is increasing while the training loss curve is decreasing at the same time indicates that the model is fitting well to the training dataset, while getting worse at predicting the validation dataset, this is a sign of overfitting. On the other hand, when validation loss is staying the same, it is a sign that the model is not learning or not getting better at predicting the correct output. Lastly, ideal training and validation loss curves would decrease together, converge at some point and both stop decreasing, indicating that the model has learned all it can from the training dataset, at which point the training process should be stopped.

Now on to discussing the results of our CNN and LSTM models, as shown in section 5.1.3 Performance Optimization. Looking at every training loss against validation loss plot for models that use feature extraction — Figures 5.4, 5.5, 5.6 and 5.7, we observe that at some point during training, validation loss begins increasing while training loss continues to decrease. We deduce that the point where validation loss starts increasing is where the model is beginning to overfit, despite our best efforts to thwart this by adding batch normalization layer in CNN and dropout layer in LSTM, as well as tuning the initial learning rate in the

range shown in our performance test results Excel files, we think that the model which had the most ideal training and validation loss plot is the LSTM model with MFCC combined with delta MFCC and delta-delta MFCC feature extraction on PCG signals. Here we observe the two loss curves decreasing together, albeit rather slowly, before validation loss starts increasing at a later point in the training than the other models. As an improvement we could have stopped its training earlier by using a smaller value for the patience parameter in the early stopping callback, such that it halts training sooner rather than waiting for the validation loss to improve for that many epochs.

Next, looking at the loss plots for the models trained on raw features in Figure 5.8, 5.9 and 5.10, we observe that the validation loss curve does not decrease much from its starting point, showing that model is not learning by much as training progresses, in other words it is not doing much better than simply random guessing. We saved the worst for last, as in Figure 5.11 the validation loss never improves at all since the start. At least this proves that the early stopping is working and ending training when it sees no improvement in validation loss after 10 epochs, which is exactly how long this training ran for.

6 Software Deliverable

6.1 Summary of Software Deliverable

6.1.1 What is Delivered

For our project, the deliverable will be our source code that contains our developed deep learning system that we have uploaded to our version control tool GitLab. The full source code can be found at *https://git.infotech.monash.edu/fit3162-heart-cnn/fit3162-heart-cnn*.

In the GitLab repository, the two main directories that house our software are the *project* and *performance-test* directories. The *project* directory contains our main files that houses our software code for each aspect of our deep learning project while for *performance-test* is a copy of the *project* directory with implemented performance optimizations. Here are the directories, files and their descriptions of what they contain listed below:

1) data_processing: contains functions for processing raw MAT files
2) feature_extraction: contains functions for MFCC and CWT
3) cnn_visualization: for visualizing feature extracted data
4) performance_metrics: for calculating evaluation results
5) implement_model: for building a training our CNN and LSTM models
6) cnn.py: Python file for running the CNN algorithm
7) main-lstm.py: Python file for running the LSTM algorithm

6.1.2 Sample Screenshots and Description of Usage

Here is a sample of usage for the command line that the user will input into the command prompt to run a CNN algorithm. The following screenshot shows the command for running a LSTM algorithm that the user will input.

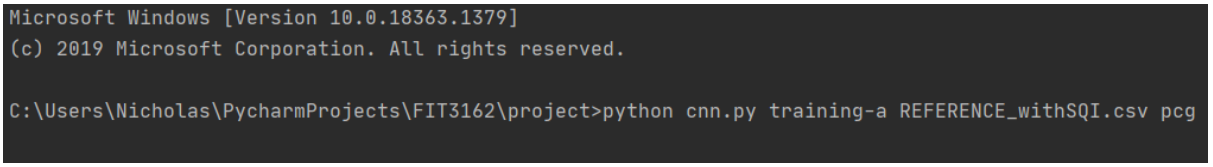


Figure 6.1. Screenshot of a command to run the CNN algorithm using 'training-a' as the folder for MAT files, 'REFERENCE_withSQI.csv' as the file for SQI and 'pcg' as signal type.

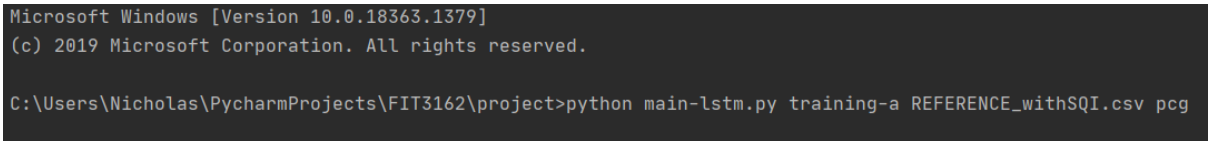


Figure 6.2. Screenshot of a command to run the LSTM algorithm using 'training-a' as the folder for MAT files, 'REFERENCE_withSQI.csv' as the file for SQI and 'pcg' as signal type.

After initiating our software, for the CNN algorithm, it is expected that the user will be prompted with a figure of the data after performing MFCC for PCG signals and Wavelet Transform for ECG signals. Here's the screenshots for the data visualization for feature extraction.

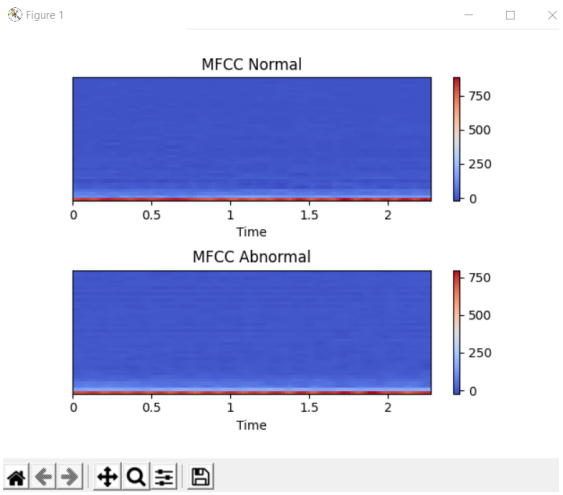


Figure 6.3. Screenshot of data visualization for MFCC feature extraction for CNN.

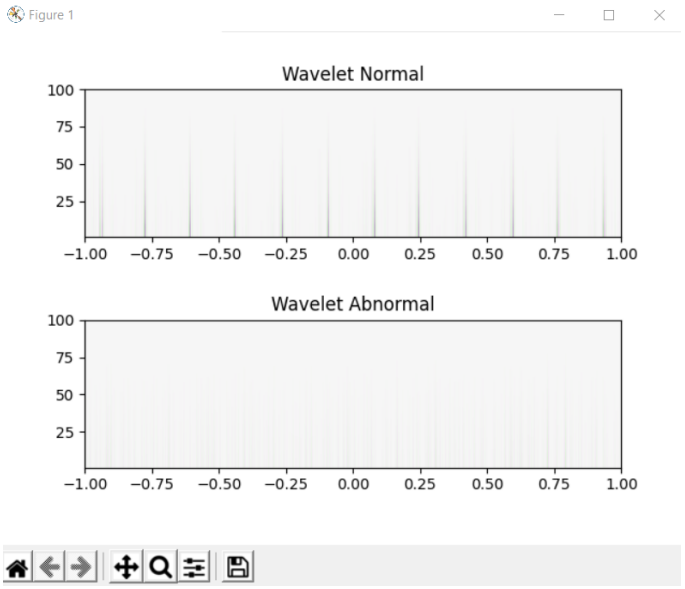


Figure 6.4. Screenshot of data visualization for CWT feature extraction for CNN.

