Automatic Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence

BY

KOH ZI KANG

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

JUN 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: Automatic Detection of Myocardial Infarction (MI) using ECG signals
with Artificial Intelligence_____________________________________
____

Academic Session: ______JUN 2024_______

I ______KOH ZI KANG____
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_________KOH________________
(Author's signature)

____
(Supervisor's signature)

Address:
_1910, Jalan Berlian 1________
_Taman Bandar Baru_________
_31900 Kampar, Perak________

Dr. Mogana Vadiveloo
____
Supervisor's name

Date: __~~2~~ 13 / 9 / 2024__________

Date: 13/09/2024

Universiti Tunku Abdul Rahman			
Form Title : **Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY/INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

13
Date: ____~~2~~ / 9 / 2024_____

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ***KOH ZI KANG*** (ID No: ***22ACB00550***) has completed this final year project/ dissertation/ thesis* entitled "Automatic Detection of Myocardial Infarction (MI) using ECG signals with Artificial Intelligence" under the supervision of Dr Mogana a/p Vadiveloo (Supervisor) from the Department of Computer Science, Faculty/Institute* of Information and Communication Technology.

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

_____KOH ZI KANG_______________
(*Student Name*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**AUTOMATIC DETECTION OF MYOCARDIAL INFARCTION (MI) USING ECG SIGNALS WITH ARTIFICIAL INTELLIGENCE**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : ______KOH___________________

Name : __________KOH ZI KANG_______________

Date : ______2 / 9 / 2024___________________

ACKNOWLEDGEMENTS

I would like to express thanks and appreciation to my former and current supervisors, Dr Ashvaany a/p Egambaram and Dr Mogana a/p Vadiveloo who have given me a golden opportunity to engage in a signal processing and an artificial intelligence model design project. It is my first step to establishing a career in artificial intelligence. A million thanks to both of you.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

Myocardial Infarction (MI) is known as heart attack, it is one of the most life-threatening cardiovascular diseases. During infarction, the coronary artery which is responsible for the delivery of blood, oxygen and nutrients, is fully or partially blocked, and the heart muscle will die of ischemia. Percutaneous Coronary Intervention (PCI) is a nonsurgical technique to treat MI, the faster the patient receives PCI treatment, the higher the survival rate. The heart activity (pumping blood) is controlled by the electrical current generated by itself, therefore 12-lead electrocardiogram is an excellent tool to capture the activity of the heart, the pattern of a complete heart cycle is referred to as the PQRST cycle. MI will cause a morphological change to this pattern, therefore this can be used to diagnose the MI. In order to avoid the intra-/inter-observer effect caused by manual human interpretation, many researchers proposed machine-learning-based methods and then nowadays many deep-learning-based methods have emerged to perform automatic and end-to-end classification. Nevertheless, many studies that emphasized deep learning models did not care about the data split method during their experiment, this led to a misleadingly supreme performance due to information leakage problem. The models might be trained to memorize which subjects have MI heartbeats instead of learning the features related to the disease itself from the amplitude and time (in a sequential model). Thus, this research proposed three models: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) and 1-dimensional Convolutional Neural Network (1D-CNN) with the implementation of intra-patient and inter-patient data split techniques. In FYP 2, the architecture for each model had been improved compared to FYP 1 to augment the performance, then regularization and dropout techniques were applied to increase the generalization ability and finally, one transformer model had been developed to test its potential in processing ECG signal. From the perspective of the inter-patient method, the LSTM model obtained 90.53% accuracy, while the 1D-CNN and GRU models obtained 85.82% and 86.65% accuracy respectively. On the other hand, for all the intra-patient models, LSTM and GRU obtained a similar 95.4% accuracy while 1D-CNN obtained a 97.68% accuracy. The transformer model achieved 82.28% and 91.15% in intra-patient and inter-patient analysis. Obviously, this has proven that the intra-patient models can produce a misleadingly high result. Another dataset is obtained from the open database on the Internet, but unfortunately, the testing result has shown that all of the models failed to generalize.

TABLE OF CONTENTS

TITLE PAGE i

REPORT STATUS DECLARATION FORM ii

FYP THESIS SUBMISSION FORM iii

DECLARATION OF ORIGINALITY iv

ACKNOWLEDGEMENTS v

ABSTRACT vi

TABLE OF CONTENTS vii

LIST OF FIGURES xii

LIST OF TABLES xviii

LIST OF ABBREVIATIONS xix

CHAPTER 1 INTRODUCTION 1

1.1 Problem Statement and Motivation 5

1.2 Objectives 7

1.3 Project Scope and Direction 7

1.4 Contributions 7

1.5 Report Organization 7

CHAPTER 2 LITERATURE REVIEW **9**
2.1 Previous Works on Deep Learning 9
2.1.1 Near real-time single-beat myocardial infarction detection from single-lead electrocardiogram using Long Short-Term Memory Neural Network 9
2.1.2 Deep Learning Networks Accurately Detect ST-Segment Elevation Myocardial Infarction and Culprit Vessel 14
2.1.3 Evolution of single-lead ECG for STEMI detection using a deep learning approach 19
2.1.4 Early detection of ST-segment elevated myocardial infarction by artificial intelligence with 12-lead electrocardiogram 21
2.1.5 Application of CNN for Detection and Localization of STEMI Using 12-Lead ECG Images 24
2.1.6 Multiclass classification of myocardial infarction with convolutional and recurrent neural networks for portable ECG devices 27
2.1.7 A New Automatic Approach to Distinguish Myocardial Infarction Based on LSTM 30
2.1.8 Deep Learning with Long Short-Term Memory for Enhancement Myocardial Infarction Classification 31
2.1.9 Classification of myocardial infarction with multi-lead ECG signals and deep CNN 33
2.1.10 Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals 35
2.2 Summary 36

CHAPTER 3 SYSTEM DESIGN & MODEL **39**
3.1 System Design Diagram 39
3.2 Model Architecture Diagram 41
3.2.1 LSTM classifier 41

3.2.2 GRU classifier 42
3.2.3 1D-CNN classifier 44
3.2.4 Transformer classifier 45
3.3 Performance Metrics 46
3.3.1 Accuracy 46
3.3.2 Sensitivity 46
3.3.3 Precision 47
3.3.4 F1-Score 47

CHAPTER 4 IMPLEMENTATION & EXPERIMENTAL RESULTS 48
4.1 Hardware & Software Setup 48
4.1.1 Hardware 48
4.1.2 Software 48
4.2 Data Collection 48
4.3 Data Pre-Processing 50
4.3.1 Intra-patient splitting 51
4.3.2 Inter-patient splitting 52
4.3.3 Denoise & Baseline Wandering Removal 54
4.3.4 R-Peak Detection & Segmentation 55
4.3.5 Transformation 57
4.4 Network Architecture 58
4.5 Helpers 61
4.5.1 Training Loop 61
4.5.2 Early Stop Mechanism 62
4.5.3 Testing 64

4.6 Main function 65
4.7 Visualization 69
4.8 Training & Validation Performance Metrics 72
4.8.1 Visualization 73
4.8.1.1 Intra-Patient Uni-Directional LSTM 74
4.8.1.2 Inter-Patient Uni-Directional LSTM 75
4.8.1.3 Intra-Patient Bi-Directional LSTM 77

4.8.1.4 Inter-Patient Bi-Directional LSTM 78
4.8.1.5 Intra-Patient Uni-Directional GRU 80
4.8.1.6 Inter-Patient Uni-Directional GRU 81
4.8.1.7 Intra-Patient Bi-Directional GRU 83
4.8.1.8 Inter-Patient Bi-Directional GRU 84
4.8.1.9 Intra-Patient 1D-CNN 86
4.8.1.10 Inter-Patient 1D-CNN 87
4.8.1.11 Intra-Patient Transformer 89
4.8.1.12 Inter-Patient Transformer 90

CHAPTER 5 SYSTEM EVALUATION AND DISCUSSION 92

5.1 Overview of Testing Performance Metrics of Classifiers 92
5.2 Visualization 94
5.2.1 Intra-Patient Uni-Directional LSTM 94
5.2.2 Inter-Patient Uni-Directional LSTM 94
5.2.3 Intra-Patient Bi-Directional LSTM 95
5.2.4 Inter-Patient Bi-Directional LSTM 95
5.2.5 Intra-Patient Uni-Directional GRU 96
5.2.6 Inter-Patient Uni-Directional GRU 96
5.2.7 Intra-Patient Bi-Directional GRU 97
5.2.8 Inter-Patient Bi-Directional GRU 97
5.2.9 Intra-Patient 1D-CNN 98
5.2.10 Inter-Patient 1D-CNN 98
5.2.11 Intra-Patient Transformer 99
5.2.12 Inter-Patient Transformer 99
5.3 Challenges 99
5.4 Objectives Evaluation 100

CHAPTER 6 CONCLUSION AND LIMITATIONS **101**
6.1 Conclusion 101
6.2 Limitations 101

REFERENCES **103**
APPENDIX **106**
WEEKLY LOG **107**
POSTER **113**
PLAGIARISM CHECK RESULT **114**
FYP2 CHECKLIST **125**

LIST OF FIGURES

Figure Number	Title	Page
Figure 1.1	Statistics on diabetes, hypertension and high cholesterol in Malaysia	1
Figure 1.2	Prevalence of CVD risk factor in Malaysia	2
Figure 1.3	Statistics on causes of death in Malaysia	2
Figure 1.4	Visualization of coronary artery blockage which induces the morphological change of ECG signal	4
Figure 1.5	The names of different phases during the total ischaemic time	4
Figure 1.6	The performance of humans in different medical resource levels	5
Figure 2.1	Architecture of LSTM model	10
Figure 2.2	Performance of LSTM model optimized using different metrics: Accuracy and J-measure	10
Figure 2.3	Evolution of performance of inter-patient classifier based on optimization of accuracy and J-measure metrics	11
Figure 2.4	Evolution of performance of intra-patient (beat-split) classifier	12
Figure 2.5	Experimental result to test if the intra-patient classifier will memorize the patients	13
Figure 2.6	Relationship of MI detection rate and MI location	14
Figure 2.7	Comparison of performance between the proposed model and other studies	14
Figure 2.8	Process of collection of ECG data from two different institutions	16
Figure 2.9	Methodology of this research, the development of model 1 in stage 1 will be the useful part to refer to for this research	16
Figure 2.10	CNN architecture	17
Figure 2.11	LSTM architecture	17
Figure 2.12	CNN-LSTM architecture	18

Figure 2.13 Performance of each classifier in model 1 in classification stage 1 18
Figure 2.14 Data Collection and Machine Learning System of this research 20
Figure 2.15 Advantages and disadvantages of training classifier with only single-lead data 21
Figure 2.16 Data Collection 22
Figure 2.17 Data Collection 22
Figure 2.18 C and D are the result of pre-processing steps 23
Figure 2.19 CNN architecture 23
Figure 2.20 Result of comparative test 24
Figure 2.21 Result of comparative test 25
Figure 2.22 Pre-processing steps taken in [26] 26
Figure 2.23 Network architecture of 2D-CNN classifier 26
Figure 2.24 Performance of 2D-CNN classifier 27
Figure 2.25 Result of comparative test between other transfer learning models and the proposed model 27
Figure 2.26 CNN architecture 28
Figure 2.27 Convolutional block architecture 28
Figure 2.28 CNN-LSTM architecture 29
Figure 2.29 Meta-classifier architecture 29
Figure 2.30 Performance of each classifier 29
Figure 2.31 Comparison of performance between the proposed model and other models 30
Figure 2.32 Performance of the LSTM mod 31
Figure 2.33 Comparison of performance between proposed LSTM and other proposed models 31
Figure 2.34 Formula of balanced accuracy and Matthew's Correlation Coefficient 32
Figure 2.35 Performance of each model in classification 32
Figure 2.36 Distribution of classes in each lead 33
Figure 2.37 CNN architecture 33
Figure 2.38 CNN architecture 34

Figure 2.39 Accuracy achieved by CNN in each lead 34
Figure 2.40 Performance metrics achieved by lead V4 35
Figure 2.41 CNN architecture 35
Figure 2.42 Performance metrics of each dataset 36
Figure 3.1 System design of this project 39
Figure 3.2 Network architecture of LSTM classifier 41
Figure 3.3 Parameters of uni-directional LSTM classifier 42
Figure 3.4 Parameters of bi-directional LSTM classifier 42
Figure 3.5 Network architecture of GRU classifier 43
Figure 3.6 Parameters of uni-directional GRU classifier 43
Figure 3.7 Parameters of bi-directional GRU classifier 43
Figure 3.8 Network architecture of 1D-CNN classifier 44
Figure 3.9 Parameters of 1D-CNN classifier 44
Figure 3.10 Network architecture of Transformer classifier 45
Figure 3.11 Parameters of Transformer classifier 45
Figure 3.12 Example of confusion matrix where TP is a true positive, TN is a true negative, FP is a false positive, and FN is a false negative 46
Figure 4.1 Number of subjects in each diagnostic class 49
Figure 4.2 Technical description of the signal recorder 49
Figure 4.3 Organization of patient data in PTB database 50
Figure 4.4 Contents of patient folder 50
Figure 4.5 Example of k-fold cross-validation 51
Figure 4.6 Time-shift Data Augmentation 51
Figure 4.7 Intra-Patient Splitting 52
Figure 4.8 Inter-Patient Splitting Part 1 53
Figure 4.9 Inter-Patient Splitting Part 2 53
Figure 4.10 Inter-Patient Splitting Part 3 54
Figure 4.11 Implementation of SG filter 54
Figure 4.12 Result of successful filtering, the red line represents the filtered signal, while the green line represents the baseline 55
Figure 4.13 Implementation of R-peak detection and segmentation 56
Figure 4.14 R-peaks were detected 56

Figure 4.15 Segmentation from 300 points before R-peak and 500 points after R-peak 56
Figure 4.16 Number of MI ECGs and normal ECGs in each set of each fold (Intra-Patient) 56
Figure 4.17 Number of MI ECGs and normal ECGs in each set (Inter-Patient) 57
Figure 4.18 Implementation of transformation 58
Figure 4.19 The training and testing set is encapsulated in the data loader 58
Figure 4.20 Implementation of LSTM networks 59
Figure 4.21 Implementation of GRU networks 59
Figure 4.22 Implementation of 1D-CNN network 60
Figure 4.23 Implementation of Transformer classifier 61
Figure 4.24 Training function 62
Figure 4.25 Early Stop Mechanism (Validation Loss) 63
Figure 4.26 Early Stop Mechanism (Validation Accuracy) 63
Figure 4.27 An example of early stopping 64
Figure 4.28 Testing function 65
Figure 4.29 Main code part 1 66
Figure 4.30 Main code part 2 67
Figure 4.31 Main code part 3 68
Figure 4.32 Main code part 4 68
Figure 4.33 Graph plot for loss, accuracy, precision, recall and F1-score of best-performing model 70
Figure 4.34 Confusion matrix of best-performing model’s training and validation performance metrics 71
Figure 4.35 Confusion matrix of best-performing model’s testing performance metrics 71
Figure 4.36 Graphs of performance metrics of intra-patient uni-directional LSTM classifier 74
Figure 4.37 Training confusion matrix of intra-patient uni-directional LSTM classifier 74
Figure 4.38 Validation confusion matrix of intra-patient uni-directional 75

LSTM classifier
Figure 4.39 Graphs of performance metrics of inter-patient uni-directional LSTM classifier 75
Figure 4.40 Training confusion matrix of inter-patient uni-directional LSTM classifier 76
Figure 4.41 Validation confusion matrix of inter-patient uni-directional LSTM classifier 76
Figure 4.42 Graphs of performance metrics of intra-patient bi-directional LSTM classifier 77
Figure 4.43 Training confusion matrix of intra-patient bi-directional LSTM classifier 77
Figure 4.44 Validation confusion matrix of intra-patient bi-directional LSTM classifier 78
Figure 4.45 Graphs of performance metrics of inter-patient bi-directional LSTM classifier 78
Figure 4.46 Training confusion matrix of inter-patient bi-directional LSTM classifier 79
Figure 4.47 Validation confusion matrix of inter-patient bi-directional LSTM classifier 79
Figure 4.48 Graphs of performance metrics of intra-patient uni-directional GRU classifier 80
Figure 4.49 Training confusion matrix of intra-patient uni-directional GRU classifier 80
Figure 4.50 Validation confusion matrix of intra-patient uni-directional GRU classifier 81
Figure 4.51 Graphs of performance metrics of inter-patient uni-directional GRU classifier 81
Figure 4.52 Training confusion matrix of inter-patient uni-directional GRU classifier 82
Figure 4.53 Validation confusion matrix of inter-patient uni-directional GRU classifier 82
Figure 4.54 Graphs of performance metrics of intra-patient bi-directional GRU classifier 83

Figure 4.55 Validation confusion matrix of intra-patient bi-directional GRU classifier 83
Figure 4.56 Validation confusion matrix of intra-patient bi-directional GRU classifier 84
Figure 4.57 Graphs of performance metrics of inter-patient bi-directional GRU classifier 84
Figure 4.58 Validation confusion matrix of inter-patient bi-directional GRU classifier 85
Figure 4.59 Validation confusion matrix of inter-patient bi-directional GRU classifier 85
Figure 4.60 Graphs of performance metrics of intra-patient 1D-CNN classifier 86
Figure 4.61 Training confusion matrix of intra-patient 1D-CNN classifier 86
Figure 4.62 Validation confusion matrix of intra-patient 1D-CNN classifier 87
Figure 4.63 Graphs of performance metrics of inter-patient 1D-CNN classifier 87
Figure 4.64 Training confusion matrix of inter-patient 1D-CNN classifier 88
Figure 4.65 Validation confusion matrix of inter-patient 1D-CNN classifier 88
Figure 4.66 Graphs of performance metrics of intra-patient Transformer classifier 89
Figure 4.67 Training confusion matrix of intra-patient Transformer classifier 89
Figure 4.68 Validation confusion matrix of intra-patient Transformer classifier 90
Figure 4.69 Graphs of performance metrics of inter-patient Transformer classifier 90
Figure 4.70 Training confusion matrix of inter-patient Transformer classifier 91
Figure 4.71 Validation confusion matrix of inter-patient Transformer 91

classifier
Figure 5.1 Testing confusion matrix of intra-patient uni-directional LSTM classifier 94
Figure 5.2 Testing confusion matrix of inter-patient uni-directional LSTM classifier 94
Figure 5.3 Testing confusion matrix of intra-patient bi-directional LSTM classifier 95
Figure 5.4 Testing confusion matrix of inter-patient bi-directional LSTM classifier 95
Figure 5.5 Testing confusion matrix of intra-patient uni-directional GRU classifier 96
Figure 5.6 Testing confusion matrix of inter-patient uni-directional GRU classifier 96
Figure 5.7 Testing confusion matrix of intra-patient bi-directional GRU classifier 97
Figure 5.8 Testing confusion matrix of inter-patient bi-directional GRU classifier 97
Figure 5.9 Testing confusion matrix of intra-patient 1D-CNN classifier 98
Figure 5.10 Testing confusion matrix of inter-patient 1D-CNN classifier 98
Figure 5.11 Testing confusion matrix of intra-patient Transformer classifier 99
Figure 5.12 Testing confusion matrix of inter-patient Transformer classifier 99

LIST OF TABLES

Table Number	**Title**	**Page**
Table 2.1	Summary of previous works	36
Table 4.1	Specification of laptop	48
Table 4.2	Training and Validation performance metrics of trained classifier in both types of analysis	72
Table 5.1	Testing performance metrics of trained classifier in both types of analysis	

LIST OF ABBREVIATIONS

CVD	Cardiovascular Disease
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
MI	Myocardial Infarction
STEMI	ST-elevation Myocardial Infarction
NSTEMI	Non-ST-elevation Myocardial Infarction
LV	Left Ventricle
CNN	Convolutional Neural Network
1D-CNN	1-Dimensional Convolutional Neural Network
2D-CNN	2-Dimensional Convolutional Neural Network
ECG	Electrocardiogram
LLM	Large Language Model
FC	Fully Connected
IDE	Integrated Development Environment

Chapter 1

Introduction

In this chapter, the background and motivation of this research are presented, along with the contributions to the field, and the outline of the thesis.

Nowadays, risk factors such as the excessive use of tobacco and alcohol, high salt content in food, less fruit and vegetable consumption as well as being physically not active have resulted in hypertension, raised blood glucose (diabetes) as well as obesity or high cholesterol [1]. Figure 1.1 [2] shows that 1.7 million Malaysians lived with major risk factors and Figure 1.2 [3] points out prevalence of diabetes and obesity among Malaysians had shown a steady increment from 2011 to 2019. They act as the contributing factor to cardiovascular disease (CVD) and by the estimation of [3], it is taking 17.9 million lives away from us every year. As a matter of fact, the main cause of death in Malaysia in 2022 is actually ischaemic heart disease and surprisingly it is four times the deaths caused by COVID-19 as shown in Figure 1.3 [4], which is classified as one of the CVDs and its prevalence took the most significant position among the life-threatening incidents. CVDs is a general medical term comprising various symptomatic and asymptomatic [5] disorders of the heart and blood vessels. According to WHO [6], it includes but is not limited to coronary heart disease, arrhythmia, peripheral arterial disease, rheumatic heart disease and congenital heart disease as well as deep vein thrombosis and so on. Obviously, we can tell from the name that CVDs are life-threatening enough to cause damage to every part of our body from the brain to the heart and the leg.