After feature extraction, the feature-extracted data will be passed to the selected classifier by the user which is either LSTM or CNN. Once classification is complete for the specified signal type, the output would be plotted into a graph and shown to the user. The graph contains training loss versus validation loss and training accuracy versus validation accuracy. Here is the expected plotted graph below.

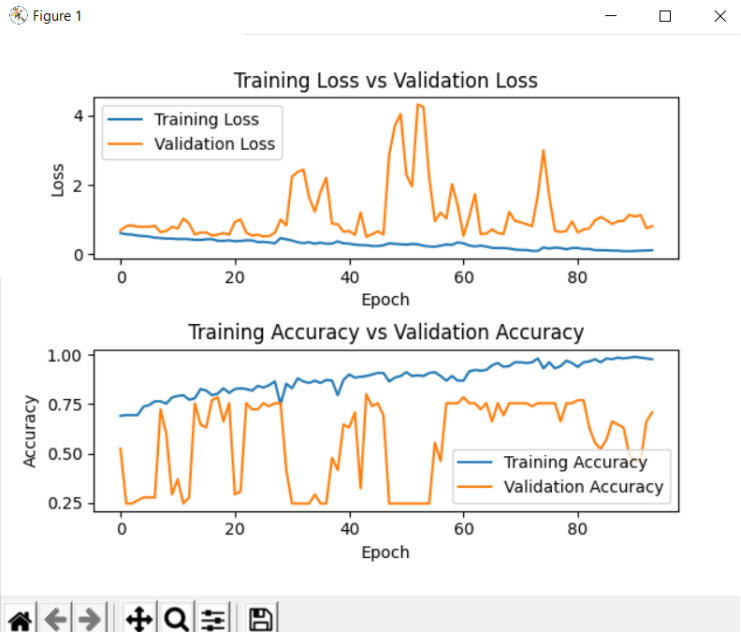


Figure 6.5. Screenshot of final output for both CNN and LSTM algorithms.

6.2 Summary and Discussion of Software Qualities that are Handled in the Software

6.2.1 Documentation and Maintainability

Documentation for Python has always been simple and easy with Python docstrings and comments for each function and complex codes to make interpreting our software easier. Python docstring gives us the flexibility to write a short description regarding our functions along with descriptions for each parameter that is used by the function and the description for the function's return value. The flexibility of Python docstrings is that it not only can be used for functions but also Python modules, classes and methods. To begin writing Python docstrings, they are declared by using triple double quotes or triple single quotes just below the class, method or function declaration. To ensure consistency and ease of readability for our software, we have documented each function of our system using Python docstrings. An example of Python docstring that we have implemented is as shown below for our MFCC feature extraction algorithm.

```
def visualize_mfcc(mfcc_data, labels):
    """
    Function that displays a spectrogram to visualize an
    MFCC series of PCG normal and abnormal signals.

    :param mfcc_data: The MFCC numpy array
    :param labels: The binary classification labels (normal/abnormal)
    :return: None
    """
```

Figure 6.6. Python docstring.

As for maintainability, as a degree to which our system can be understood, repaired or enhanced, overall has been decent but our software still has areas that can be further improved. From a repairability point of view, our software has shown that it has been fairly simple to troubleshoot with the help from Python's coding flexibility, correct coding practices and sufficient documentation. This allows us to efficiently make repairs or changes to code with ease.

6.2.2 Security

The definition of security for code is a practice of developing software with secure coding techniques in order to obliviate software vulnerabilities that could be exploited by hackers and untrusted sources through malicious code or application backdoors. For our project, security does not apply to us because our project does not deal or manage with private information between the user and software. This is due to our system only dealing with input data which in this case is ECG and PCG heart sounds and Signal Quality Index (SQI) numbers which are data that does not bring significant harm or privacy leak towards an individual. Hence, our project does not require an application of a security encryption as the

only input the user will be using are heart sounds data and the Signal Quality Index (SQI) of the heart sounds that do not hold information that can be used against the user or patient.

6.2.3 Robustness

Software robustness deemed by how an algorithm performs when training and testing with slightly different data as compared to the data we used for development. An example of testing robustness for our deep learning system is to use data that has more or less noise and observe the changes in accuracy ratings. From the development of our software, there is one limitation that could be highlighted that will affect our accuracy percentage which is the lack of noise reduction filters when preprocessing raw data. Due to this, if a dataset has a large amount of noise for ECG and PCG signals, it can be difficult to get consistent or high accuracy results for classifying if the patient's heart sounds normal or abnormal.

6.2.4 Usability

From a usability standpoint, using the command prompt for user input for specifying the files allows the user to manually input the file names to the Command Prompt which makes it easier for the user. Reason being he or she does not need to rename or upload files into our UI to run our program especially when the MATLAB folder file is large. However, from an intuitive standpoint for using the Command Prompt is that the user may not know how to initiate our program if the User and Technical Guide have not been revised by the user before using our program. This poses as one of the shortcomings of our software that not every single user with different backgrounds could understand how to use our software. Here is a figure showcasing the command-line along with a sample input.

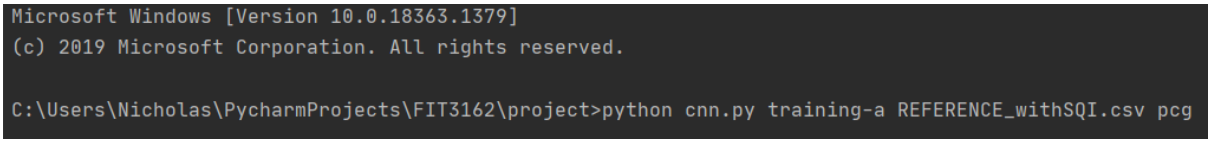


Figure 6.7. Command-line interface.

For result and data visualization, our program utilizes the *matplotlib* library's GUI to plot the output into graphs for users to manipulate and save. The graphs are shown as below for data and result visualization.