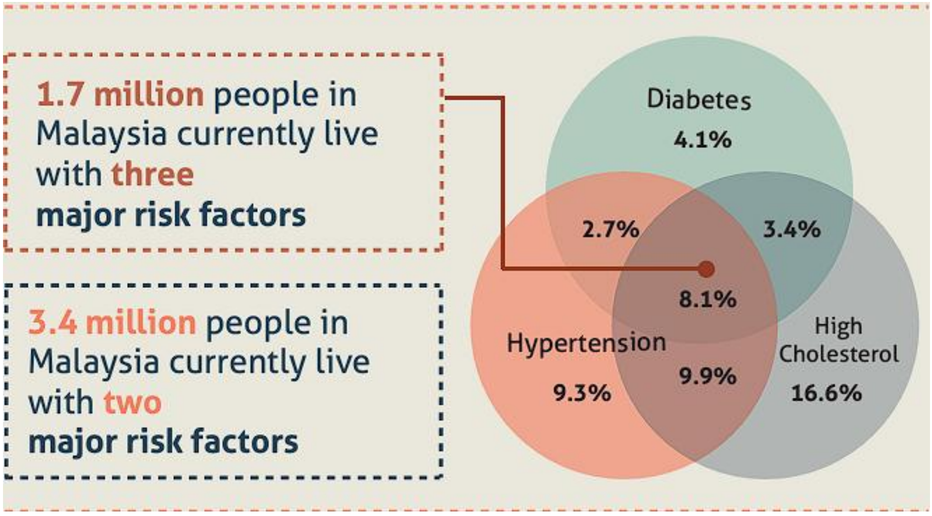
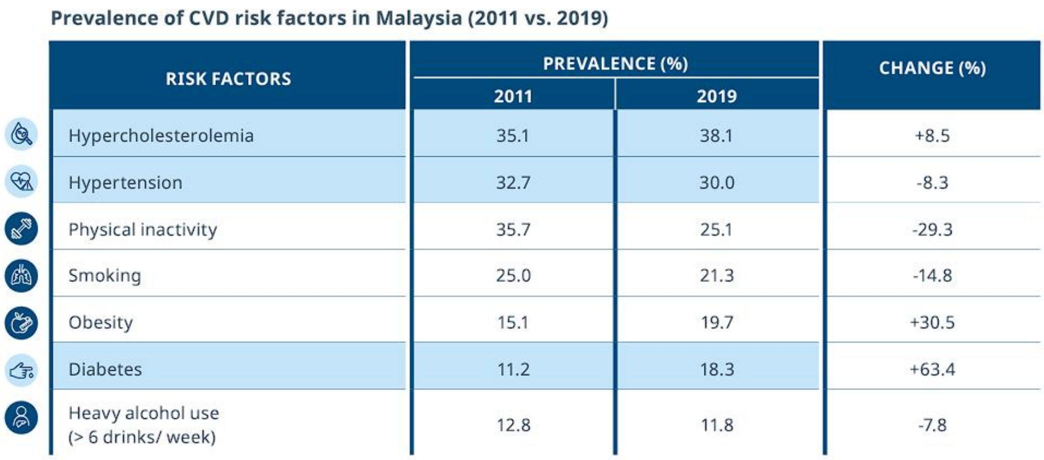


Figure 1.1 Statistics on diabetes, hypertension and high cholesterol in Malaysia [1]



Prevalence of CVD risk factors in Malaysia (2011 vs. 2019)

RISK FACTORS	PREVALENCE (%)		CHANGE (%)
	2011	2019	
Hypercholesterolemia	35.1	38.1	+8.5
Hypertension	32.7	30.0	-8.3
Physical inactivity	35.7	25.1	-29.3
Smoking	25.0	21.3	-14.8
Obesity	15.1	19.7	+30.5
Diabetes	11.2	18.3	+63.4
Heavy alcohol use (> 6 drinks/ week)	12.8	11.8	-7.8

Figure 1.2 Prevalence of CVD risk factor in Malaysia [2]

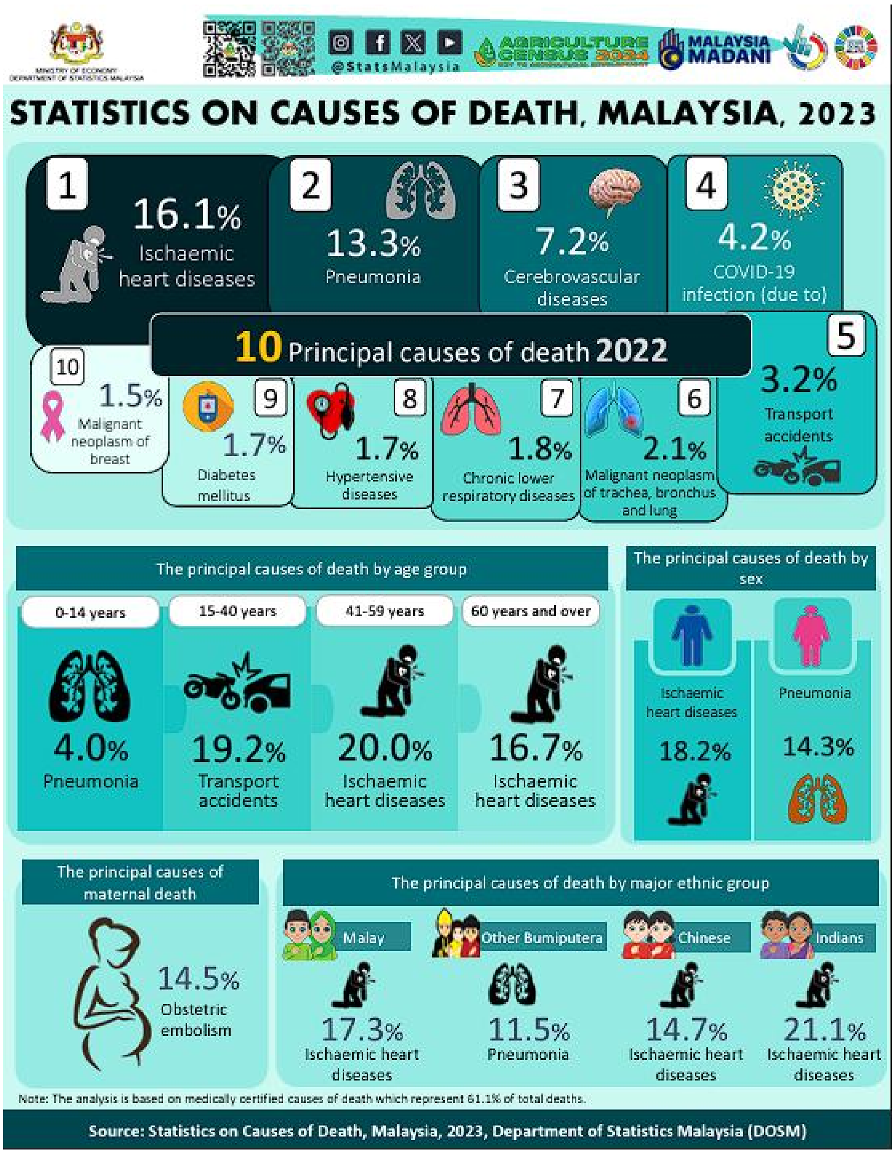


Figure 1.3 Statistics on causes of death in Malaysia [4]

This study is mainly focused on one of the subfields of CVDs, myocardial infarction (MI) which results from coronary heart disease. Coronary heart disease can be explained as obstruction or hindrances of blood flow through the coronary arteries and therefore possibly trigger a myocardial infarction [7]. In layman's terms, myocardial infarction is often described as a heart attack [8]. A person who is having MI probably experiencing chest pain or discomfort (angina) in the chest, arm or shoulder. The coronary artery is responsible for supplying oxygenated blood to our heart muscles (myocardium). Therefore, the blockage of the coronary artery will interfere with the normal blood flow to the myocardium, consequently, they will start to die of ischemia (deficiency of nutrients and oxygen [8]). The death of myocardium brings rapid and irreversible damage to the structure and function of the left ventricle (LV) [10] to lose its original contractility so the cardiac output to the entire body [11] cannot be maintained. This is the reason for the high mortality rate due to MI. Currently, MI can be classified into two classes according to its infarct extent [10], transmural and non-transmural. Transmural infarcts affect the whole myocardium wall (epicardium to endocardium) whereas non-transmural ones only affect 1/3 of the myocardium wall. Transmural MI is also named ST-elevation myocardial infarction (STEMI) [10], it is a subclass of myocardial infarction and also one of the most severe manifestations of acute coronary disease. Its name originated from the distinct morphological change of ECG signal where a heartbeat cycle is composed of PQRST phases, ST segment would show subtle or significant elevation when compared with the healthy patient's normal ECG as the blockage of blood flow will change of electrical activity of heart [12] as shown in Figure 1.4. According to Clinical Practice Guideline Malaysia in 2019 [13], the in-hospital, 30-day and 1-year mortality following STEMI was 10.6% (8.0% in non-STEMI patients), 12.3% and 17.6% while the patients were generally more ill.

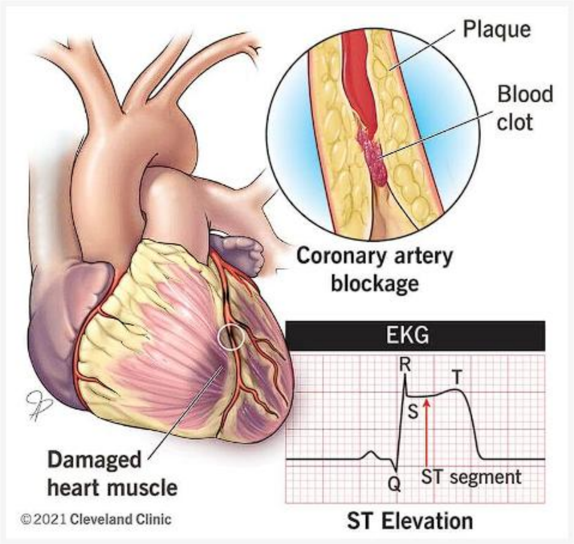


Figure 1.4 Visualization of coronary artery blockage which induces the morphological change of ECG signal [12]

Medical experts apply many tools in diagnosing MI and classifying infarct extents such as serial ECGs, troponin/CK-MB/LDH test (cardiac biomarkers), and immediate or delayed coronary angiography (imaging) [10,14]. In [15], the authors defined onset-to-balloon as the time from occurrence of symptoms to operation of percutaneous coronary intervention (PCI). The second one is door-to-balloon time as the time from arrival at the hospital to the operation of PCI, and presentation time represents the time from symptom onset to arrival at the hospital. A clearer visualization is shown in Figure 1.5.

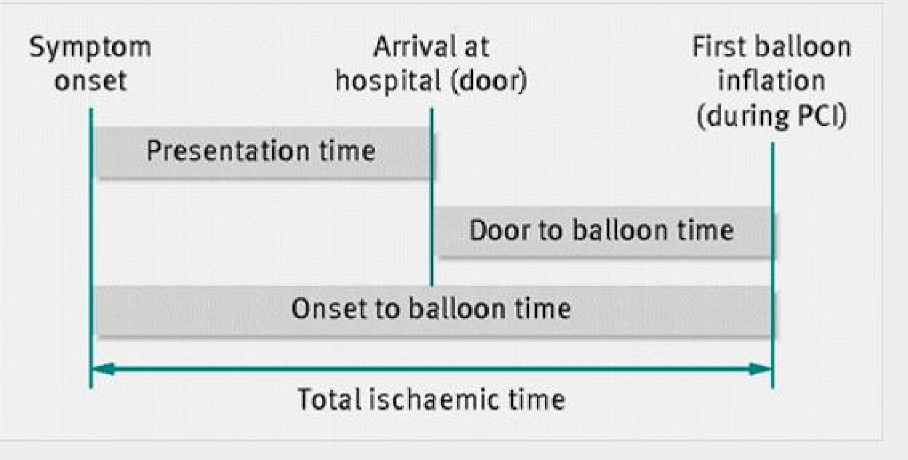


Figure 1.5 The names of different phases during the total ischaemic time [15]

As mentioned before, the detection and diagnosis of MI can be done by using diverse methodologies. Among all the methods that can help in the diagnosis of MI, ECG is the non-invasive and fastest one yet it is cost-effective [16] compared to CAG. An electrocardiogram (ECG) is the most common device used in the medical field to capture and record the electrical activity of the heart [12, 17, 18] for the evaluation and speculation of the occurrence of CVD by interpreting the morphologic change of the signals. It helps show the cardiac cycle and rhythm of humans and enables the doctor to speculate the occurrence of abnormalities in the heart's atrium and ventricles as many arrhythmias and ECG morphological changes associated with angina and atherosclerosis. A conventional ECG is usually named a 12-lead ECG as it consists of six limb leads called I, II, III, aVL, aVR, and aVF as well as six chest leads called V1 to V6 [18, 26]. Besides, ECG plays an extremely important role in telemedicine in ST-Elevation Myocardial Infarction (STEMI) intervention as shown in Figure 1.6. Nevertheless, in the traditional telemedicine network such as LATIN and RAHAT, two population-based telemedicine programs for STEMI in South America, the interpretation of ECG still depends on human resources [19] as it could be observed that besides tertiary hospitals that have expert cardiologists, other medical institutions do not perform as well as the former. In addition to that, it is also crucial to be aware that not all cardiologists in worldwide countries can always perform with 90% accuracy.

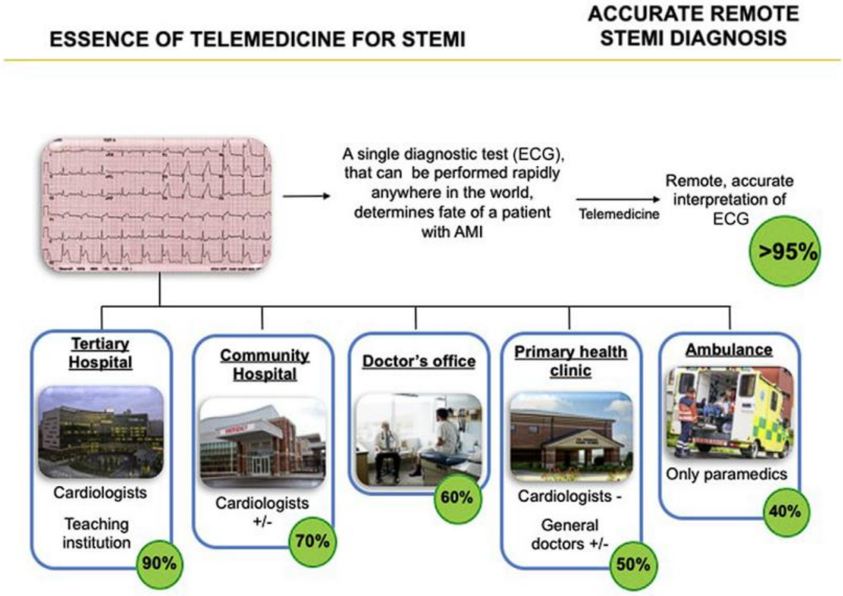

ESSENCE OF TELEMEDICINE FOR STEMI
ACCURATE REMOTE STEMI DIAGNOSIS
A single diagnostic test (ECG), that can be performed rapidly anywhere in the world, determines fate of a patient with AMI
Telemedicine
Remote, accurate interpretation of ECG
>95%
Tertiary Hospital
Cardiologists
Teaching institution
90%
Community Hospital
Cardiologists +/-
70%
Doctor's office
60%
Primary health clinic
Cardiologists -
General doctors +/-
50%
Ambulance
Only paramedics
40%

Figure 1.6 The performance of humans in different medical resource levels [19]

1.1 Problem Statement and Motivation

According to several systematic and comprehensive reviews [20, 21, 22], nowadays many researchers have been utilizing AI techniques mainly deep-learning-based models for automatic and accurate MI detection. The most popular type of deep learning network architecture seems to be CNN (1D-CNN and 2D-CNN) while models like Transformer or sequential models such as LSTM and GRU were less likely to be developed and deployed.

Following using these models a high performance from the perspective of accuracy, specificity, sensitivity, F1, etc. was achieved by conducting their research without specifying the type of data-split method or only applying intra-patient analysis, this can be observed through the tables in [20] collecting the performance figures. An intra-patient analysis is about splitting and randomly allocating the heartbeat data produced from one patient into training sets, validation sets and testing sets [23] which do not truly represent real-world situations where we might encounter an unseen patient. Consequently, the patient-level information is leaked to the model makes the model memorize the ownership of the heartbeat and build a relationship between the ownership and the category of the patient (0/1). As we cannot guarantee that the patient has been seen before, thus instead, the model should learn the temporal relationship between the data points (voltage reading of ECG) in an effort to recognize and differentiate the pattern of MI ECG from the normal ones.

Other than that, some of the researchers would use 12-lead ECG data as the input of their models, it was considered time-consuming and extremely increased the complexity as well as the computational cost of the model. In real-world applications, utilizing 12-lead data will affect the speed of prediction due to the longer processing time needed by loading and pre-processing the data in order to transform it into the format that the model can recognize.

The aforementioned has motivated this research to explore and further validate the performance and feasibility of LSTM, GRU and Transformer for the inter-patient analysis for MI detection in ECG. LSTM, GRU and Transformer are chosen as they are also suitable for processing sequential data just like the readings (amplitude) of ECG while the Transformer has advantages such as a multi-head self-attention mechanism to allow the parallelism and captures an even longer range of dependencies than LSTM and GRU as we can see it is the foundation of LLM nowadays.

On top of that, 2D-CNN requires the ECG images as its input, which is troublesome for portable devices to keep capturing and sending images to their cloud or on-site server at a high frequency as this will lead to flooding of internet bandwidth and a more severe device heating. Instead, sequential models such as 1D-CNN, LSTM, GRU and Transformer only require the string/text data containing ECG readings recorded by the pre-installed sensor in the device, the total size of transferred data is significantly reduced compared to graphics.

1.2 Research Objectives

1. To propose an automatic detection of myocardial infarction (MI) from ECG signals for inter-patient analysis, utilizing data from a single ECG lead, deploying models including LSTM, GRU, 1D-CNN and Transformer.

2. To evaluate the accuracy, precision, sensitivity and F1-score of MI detection within LSTM, GRU, 1D-CNN and Transformer networks for inter-patient analysis.

3. To investigate the effect of deploying different data-split methods on the performance of the models for inter-patient and intra-patient analysis.

1.3 Project Scope and Direction

1. The deliverable of this project is four deep learning models (LSTM, GRU, 1D-CNN and Transformer with each trained from two different data-split methods) that can automatically diagnose MI by feeding the pre-processed single-lead data to the models.
2. ECG signal data will be obtained from the open-source database, PTB Diagnostic ECG database [24] as it is one of the few that can be obtained for free. ECG signal data is collected from 200 patients consisting of 148 MI patients and 52 normal subjects.
3. Lead II data will be chosen for this research as this is the most commonly used in other research and the performance is satisfactory.

1.4 Contributions

The inter-patient splitting method contributed to the improved reliability of the model to be deployed in the real world for unseen patients. On the other hand, the models trained by the intra-patient method are suitable for detecting the presence of a heart attack in a seen patient. Secondly, using single-lead ECG data contributed to the lower computational cost and complexity of the models. By comparing the intra-/inter-patient models' performance, the effect of different data-split methods on performance can be observed. Ultimately improving patient outcomes, which in return increases the chance of successful treatment and better overall recovery of the patient.

1.5 Report Organization

This thesis was written to show the details of the research in the following chapters. In Chapter 2, a systematic and comprehensive review of some of the current literature was conducted and represented by a summary table. Chapter 3, is a clear manifestation of the system model and design. In Chapter 4, the implementation of the system model was shown to build deep learning models using the Pytorch framework. In Chapter 5, a comparison between the performance of inter-patient models and intra-patient models was conducted with a discussion of the findings. Eventually, a conclusion was drawn from the experimental result for the whole development and progress of the project in Chapter 6. Besides, the limitations and future works were also discussed in this chapter.

Chapter 2

Literature Review

2.1 Previous works on developing Deep-Learning-Based MI classifier

2.1.1 Near real-time single-beat myocardial infarction detection from single-lead electrocardiogram using Long Short-Term Memory Neural Network [23]

In this research, an LSTM model was proposed and the problem of data leakage, overfitting and bias resulting from the intra-patient splitting method was highlighted. The authors conducted experiments to examine the effects of different data-split techniques on the performance of the model in the testing phase as they claimed that the intra-patient splitting technique could produce a misleadingly outstanding performance. In the end, the model could achieve an accuracy of 89.56%, a recall/sensitivity of 91.88%, and a specificity of 80.81%.

The authors noticed that abundant papers regarding to this field used different sources of data, different classification methods and different data-split methods. The data-split methods were classified by the authors into three groups: beat-split, record-split and patient-split. The beat-split technique is the least restrictive method which allows an individual patient could have a set of heartbeats in the training and another set of heartbeats, albeit different, in the testing set. The record-split technique represents the individual heartbeats associated with each of the individual records that can only be present in one of the sets. In simpler words, any given record cannot have a simultaneous appearance in the testing and training sets. The last and also the most restrictive method, the patient-split method allows each patient with all of its available records to be present in one of the sets. Although some papers achieved good results, they did not mention clearly their types of classification whether it was trained using an intra-patient or an inter-patient analysis. At the same time, the authors also observed that the CNN classifier from another paper which claimed a title of real-time classification requires whole ECG images to be the input of the classifier.