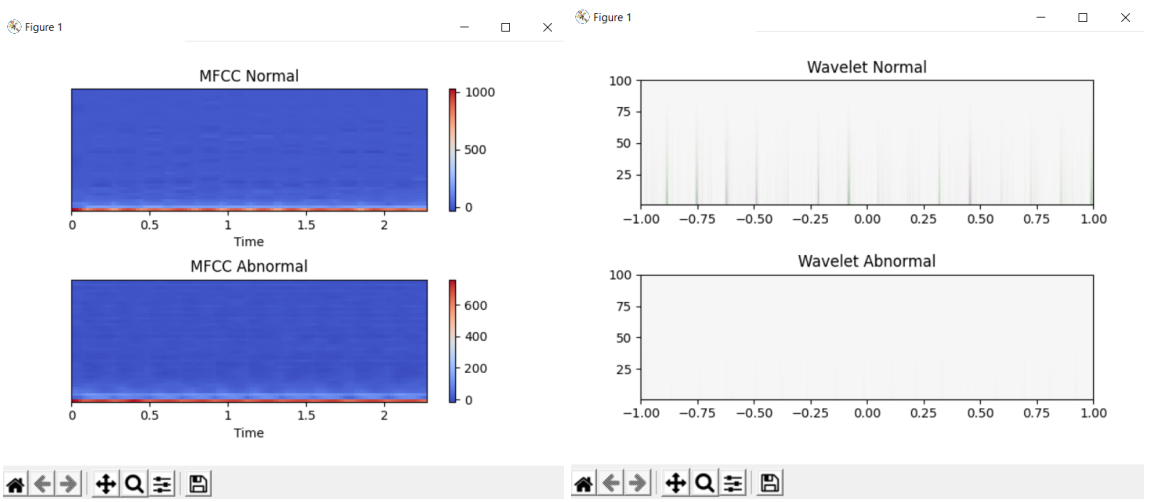


Figure 6.8. Graphical visualization of feature extractions.

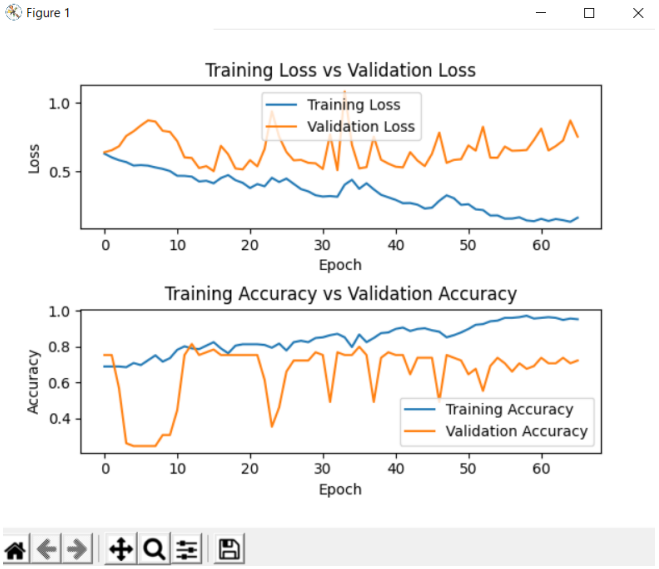


Figure 6.9. Subplots of training and validation accuracies and losses.

As shown from the figures, *matplotlib* library's plotting UI provides an array of tools for navigating the graphs that the user could use to highlight a specific section of the output. These tools include a home button to return to the graph's initial plot, back and forward buttons to navigate through user's changes, a pan button for moving the graph, a zoom button for zooming into the graph, a graph editor button and a save button to save the figure. Hence, this makes visualizing and observing our software's output much more intuitive to suit the user's needs and to ease the user's workload as he only does not need to additionally input other information for our software to get the finalized output. The only potential drawback to this visualization software is that it may take some time to familiarize with the tools and controls.

6.2.5 Scalability

Scalability for code is defined by the capability of the code to adapt to a larger scale dataset. For our project, scalability does not apply to us because for our project, the majority of our software is based on manipulating the raw ECG and PCG data specified from the UI which will produce a classification output after computation using just the Python programming language. Hence, even if a different dataset is used as input, our software does not utilize external calls or databases which does not give us any limitations towards server or network issues.

7 Critical Discussion

7.1 Reflections on Development

The team's initial proposed plan to tackle the project was to start from a simplified implementation of a machine learning model as our prototype design, then upgrade it to a deep learning model either through CNN or LSTM by pinpointing the similar areas between the two fields as highlighted from our project proposal. As development proceeded, the team became more competent on the concepts and further understood the scope of our project, allowing us to reconsider some prior propositions mentioned that yielded to be out-of-scope.

According to our proposed external design, we intended to develop a full stack web-based application using Flask's framework, consisting of full-fledged controls to tweak hyperparameters and graphical visualizations. In addition, we planned to store our dataset in a database and retrieve it through RESTful API services. However, it was later discovered that this was not part of the proposed scope introduced by our supervisor at all as the main objective was to develop and evaluate a deep learning model on its performance. This led to an agreement where we should simplify the frontend into a command-line interface due to the following reasons:

1. The dataset was fixed since it was provided to us by our supervisor. Storing it in an online database did not provide any benefit on enhancing classification performance, which was the main objective of the project. In terms of the concerns on storage capacity, our proposed hardware specifications were more than sufficient to handle the dataset locally. In fact, we had highlighted that the dataset was small, making it a performance bottleneck to our model, thus accentuating that we could still afford a larger dataset if possible.

2. Making API requests to retrieve large datasets poses the risk of rate limiting, and thus, goes against the motivation of building a performant preprocessing algorithm. Furthermore, utilizing a database would require additional monetary expenses and additional resources for continuous maintenance and upkeep.

3. We agreed that the benefits of trading off a more user-friendly application to focus more on improving the preprocessing steps and training model far outweigh the drawbacks. This showed when the team could afford additional time to debug the challenges of feature extraction to successfully implement it, improving the accuracy further to 75-82% accuracy for unstratified training and 70-72% for stratified training.

4. There were available options to display a GUI even if the program is run on the terminal. Some existing libraries include *matplotlib* and *pandas* that provide stable interactive figures to be charted out, thus enabling us to focus more towards the normalization of the dataset, and less on testing the correctness of the display.

As for the classification approach proposed, we managed to follow it closely whereby the key components to the structure of our system were properly developed — data preprocessing, feature extraction, classification and benchmarking. Furthermore, the proposal allowed room for experimentation in terms of the hyperparameters imposed and the preprocessing measures used. However, some ideas could not be fully realized due to time constraints and issues implementing MFCC which are further explained in the following section.

7.2 Problems Encountered

The team missed a few enhancements towards preprocessing which could potentially have resulted in a cleaner dataset to work with, that is the proposed 3 step noise filtering method by Rawther and Cheriyan (2015) on electromyographic noise removal and Butterworth high-pass and low-pass filterings. This is a result of a lack of understanding on noise filtering techniques and the delayed progress on MFCC where the team struggled to understand the approach to passing in time-series data as audio features. This is however compensated with the added implementation of CWT and the use of a Standard Scaler for normalization.

In addition, the team decided to instead focus on optimizing feature extractions further to make up for what was neglected, since the team had a better comprehension of feature extraction after the time spent working on it, we want to control what we could ascertain on the areas of improvement moving forward, such as to include first and second derivatives of MFCC as additional features to be classified. It would impose a greater risk if we were to work on implementing noise filters when we had not fully understood the approach given the remaining time we had left on development.

7.3 Achievements

Our biggest milestone was on the implementations of CNN and LSTM. This is due to a greater emphasis on reflecting our proposed literature reviews into our implementations as the development of the model was the crux of the entire project. To enable this, the timeline's duration of the classification module was adjusted after determining which other modules did not require as much time, such as the frontend modules.

In addition to benchmarking, we had further extended the coverage on performance testing instead of a basic summary of metrics calculated to enforce consistency on justifying the performance of our models. This is through additional manipulations of the learning rate in response to the training and validation accuracies. This process helped align our understanding of the project scope and to further determine what we could include as future work, such as a stratified k-fold cross validation to reduce overfitting, analysis on synchronous ECG and PCG signals, and noise removal filters.

8 Conclusion

In conclusion, with the increase of cardiovascular related diseases on a rise, from the development of our project, it showed the importance towards early detection. Cardiovascular diseases that represent 31% of global deaths reflect the severity of life-threatening diseases as heart diseases can be more difficult to detect than other major diseases such as cancer. To achieve building a successful model that could tackle this issue of predicting normal and abnormal heart diseases, we came up with our proposal and research towards machine learning for signal classification.

The research that we have conducted highlighted the aspects of what made an efficient and effective deep learning model which researchers have used a variety of methods towards different purposes. From here, we began on building our model based on the knowledge of our peers and from credible sources that focuses on deep learning which consists of a few main aspects of a deep learning model which are processing of data, feature extraction, data visualization and classification.

A methodology was developed after observing how credible researchers implemented their approaches towards their deep learning models. We implemented a waterfall model towards team management of tasks and development using different platforms such as GitLab, Discord and WhatsApp for sharing code implementation. Application of feature extraction algorithms such as MFCC and CWT after processing signal data has shown us how these complex algorithms work using data visualization as part of our project's objectives and has proved to significantly improve both our deep learning model's overall accuracy later on. Using the features extracted, we then passed it into our deep learning model of CNN and LSTM for automatic classification of normal and abnormal cardiac events which fulfills another project objective. Our models gave an accuracy score of an average of 75-82% classification accuracy for unstratified data and 70% with stratified data with CNN performing slightly better with more occurrences of obtaining a higher accuracy percentage. Hence, with the automatic detection system that we have developed for heart abnormality detection using deep learning, we were able to successfully compile a working system that could be useful as an auxiliary tool for heart abnormality detection using ECG and PCG signals.

9 Appendix

9.1 Appendix A: Team Members' Contribution

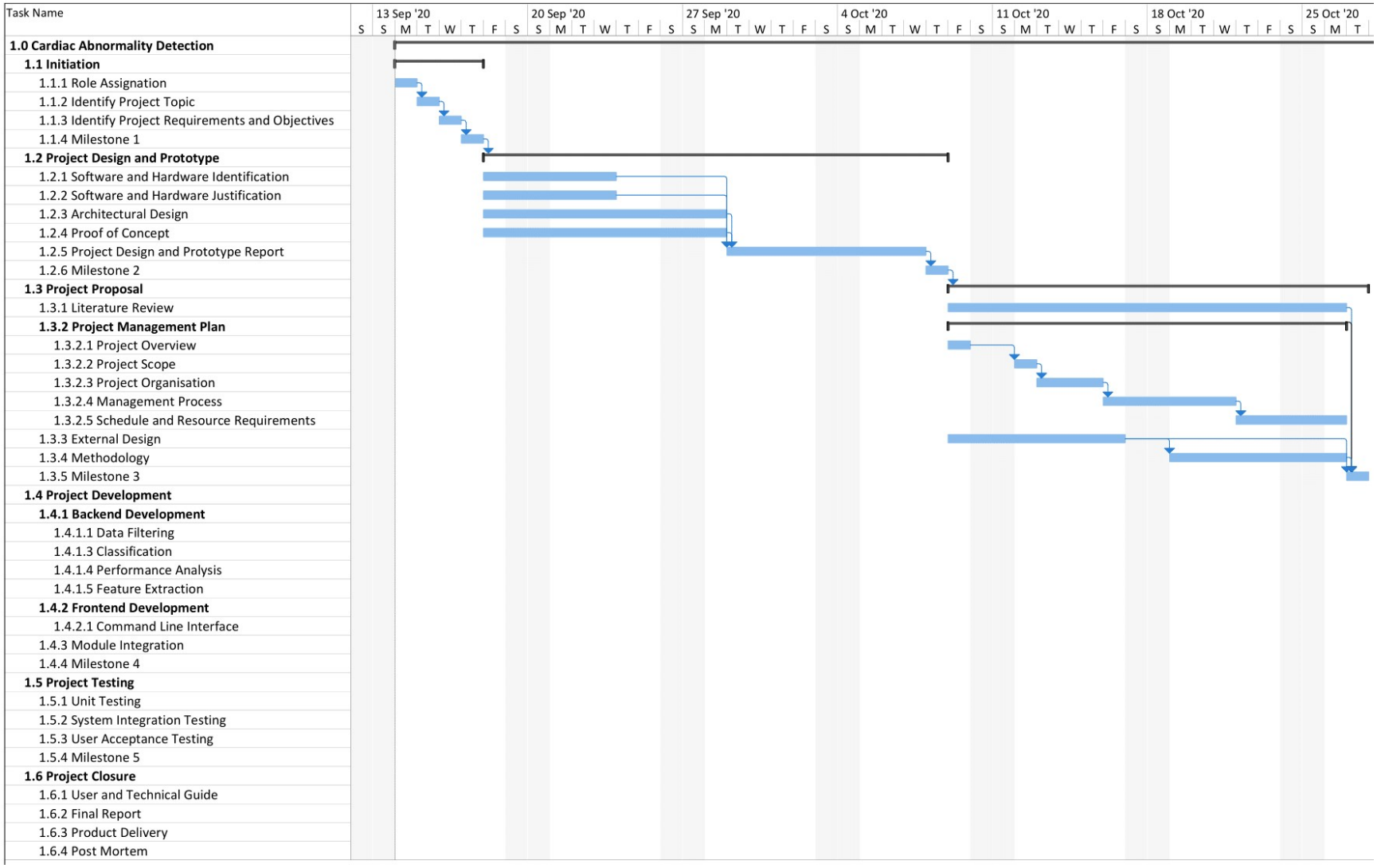
Deliverable	Section	Contributor	Percentage of Contribution
Software Deliverable	Data Preprocessing	Bryan Ho	70%
		Ryan Chow	30%
	Feature Extraction	Bryan Ho	50%
		Ryan Chow	50%
	Visualization	Bryan Ho	50%
		Ryan Chow	50%
	CNN	Bryan Ho	100%
	LSTM	Ryan Chow	100%
	Performance Metrics	Ryan Chow	90%
		Bryan Ho	10%
Test Report	Test Plan	Shen Min	100%
	Blackbox Testing	Shen Min	100%
	Whitebox Testing	Shen Min	50%
		Ryan Chow	50%
	System Testing	Shen Min	100%
	Performance Testing	Ryan Chow	100%
	Scalability and Usability	Shen Min	100%
	Test Limitations	Shen Min	100%
User and Technical Guide	End User Guide	Bryan Ho	70%
		Ryan Chow	30%
	Technical Guide	Bryan Ho	100%
Team Management Report	Reflection	Bryan Ho	100%
	Minutes of Meeting	Bryan Ho	100%