Single-lead (lead II) ECG data was used in this research even though the authors knew that it was challenging to use only single-lead data to detect MI occurring in different regions of the heart. The reason for using lead II data was because of the common deployment of this lead in sports equipment such as trackers. It is crucial to highlight that in this research, while deploying the patient split method, no heartbeats or subjects (MIs and HCs) seen in the training phase appeared in the testing phase again, therefore the classifier was an inter-patient classifier.

The pre-processing steps that have been utilized in this research included A 60 Hz stop-band and a 500-ms moving average filter for purposes of denoise and baseline wandering removal, Independent Component Analysis (ICA) based algorithm to identify the R-peak locations and thereafter segmented each heartbeat with a consistent sequence length of 1000ms.

A three-layer LSTM was built with layers 1 and 2 having 100 neuron units (hidden size = 100) by using the tanh activation function while layer 3 only consists of 1 neuron by using the sigmoid activation function. A clearer visualization (**Figure 2.1**) helps in understanding the structure. An RMSprop with settings of learning rate = 0.1, weight decay = 0.95, and epsilon = 10e− 6, and gradient clipping (5.0) optimizer was chosen in the training phase. The weights of this model did not start from zero, instead, Xavier initialization was chosen. Moreover, early stopping was set to 10 for the training set so that the process would stop after 10 epochs with no performance improvement.

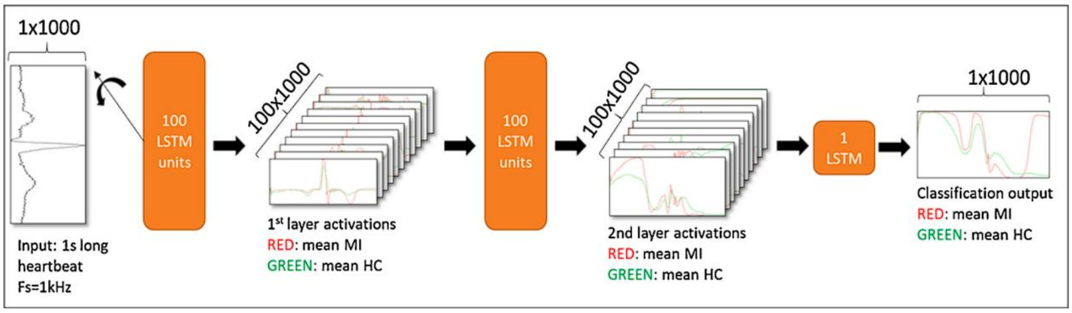
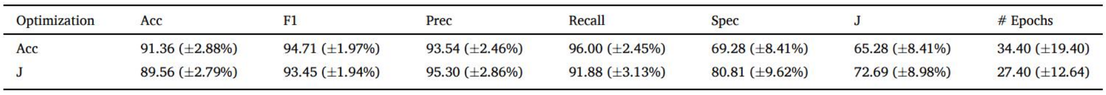


Figure 2.1 Architecture of LSTM model [23]

The performance metrics of this research included Accuracy, F1-score, Precision, Recall, Specificity and J-measure (=Recall + Specificity − 1). The experiment also conducted optimization according to two different metrics: Accuracy and J-measure. **Figure 2.2** shows the respective results according to the categories. The mean training time for the model took 34.4 epochs and 27.4 epochs for respective metrics. The mean testing time of 1-s sample ECG only took around 40ms therefore the authors claimed this was enough for real-time application as the time between fast-pace heartbeats was around 300ms. The authors explained that the specificity could easily obtain bad results as the dataset is heavily imbalanced: the number of MI heartbeats is triple that of the normal subject.



Optimization	Acc	F1	Prec	Recall	Spec	J	# Epochs
Acc	91.36 (±2.88%)	94.71 (±1.97%)	93.54 (±2.46%)	96.00 (±2.45%)	69.28 (±8.41%)	65.28 (±8.41%)	34.40 (±19.40)
J	89.56 (±2.79%)	93.45 (±1.94%)	95.30 (±2.86%)	91.88 (±3.13%)	80.81 (±9.62%)	72.69 (±8.98%)	27.40 (±12.64)

Figure 2.2 Performance of LSTM model optimized using different metrics: Accuracy and J-measure [23]

After developing and training the model well using an inter-patient method, the authors decided to retrain the model using different data split methods to examine how they affect the performance metrics. Firstly, comparing the evolution of performance metrics of the inter-patient analysis having the trend in **Figure 2.3**. Nevertheless, a different trend was observed in the intra-patient analysis as shown in **Figure 2.4**, the testing performance almost followed/tracked the training performance, and this was very abnormal. This is due to when meeting a new unseen patient, the model should be learning the new variability or any other useful information related to the disease itself that improves the performance, instead, the intra-patient classifier tended to learn the features across the heartbeats that belong to certain patients. Ultimately, the classifier appeared to memorize how to differentiate the patient according to the heartbeat and decide the classification based on its prior knowledge. This conclusion can be further validated by an experiment: the authors randomized the correct labels (0/1) and distributed them uniformly before training the intra-patient classifier. Therefore, if the aforementioned was wrong, the classifier should not achieve an accuracy not greater than 50%. As a matter of fact, **Figure 2.5** already proved that inappropriate data-split technique could lead to mask overfitting and extremely good performance metrics.

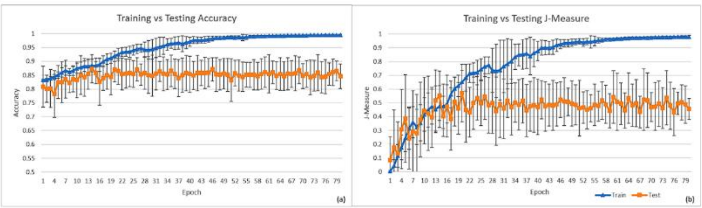


Figure 2.3 Evolution of performance of inter-patient classifier based on optimization of accuracy and J-measure metrics [23]

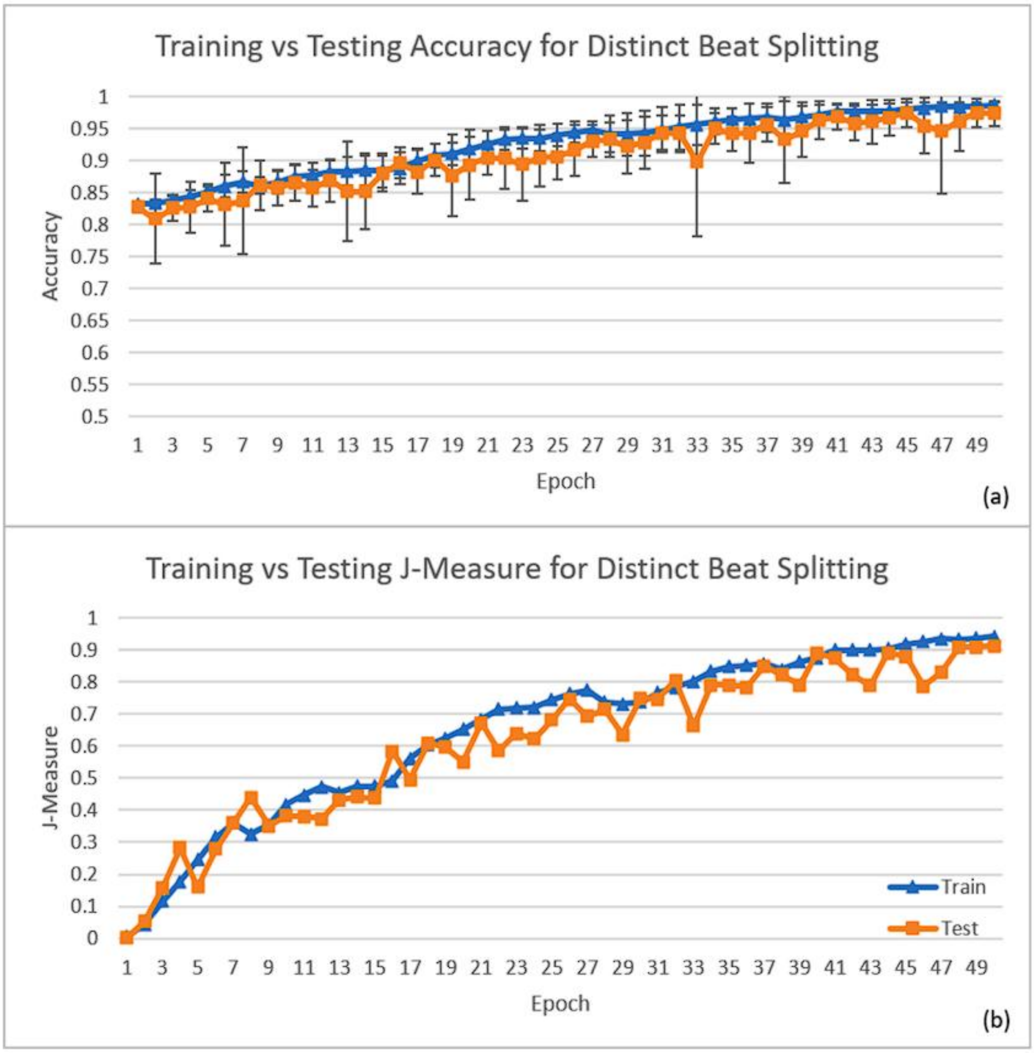


Figure 2.4 Evolution of performance of intra-patient (beat-split) classifier [23]