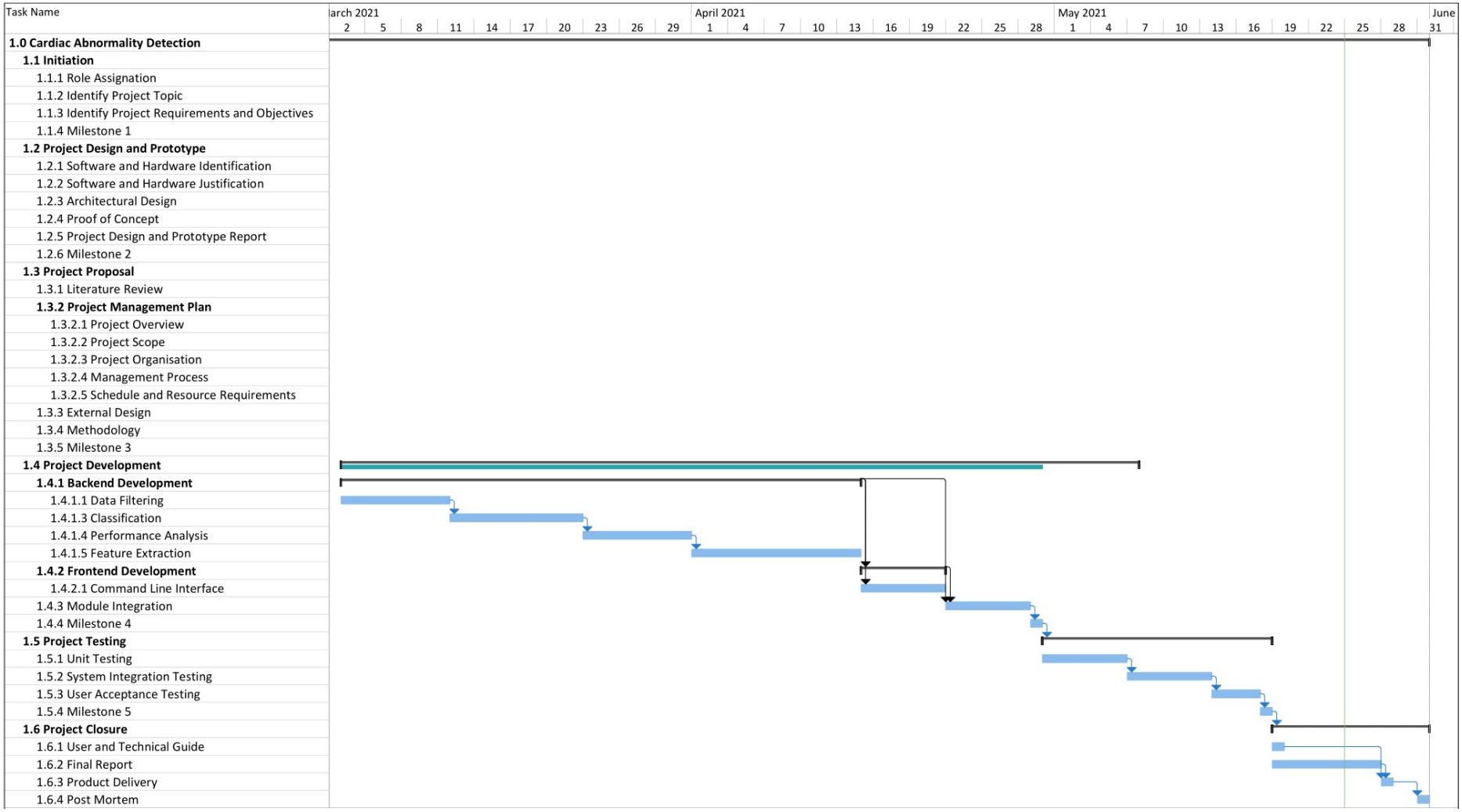
Final Project Report	Introduction	Shen Min	100%
	Background	Ryan Chow	100%
	Methodology	Bryan Ho	70%
		Ryan Chow	30%
	Project Management	Bryan Ho	100%
	Outcomes	Shen Min	20%
		Ryan Chow	80%
	Software Deliverable	Shen Min	100%
	Critical Discussion	Bryan Ho	100%
	Conclusion	Shen Min	100%