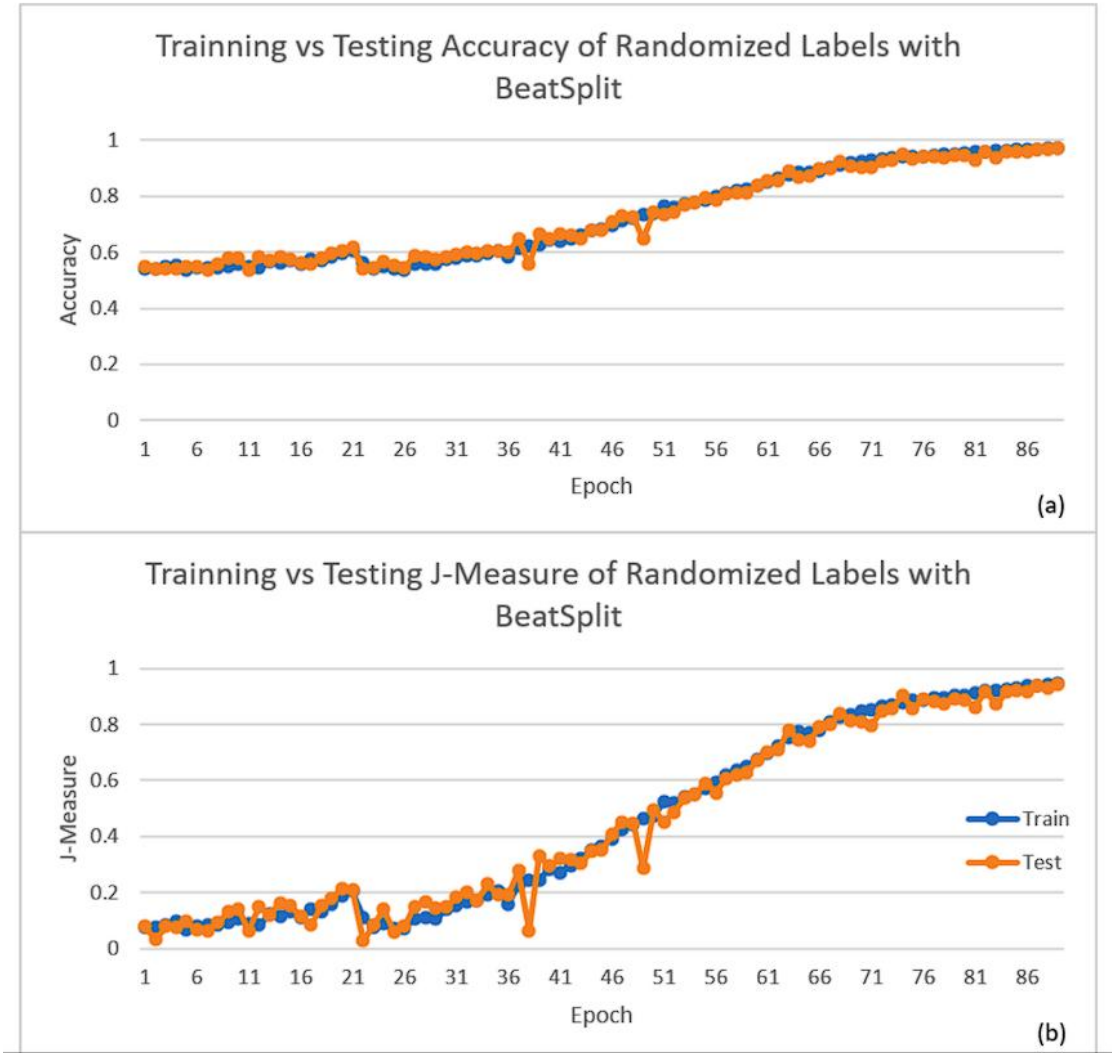


Figure 2.5 Experimental result to test if the intra-patient classifier will memorize the patients [23]

The authors also emphasized that using only lead II data has its weakness in detecting MI when it occurs in some regions of the heart such as lateral MI and posterior MI as shown in **Figure 2.6**. In conclusion, the authors claimed that their research was unique at that time as they were using the combination of single-lead data, single heartbeat plus inter-patient splitting method, so there were no other studies that could fairly compare to their work. **Figure 2.7** shows that different flaws such as a low number of data or only focus on Inferior MI existed in other papers albeit they used the patient-split method too.

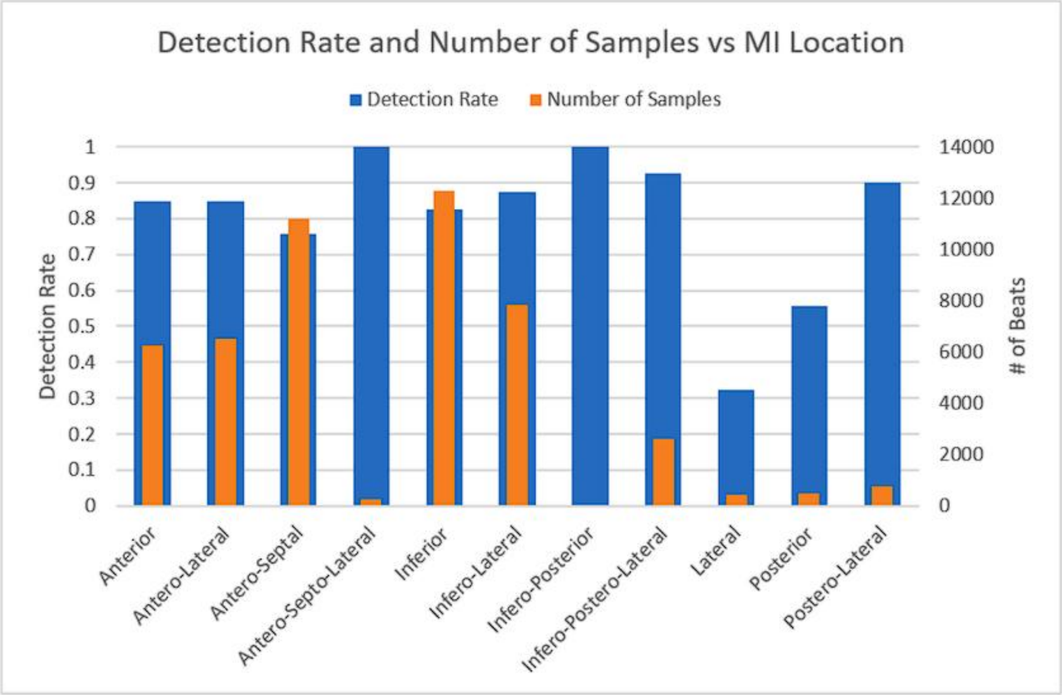
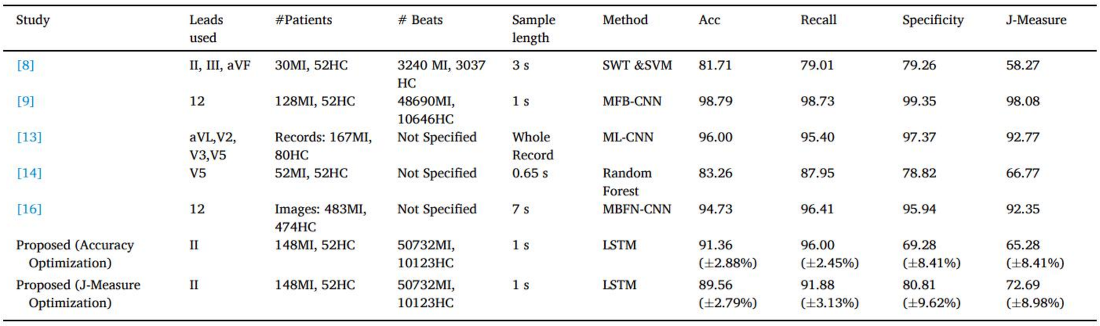


Figure 2.6 Relationship of MI detection rate and MI location [23]



Study	Leads used	#Patients	# Beats	Sample length	Method	Acc	Recall	Specificity	J-Measure
[8]	II, III, aVF	30MI, 52HC	3240 MI, 3037 HC	3 s	SWT &SVM	81.71	79.01	79.26	58.27
[9]	12	128MI, 52HC	48690MI, 10646HC	1 s	MFB-CNN	98.79	98.73	99.35	98.08
[13]	aVL,V2, V3,V5	Records: 167MI, 80HC	Not Specified	Whole Record	ML-CNN	96.00	95.40	97.37	92.77
[14]	V5	52MI, 52HC	Not Specified	0.65 s	Random Forest	83.26	87.95	78.82	66.77
[16]	12	Images: 483MI, 474HC	Not Specified	7 s	MBFN-CNN	94.73	96.41	95.94	92.35
Proposed (Accuracy Optimization)	II	148MI, 52HC	50732MI, 10123HC	1 s	LSTM	91.36 (±2.88%)	96.00 (±2.45%)	69.28 (±8.41%)	65.28 (±8.41%)
Proposed (J-Measure Optimization)	II	148MI, 52HC	50732MI, 10123HC	1 s	LSTM	89.56 (±2.79%)	91.88 (±3.13%)	80.81 (±9.62%)	72.69 (±8.98%)

Figure 2.7 Comparison of performance between the proposed model and other studies [23]

2.1.2 Deep Learning Networks Accurately Detect ST-Segment Elevation Myocardial Infarction and Culprit Vessel [16]

A research paper produced by Wu, Lin et al. was published which focuses on using deep learning networks including CNN-LSTM, LSTM and CNN to detect STEMI and find culprit vessels. In this paper, the authors highlighted that coronary angiography (CAG) is an invasive and radioactive examination method to accurately prove the presence of STEMI instead of other CVDs. This instead indicated the importance of using twelve-lead ECG as it is a non-invasive and cost-effective method.

Nevertheless, the introduction of ECG to the medical field cannot permanently solve the dilemma of differentiating STEMI from some diseases such as pneumothorax, aortic dissection and pulmonary embolism. From the morphological aspect of ECG, its dynamic and spontaneous ST elevation, lead positioning, improper high-pass filter setting and QRS sections' width as well as axis might affect the ST elevation magnitude too. This eventually resulted in low specificity and sensitivity in detecting STEMI through ECG because the ECG is not user-friendly to medical personnel. That is the reason why machine learning-based (ML-based) or deep learning-based (DL-based) methods are introduced for interpretation. The authors realized that previous studies were limited by only utilizing the data from MIT-BIH and PTB databases. On top of this, some other research only employed typical STEMI ECG patterns and excluded other ECG-related ST segment changes such as ventricular premature beat, left ventricular hypertrophy, complete left bundle branch block, and complete right bundle branch block. In conclusion, as an effort to avoid the weaknesses of previous models, authors established their own real-world ECG dataset based on verification of CAG result.

The authors collected their STEMI and control (healthy) data from the Hospital Information system and Cardio-Catheter Room database from January 2015 to December 2018 in the Third-Affiliated Hospital of Sun-Yat-sen University (Cohort 1) as their internal dataset. Moreover, the external dataset was collected from Guangzhou First People's Hospital (Cohort 2). The authors set strict inclusion and exclusion criteria for the data collection. The ECG sample will be collected if the final diagnosis of STEMI is present and without a history of myocardial infarction or PCI. The ECG sample will be rejected if he/she is a patient who needs CAG for any reason and does not reach the diagnosis of STEMI, data with excessive noise, unstable or incomplete baseline, more than one vessel disease other than STEMI, no CAG performed during the first 24 hours of the onset of symptoms of STEMI. The collected ECG was performed at a sampling rate of 1000Hz and data with excessive noises were removed using wavelet transform. The authors also applied a 5-s segment ECG input model. The final specification of ECG data was (5000,12). **Figure 2.8** shows the process of collection of the ECG dataset.