9.2 Appendix B: Work Breakdown Structure

Task Name	Duration	Start	Finish	Predecessors
1.0 Cardiac Abnormality Detection	**186 days**	**Mon 14/9/20**	**Mon 31/5/21**	
1.1 Initiation	**4 days**	**Mon 14/9/20**	**Thu 17/9/20**	
1.1.1 Role Assignation	1 day	Mon 14/9/20	Mon 14/9/20	
1.1.2 Identify Project Topic	1 day	Tue 15/9/20	Tue 15/9/20	3
1.1.3 Identify Project Requirements and Objectives	1 day	Wed 16/9/20	Wed 16/9/20	4
1.1.4 Milestone 1	1 day	Thu 17/9/20	Thu 17/9/20	5
1.2 Project Design and Prototype	**15 days**	**Fri 18/9/20**	**Thu 8/10/20**	**6**
1.2.1 Software and Hardware Identification	4 days	Fri 18/9/20	Wed 23/9/20	
1.2.2 Software and Hardware Justification	4 days	Fri 18/9/20	Wed 23/9/20	
1.2.3 Architectural Design	7 days	Fri 18/9/20	Mon 28/9/20	
1.2.4 Proof of Concept	7 days	Fri 18/9/20	Mon 28/9/20	
1.2.5 Project Design and Prototype Report	7 days	Tue 29/9/20	Wed 7/10/20	8,9,10,11
1.2.6 Milestone 2	1 day	Thu 8/10/20	Thu 8/10/20	12
1.3 Project Proposal	**13 days**	**Fri 9/10/20**	**Tue 27/10/20**	**13**
1.3.1 Literature Review	12 days	Fri 9/10/20	Mon 26/10/20	
1.3.2 Project Management Plan	**12 days**	**Fri 9/10/20**	**Mon 26/10/20**	
1.3.2.1 Project Overview	1 day	Fri 9/10/20	Fri 9/10/20	
1.3.2.2 Project Scope	1 day	Mon 12/10/20	Mon 12/10/20	17
1.3.2.3 Project Organisation	3 days	Tue 13/10/20	Thu 15/10/20	18
1.3.2.4 Management Process	4 days	Fri 16/10/20	Wed 21/10/20	19
1.3.2.5 Schedule and Resource Requirements	3 days	Thu 22/10/20	Mon 26/10/20	20
1.3.3 External Design	6 days	Fri 9/10/20	Fri 16/10/20	
1.3.4 Methodology	6 days	Mon 19/10/20	Mon 26/10/20	22
1.3.5 Milestone 3	1 day	Tue 27/10/20	Tue 27/10/20	15,16,22,23
1.4 Project Development	**48 days**	**Wed 3/3/21**	**Fri 7/5/21**	
1.4.1 Backend Development	**31 days**	**Wed 3/3/21**	**Wed 14/4/21**	
1.4.1.1 Data Filtering	7 days	Wed 3/3/21	Thu 11/3/21	
1.4.1.3 Classification	7 days	Fri 12/3/21	Mon 22/3/21	27
1.4.1.4 Performance Analysis	7 days	Tue 23/3/21	Wed 31/3/21	28
1.4.1.5 Feature Extraction	10 days	Thu 1/4/21	Wed 14/4/21	29
1.4.2 Frontend Development	**5 days**	**Thu 15/4/21**	**Wed 21/4/21**	**26**
1.4.2.1 Command Line Interface	5 days	Thu 15/4/21	Wed 21/4/21	26
1.4.3 Module Integration	5 days	Thu 22/4/21	Wed 28/4/21	26,31
1.4.4 Milestone 4	1 day	Thu 29/4/21	Thu 29/4/21	33
1.5 Project Testing	**13 days**	**Fri 30/4/21**	**Tue 18/5/21**	**34**
1.5.1 Unit Testing	5 days	Fri 30/4/21	Thu 6/5/21	
1.5.2 System Integration Testing	5 days	Fri 7/5/21	Thu 13/5/21	36
1.5.3 User Acceptance Testing	2 days	Fri 14/5/21	Mon 17/5/21	37
1.5.4 Milestone 5	1 day	Tue 18/5/21	Tue 18/5/21	38
1.6 Project Closure	**9 days**	**Wed 19/5/21**	**Mon 31/5/21**	**39**
1.6.1 User and Technical Guide	1 day	Wed 19/5/21	Wed 19/5/21	
1.6.2 Final Report	7 days	Wed 19/5/21	Thu 27/5/21	
1.6.3 Product Delivery	1 day	Fri 28/5/21	Fri 28/5/21	41,42
1.6.4 Post Mortem	1 day	Mon 31/5/21	Mon 31/5/21	43

9.3 Appendix C: Gantt Chart




Task Name
arch 2021
April 2021
May 2021
June
2 5 8 11 14 17 20 23 26 29 1 4 7 10 13 16 19 22 25 28 1 4 7 10 13 16 19 22 25 28 31
1.0 Cardiac Abnormality Detection
1.1 Initiation
1.1.1 Role Assignation
1.1.2 Identify Project Topic
1.1.3 Identify Project Requirements and Objectives
1.1.4 Milestone 1
1.2 Project Design and Prototype
1.2.1 Software and Hardware Identification
1.2.2 Software and Hardware Justification
1.2.3 Architectural Design
1.2.4 Proof of Concept
1.2.5 Project Design and Prototype Report
1.2.6 Milestone 2
1.3 Project Proposal
1.3.1 Literature Review
1.3.2 Project Management Plan
1.3.2.1 Project Overview
1.3.2.2 Project Scope
1.3.2.3 Project Organisation
1.3.2.4 Management Process
1.3.2.5 Schedule and Resource Requirements
1.3.3 External Design
1.3.4 Methodology
1.3.5 Milestone 3
1.4 Project Development
1.4.1 Backend Development
1.4.1.1 Data Filtering
1.4.1.3 Classification
1.4.1.4 Performance Analysis
1.4.1.5 Feature Extraction
1.4.2 Frontend Development
1.4.2.1 Command Line Interface
1.4.3 Module Integration
1.4.4 Milestone 4
1.5 Project Testing
1.5.1 Unit Testing
1.5.2 System Integration Testing
1.5.3 User Acceptance Testing
1.5.4 Milestone 5
1.6 Project Closure
1.6.1 User and Technical Guide
1.6.2 Final Report
1.6.3 Product Delivery
1.6.4 Post Mortem

9.4 Appendix D: Responsibility Assignment Matrix

Tasks	Project Manager	Technical Lead	Quality Assurance	Supervisor
Requirements	R	A	I	C
Software and Hardware	A	R	I	C
Architectural Design	A	R	I	C
Proof of Concept Code	A	R	R	C
Project Design Report	R	A	I	C
Literature Review	A	R	R	C
Project Management Plan	A	I	I	C
Scope Statement	R	A	C	C
Requirements Traceability Matrix	R	A	C	C
Risk Register	R	A	R	C
Work Breakdown Structure	A	C	C	I
Meeting Minutes	A	I	I	I
Gantt Chart	A	C	C	I
Communications Matrix	A	C	C	I
Methodology	I	R	A	C
Test Plan	I	A	R	I
Test Coverage	I	A	R	I
Test Cases	I	A	R	I
Software Installation	I	R	A	C
Version Control	I	A	R	I
Data Filtering	I	R	A	C
Feature Extraction	I	R	A	C
Classification	I	R	A	C
Evaluation and Analysis	I	R	A	C

Application Development	I	R	A	C
Code Documentation	I	R	A	I
User and Technical Guide	I	A	C	I
Unit Testing	I	A	R	I
System Integration Testing	I	A	R	I
User Acceptance Testing	I	A	R	C
Test Report	I	A	R	I
Code Review	I	A	R	I
Proofreading	A	I	R	I
License Purchase	A	R	I	C
Presentation	R	A	I	I
Post Mortem	R	A	I	I