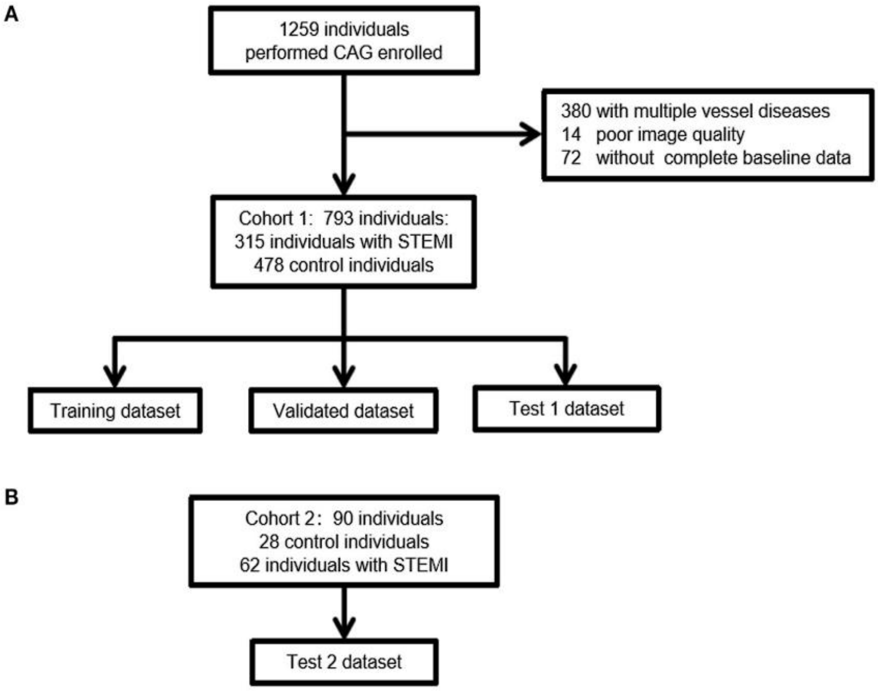

A
1259 individuals performed CAG enrolled
380 with multiple vessel diseases
14 poor image quality
72 without complete baseline data
Cohort 1: 793 individuals:
315 individuals with STEMI
478 control individuals
Training dataset
Validated dataset
Test 1 dataset
B
Cohort 2: 90 individuals
28 control individuals
62 individuals with STEMI
Test 2 dataset

Figure 2.8 Process of collection of ECG data from two different institutions [16]

The authors then started to build three different models to find the one that performed the best. They split the training process into two stages, In the first stage the models were trained for classification between MI and control while in the second stage, the models were trained to find the culprit's vessels. Thus, this literature review will only focus on the development of model 1 in stage 1. The process was visualized by the authors in **Figure 2.9**.

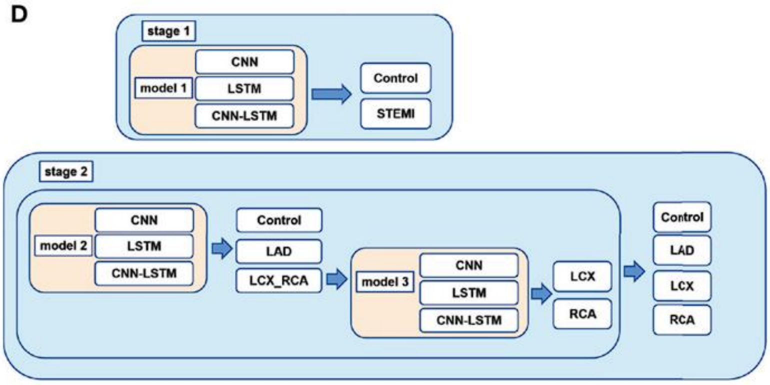
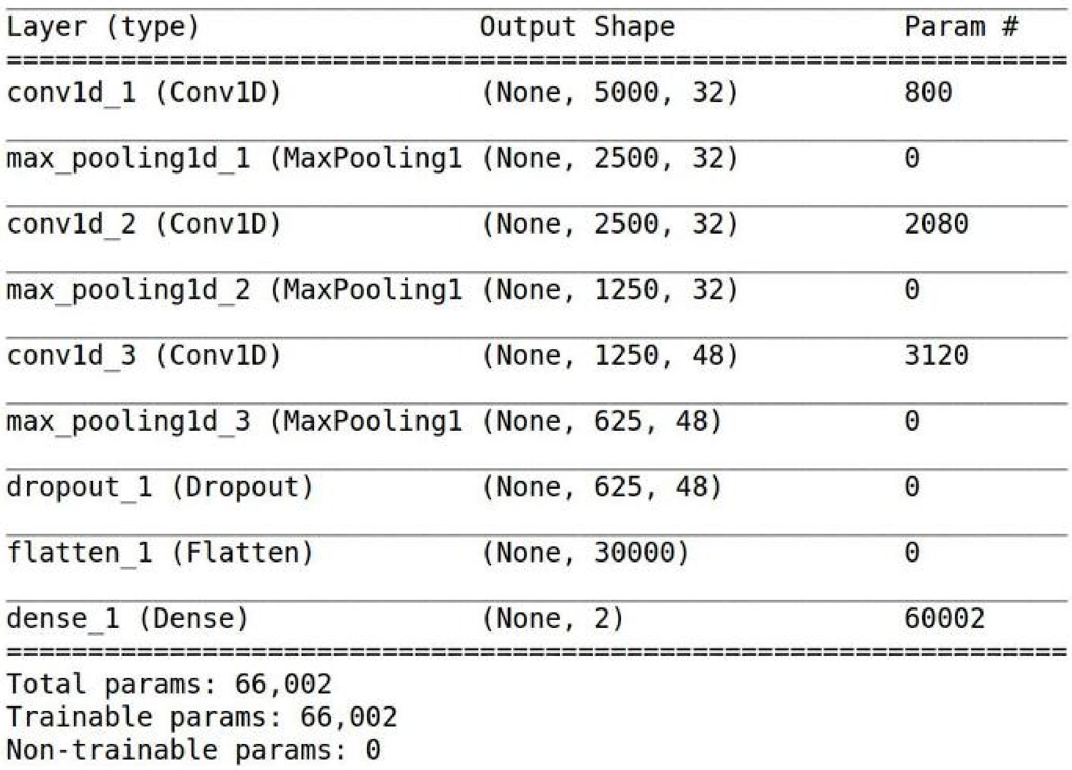


Figure 2.9 Methodology of this research, the development of model 1 in stage 1 will be the useful part to refer to for this research [16]

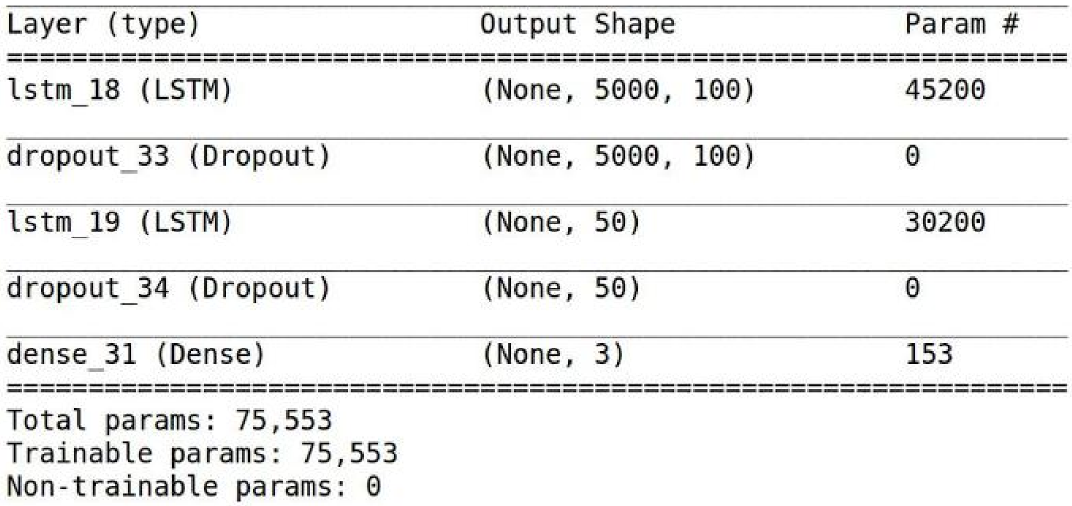
For CNN, the authors designed three layers with the kernel size 2 and random kernel number from a list of parameters 16, 24, 32, 48, and 64. Each layer was followed by a pooling layer, and yet a dropout layer was inserted to improve the generalizability. The details of the architecture of CNN are shown in **Figure 2.10**.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 5000, 32)          800
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 2500, 32)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 2500, 32)          2080
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 1250, 32)          0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 1250, 48)          3120
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 625, 48)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 625, 48)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 30000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 60002
=================================================================
Total params: 66,002
Trainable params: 66,002
Non-trainable params: 0
```

Figure 2.10 CNN architecture [16]

The authors designed two layers for LSTM which has 100 neurons and 50 neurons (best performance) in the first and second layers respectively which the neurons' numbers were randomly picked from a list of parameters of 500, 200, 100 and 50. A dropout layer with a rate of 0.2 was applied after each LSTM layer to increase generalizability. The details of the architecture of LSTM are shown in **Figure 2.11**.