9.5 Appendix E: Risk Register

No.	Rank	Risk	Description	Category	Root Cause	Triggers	Potential Responses	Risk Owner	Probability	Impact	Score	Status
R1	1	Time constraints causing project delays	Team members are unable to deliver their assigned responsibilities within the allocated time	Schedule	Team members struggling to fully commit to the project due to other commitments	Heavy workload incurred from other undertaking units	Communicate to work on improving time management and assign flexible schedules for less critical tasks	Project Manager	10	10	100	Open
R2	4	Change in project requirements	Change in demands or additional requirements requested by the client to be made to the project scope	Scope	Project scope not clearly defined at the beginning of development	Client requests for requirement changes during development	Discuss with the client to to fit the new requirements into the timeline and negotiate on the impact of the additions	Project Owner	5	8	40	Open
R3	2	Inexperience in utilising certain technologies	Team members facing unfamiliarities with certain softwares or programming paradigms resulting in technical debt	Training	Lack of training provided beforehand to ensure confidence in the team's knowledge	Team member unable to contribute when provided with new technologies	Consult client for advice on the utilisation of unfamiliar software and request for relevant learning resources	Technical Lead	8	8	64	Open
R4	8	Project ignored due to external obligations	Team members occupied with their internship duties	Schedule	University requires all IT students to undertake an internship programme	Team members prioritise internship work over the project	Discuss amongst the team on workload redistribution and realign schedules	Project Manager	1	10	10	Open
R5	9	Unable to meet with client	Client is busy on certain scheduled dates agreed upon prior	Human	Client preoccupied with other responsibilities	Client inform the team on sudden urgent duties to attend to	Arrange another date to meet or set up another communication platform as a contingency plan	Project Manager	3	3	9	Open
R6	5	Major functions not performing adequately	Important functionalities still indicating errors or bugs despite passing the testing phase	Technical	Inadequate test cases are written that provides thorough coverage	Unexpected bug found after project testing	Fall back to the testing phase when necessary through feedback loops to further improve the test cases	Quality Assurance	4	6	24	Open

R7	7	Classification performance not up to expectations	Performance of classification model developed is slower than usual	Technical	Overheads incurred from poor programming practices that increases the time complexity of the algorithm	Actual performance calculated is worse than the predicted values	Reinvestigate the code during development phase through feedback loops and client advice	Quality Assurance	2	6	12	Open
R8	3	Integration difficulties between modules	Issues in integrating front-end modules with back-end modules	Technical	Unfamiliarities or rigidities in the tech stacks used and their respective integration methods	Tech stacks used do not provide the required libraries or dependencies to integrate between modules	Explore the alternatives stated in the project design to test their feasibility	Technical Lead	6	9	54	Open
R9	6	Additional budget allocated to the project	Client increasing the allocated budget to enable purchasing additional resources for the project	Positive	Client is aware of the project limitations in terms of budget constraints	Features cannot be implemented due to lack of required resources	Use budget surplus for necessary resource procurements when facing requirement changes	Project Manager	2	8	16	Open
R10	10	Reduction in project scope	Client decides to remove certain requirements due to it being of lower necessity	Positive	Client has a change of mind and lowers the expectations of deliverables	Change of requirements requested by the client	Readjust timeline for the remaining requirements that need to be developed	Project Manager	1	8	8	Open

9.6 Appendix F: Source Code

Pre-processing:

```
def read_mat(folder_name, sqi_csv, signal_type):
    """
    Function that reads the MATLAB files given and generates the numpy arrays
    for ECG/PCG data and the labels.

    The MATLAB files must be in the root of the folder's directory.
    The SQI CSV file must have the following headers as its structure:
    - Data name
    - Label -> -1: Normal, 1: Abnormal
    - Signal Quality Index (SQI) -> 0: Noisy, 1: Clean

    :return: A tuple of (mat_data, data_labels) where
                mat_data - 3D numpy array of ECG/PCG data
                labels - 1D numpy array of labels
    """
    check_folder = f"./{folder_name}"
    check_csv = f"./{sqi_csv}"
    assert os.path.isdir(check_folder), "Dataset folder specified is not a folder"
    assert os.path.isfile(check_csv), "SQI CSV file specified is not a csv file"
    assert signal_type == "ecg" or signal_type == "pcg", "Specified signal type does not exist. PCG/ECG only."
    # 0 for PCG, 1 for ECG.
    # Change this to switch between classifying different signal types.
    signal_type_index = set_signal_type_index(signal_type)
    # Search for all MAT files in folder_name.
    # Each subject has 2 rows. The first row is PCG (heart sound) and the second row is ECG.
    directory = f"./{folder_name}/*.mat"
    list_files = glob.glob(directory)
    # Load the demographic information from CSV file.
    demo = pd.read_csv(sqi_csv)
    assert len(list_files) >= 2 and len(demo) >= 2, \
        "Must have at least 2 subjects -- 1 normal, 1 abnormal."
    # Filter off the signals that do not have signals greater than the discard threshold.
    filtered_files, signals = remove_signals(list_files, signal_type_index)
    # Get shortest length and time length.
    shortest_length, time_length = find_shortest_length(signals)
    # Preprocess MATLAB ECG/PCG data and binary classification labels.
    mat_data = preprocess_signals(signals, time_length, signal_type_index)
    data_labels = preprocess_labels(filtered_files, demo)
    print_results(list_files, mat_data, data_labels, shortest_length, time_length)
    validate_data(mat_data, data_labels)

    return mat_data, data_labels
```

Feature Extraction:

```python
def mfcc(pcg_data, sample_rate, n_mfcc, hop_length):
    """
    Function that uses Librosa for MFCC feature extraction on PCG signals.
    Referenced from: https://librosa.org/doc/main/generated/librosa.feature.mfcc.html

    :param pcg_data: The MATLAB PCG data
    :param sample_rate: The sample rate of the PCG data
    :param n_mfcc: The number of MFCCs to return
    :param hop_length: Sliding window length
    :return: A 2D numpy matrix of MFCC coefficients of dimensions: (n_mfcc, L)
             where L is the total number of sliding windows (i.e., len(pcg_data)/hop_length)
    """
    assert type(pcg_data) == np.ndarray, "array must be a numpy array"
    assert pcg_data.size != 0, "numpy array must not be empty"

    return librosa.feature.mfcc(pcg_data, sr=sample_rate, n_mfcc=n_mfcc, hop_length=hop_length)


def wavelet_transform(ecg_data, widths):
    """
    Function that uses SciPy for Wavelet Transform feature extraction on ECG signals.
    Referenced from: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.cwt.html

    Wavelet function: Ricker Wavelet / Mexican Hat Wavelet
    Referenced from: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.ricker.html#scipy.signal.ricker

    :param ecg_data: The MATLAB ECG data
    :param widths: Widths to use for the transform
    :return: A 2D numpy matrix of Wavelet coefficients of dimensions: (M, N)
             where M = len(np.arange(1, widths + 1))
                   N = len(ecg_data)
    """
    assert type(ecg_data) == np.ndarray, "array must be a numpy array"
    assert ecg_data.size != 0, "numpy array must not be empty"

    return signal.cwt(ecg_data, signal.ricker, np.arange(1, widths + 1))
```

Implement_model:

1) CNN

```python
def make_cnn_model(input_shape, num_conv_layers, num_classes):
    """
    Function that makes a Fully Convolutional Neural Network proposed in this paper:
    https://arxiv.org/abs/1611.06455.

    The implementation is based on the following link:
    https://github.com/hfawaz/dl-4-tsc/

    :param input_shape: A shape tuple (integers)
    :param num_conv_layers:
    :param num_classes: The number of classes
    :return: A Keras CNN model
    """
    assert type(input_shape) is tuple, "Input_shape is not a tuple"
    assert num_classes >= 1, "num_classes must not be empty"

    input_layer = keras.layers.Input(input_shape)
    current_layer = input_layer

    for i in range(num_conv_layers):
        current_layer = conv(current_layer)

    gap = keras.layers.GlobalAveragePooling1D()(current_layer)

    output_layer = keras.layers.Dense(num_classes, activation="sigmoid")(gap)

    return keras.models.Model(inputs=input_layer, outputs=output_layer)
```

2) LSTM

```python
def build_lstm(data_shape, config={}):
    if 'optimizer' not in config:
        raise Exception('key: optimizer, not found in config')
    if 'learning_rate' not in config:
        raise Exception('key: learning_rate, not found in config')
    if 'loss' not in config:
        raise Exception('key: loss, not found in config')
    if 'metrics' not in config:
        raise Exception('key: metrics, not found in config')

    if len(data_shape) != 3:
        raise Exception('expected 3D model input data shape, got', data_shape)

    # Build single layer LSTM network
    model = Sequential()
    # model for PCG
    # model.add(LSTM(128, input_shape=(data_shape[1], data_shape[2]))) # if want add more layer, put in first layer return_sequences=True
    model.add(LSTM(128, input_shape=(data_shape[1], data_shape[2]), return_sequences=True))
    # model.add(Dense(32))
    # model.add(Dropout(0.6))
    model.add(LSTM(128, return_sequences=True))
    # model.add(Dropout(0.6))
    model.add(LSTM(256))
    model.add(Dropout(0.6))
    # model.add(Dropout(0.3))
    # model.add(LSTM(100, return_sequences=True))
    # model.add(LSTM(100))
    model.add(Dense(32, activation='relu'))
    # model.add(Dropout(0.3))
    model.add(Dense(1, activation='sigmoid')) # softmax for classification, for this need encode labels in binary (one hot encoding)
    # for PCG using MFCCs
    optimizer_fn = config['optimizer']
    optimizer = optimizer_fn(config['learning_rate'])
    model.compile(loss=config['loss'],
                  optimizer=optimizer,
                  metrics=config['metrics'])
    return model
```

Output visualization:

```python
def visualize_cnn_accuracy_loss(history):
    """
    Function that displays a graph of the training and validation
    accuracy/loss after evaluating the CNN model.

    :param history: The History object from Keras after training the model.
    :return: None
    """
    fig, (ax1, ax2) = plt.subplots(2)
    fig.tight_layout(pad=3.5)

    # Plot model's training and validation loss.
    ax1.plot(history.history['loss'])
    ax1.plot(history.history['val_loss'])
    ax1.set_title('Training Loss vs Validation Loss')
    ax1.set(xlabel='Epoch', ylabel='Loss')
    ax1.legend(['Training Loss', 'Validation Loss'], loc='best')

    # Plot model's training and validation accuracy.
    ax2.plot(history.history['sparse_categorical_accuracy'])
    ax2.plot(history.history['val_sparse_categorical_accuracy'])
    ax2.set_title('Training Accuracy vs Validation Accuracy')
    ax2.set(xlabel='Epoch', ylabel='Accuracy')
    ax2.legend(['Training Accuracy', 'Validation Accuracy'], loc='best')

    plt.show()
    plt.close()
```

10 References

Lewis, S. (2019, February 07). What is waterfall model? - definition from whatis.com. Retrieved May 25, 2021, from https://searchsoftwarequality.techtarget.com/definition/waterfall-model

Horvath, I. (2020). Five Steps of Risk Management Process. Retrieved from https://www.invensislearning.com/blog/risk-management-process-steps/

World Health Organization: WHO. (2017, May 17). *Cardiovascular diseases (CVDs)*. World Health Organization. https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)

Ng, K., Steinhubl, S. R., deFilippi, C., Dey, S., & Stewart, W. F. (2016). Early Detection of Heart Failure Using Electronic Health Records. *Circulation: Cardiovascular Quality and Outcomes, 9*(6), 649–658. https://doi.org/10.1161/circoutcomes.116.002797

Ismail, S., Siddiqi, I., & Akram, U. (2018). Localization and classification of heart beats in phonocardiography signals —a comprehensive review. *EURASIP Journal on Advances in Signal Processing*, 2018(1). doi:10.1186/s13634-018-0545-9

Roy, Y., Banville, H., Albuquerque, I., Gramfort, A., Falk, T. H., & Faubert, J. (2019). Deep learning-based electroencephalography analysis: a systematic review. *Journal of neural engineering, 16*(5), 051001.

Wang, T., Lu, C., Sun, Y., Yang, M., Liu, C., & Ou, C. (2021). Automatic ECG Classification Using Continuous Wavelet Transform and Convolutional Neural Network. *Entropy, 23*(1), 119.

DeepAI. (2019, May 17). Feature extraction. Retrieved May 26, 2021, from https://deepai.org/machine-learning-glossary-and-terms/feature-extraction

Deng, M., Meng, T., Cao, J., Wang, S., Zhang, J., & Fan, H. (2020). Heart sound classification based on improved MFCC features and convolutional recurrent neural networks. *Neural Networks, 130*, 22-32.

Chowdhury, T. H., Poudel, K. N., & Hu, Y. (2020). Time-Frequency analysis, Denoising, Compression, segmentation, and classification of PCG Signals. *IEEE Access, 8*, 160882-160890. doi:10.1109/access.2020.3020806

Li, S., Li, F., Tang, S., & Xiong, W. (2020). A Review of Computer-Aided Heart Sound Detection Techniques. *BioMed research international*, *2020*, 5846191. https://doi.org/10.1155/2020/5846191

Tan, Y., & Du, L. (2009). Study on wavelet transform in the processing for ecg signals. *2009 WRI World Congress on Software Engineering*. doi:10.1109/wcse.2009.89

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, *33*(4), 917-963.

Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

Pandey, S. K., & Janghel, R. R. (2020). Automatic arrhythmia recognition from electrocardiogram signals using different feature methods with long short-term memory network model. *Signal, Image and Video Processing*, *14*(6), 1255-1263.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. *arXiv preprint arXiv:1705.08292*.

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Rawther, N. N., & Cheriyan, J. (2015). Detection and classification of cardiac arrhythmias based on ecg and pcg using temporal and wavelet features. *Int. J. Adv. Res. Comput. Commun. Eng*, *4*(4), 474-479.