```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_18 (LSTM)               (None, 5000, 100)         45200
_________________________________________________________________
dropout_33 (Dropout)         (None, 5000, 100)         0
_________________________________________________________________
lstm_19 (LSTM)               (None, 50)                30200
_________________________________________________________________
dropout_34 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_31 (Dense)             (None, 3)                 153
=================================================================
Total params: 75,553
Trainable params: 75,553
Non-trainable params: 0
```

Figure 2.11 LSTM architecture [16]

For the CNN-LSTM model that combined CNN and LSTM, the final output was (50, 12, and 48), reshaped into (50, 576), and then fed into subsequent LSTM networks with 50 neurons to calculate 50 samples of 576 dimensions, which became 50 dimensions after calculation (**Figure 2.12)**. After the development, the authors conducted a blind and independent comparative test between the different DL-based methods with 16 doctors which contained four medical interns, four internal medicine residents, four experienced cardiologists, and four emergency physicians.

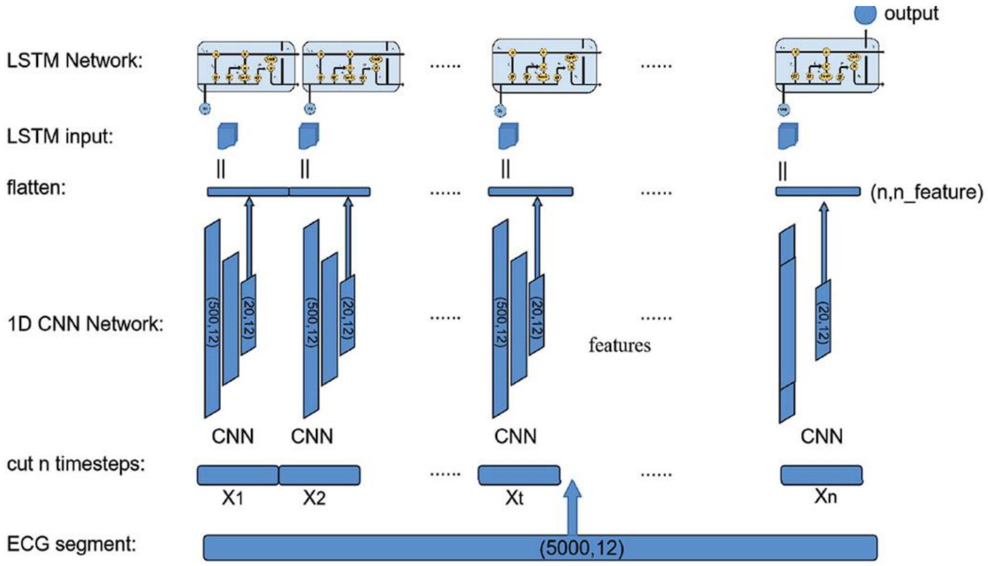
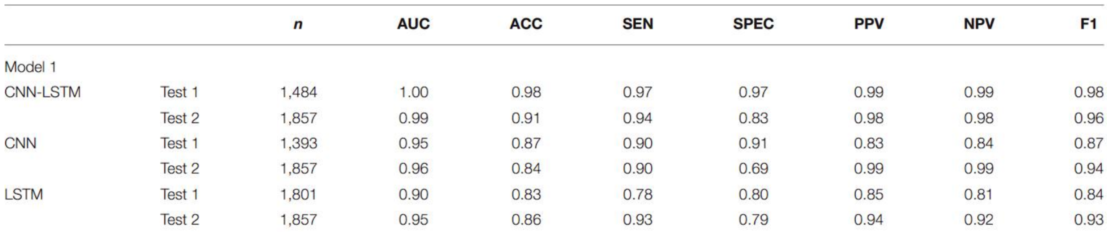


Figure 2.12 CNN-LSTM architecture [16]

Among the models built, the CNN-LSTM performed best with AUCs of 1.00 and 0.99 in test 1 and test 2 in stage 1, this has shown that the CNN-LSTM model has a good generalization ability because there is no significant difference between internal and external dataset while for the stage 2 results would not be discussed here as the objective of stage 2 was to find culprit's vessel. In a comparative test between CNN-LSTM and doctors, it outperformed doctors with an AUC of 1.0 as shown in **Figure 2.13**.



		n	AUC	ACC	SEN	SPEC	PPV	NPV	F1
Model 1									
CNN-LSTM	Test 1	1,484	1.00	0.98	0.97	0.97	0.99	0.99	0.98
	Test 2	1,857	0.99	0.91	0.94	0.83	0.98	0.98	0.96
CNN	Test 1	1,393	0.95	0.87	0.90	0.91	0.83	0.84	0.87
	Test 2	1,857	0.96	0.84	0.90	0.69	0.99	0.99	0.94
LSTM	Test 1	1,801	0.90	0.83	0.78	0.80	0.85	0.81	0.84
	Test 2	1,857	0.95	0.86	0.93	0.79	0.94	0.92	0.93

Figure 2.13 Performance of each classifier in model 1 in classification stage 1 [16]

To further emphasize the advantage of their experiment, the authors highlighted that they used a current real-world database which included arrhythmias that also affected ST segment changes, unlike databases (MIT-BIH and PTB) used by previous studies to achieve better sensitivity and specificity. On top of that, the authors used raw ECG data from 12-lead ECG as their input instead of doing feature extraction in this end-to-end model to lessen feature loss, maintain data integrity, and improve accuracy. Compared to another paper that proposed a similar network to coronary artery disease ECG classification, this research used fewer LSTM layers and could accept larger input. Nevertheless, the authors also pointed out their limitations including they should use 18-lead ECG to increase the robustness.

2.1.3 Evolution of single-lead ECG for STEMI detection using a deep learning approach [25]

According to the advantages that can be brought by single-lead analysis, Gibson et al. proposed their deep-learning model (CNN) for STEMI detection to speed diagnosis During this research, there were a total of 2 models developed for STEMI detection and localization respectively. In the beginning, the authors mentioned the concept of ‘door-to-needle’ (D2N), ‘door-to-balloon’ (D2B) times and ‘symptom-to-door’, the improvement strategies such as management and human resources that have been implemented in hospitals managed to get D2N and D2B times significantly reduced to below 90 minutes and 30 minutes respectively. The problem was, that the ‘symptom-to-door’ time remained at 2.5 hours because of the lack of improvement, this motivated the authors to develop an algorithm for STEMI detection using artificial intelligence, and yet they assumed the accuracy for the analysis using single-lead can be the same as using twelve-lead.

The authors managed to obtain good data from a reliable source, which is the Latin America Telemedicine Infarct Network (LATIN), a population-based Acute Myocardial Infarction (AMI) program. Yet, this brought limitations to their work which the patients were only made up of local people, therefore the generalizability of the classifier might not be satisfactory in other geolocations. The datasets contained 4255 STEMI ECGs and 4256 non-STEMI ECGs, the non-STEMI data was further broken down into 2128 Normal ECGs and 2128 Abnormal ECGs (interpreted by an expert cardiologist from a group of 30 specialists.) which can increase the robustness of the model that it could detect STEMI not only from STEMI data versus normal data, but also versus non-STEMI but unhealthy data. The ECGs were then pre-processed to produce heartbeats with a consistent length of 1.3 s (0.4 to the left and 0.9 to the right according to QRS location) as the input of the CNN model through wavelet technique to detect QRS complex and reduce noise. Next, the authors implemented the following methodology:

1. CNN was developed to identify a STEMI among 12-lead ECGs;
2. CNN was developed to correctly detect a STEMI by analyzing single-lead ECGs;
3. Third, the data was analysed to identify the best single lead for STEMI detection and compared against the 12-lead ECG model results.

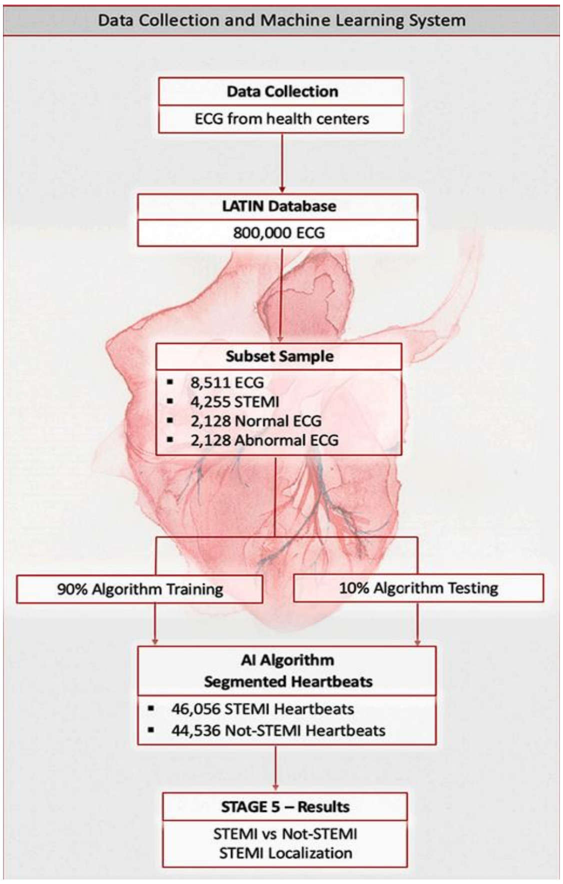
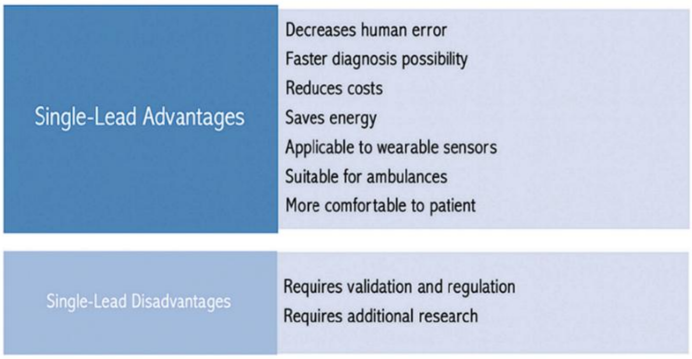


Figure 2.14 Data Collection and Machine Learning System of this research [25]

As a result, the experiment using 12-lead ECG for analysis outperformed the single-lead ECG. Their results are: 96.3%, the specificity was 96.8%, and the accuracy was 96.5% (12-lead) versus accuracy of 90.5%, sensitivity of 86% and specificity of 94.5% (single-lead V2(performed the best)). The authors also listed several advantages and disadvantages of using single-lead data to support their work in **Figure 2.15**.



Single-Lead Advantages	Decreases human error Faster diagnosis possibility Reduces costs Saves energy Applicable to wearable sensors Suitable for ambulances More comfortable to patient
Single-Lead Disadvantages	Requires validation and regulation Requires additional research

Figure 2.15 Advantages and disadvantages of training classifier with only single-lead data [25]

2.1.4 Early detection of ST-segment elevated myocardial infarction by artificial intelligence with 12-lead electrocardiogram [26]

A 1D-CNN (Res-Net) model was proposed to solve the problem of incidence and mortality of STEMI which was still unsatisfactory in China. As the most severe CVD, it caused 31.5% of global all-cause mortality, and ischemic heart disease was responsible for the contribution of 90.9% of CVD-related deaths in China. The door-to-balloon was below 90 minutes in China due to the establishment of Chest Pain Centers, whereas the total ischemic time was as high as 4 hours and this is the main reason for the high mortality of STEMI in China, thus a real-time monitoring and detection system was required. All of these supported the authors to propose an automatic detection system of STEMI to incorporate with ECG equipment to achieve the goal. The authors realized that the detection of STEMI is much more complicated than other symptoms such as arrhythmia because the data requirement is higher such as 12-lead instead of single-lead, and the second one is the requirement for the sensitivity of STEMI detection is higher as STEMI is more fatal than arrhythmia. The last of the difficulties was the filtering as one of the preprocessing steps in major studies could compromise the ST magnitude. This motivated using raw 12-lead data without any filtering process, but the authors still applied a bandpass filter in their research.

During the data collection and labelling, the authors collected raw STEMI and control ECG from Shanghai Tenth People's Hospital and Changhai Hospital, in total of 667 STEMI samples and 7571 control samples (**Figure 2.16** and **Figure 2.17**).

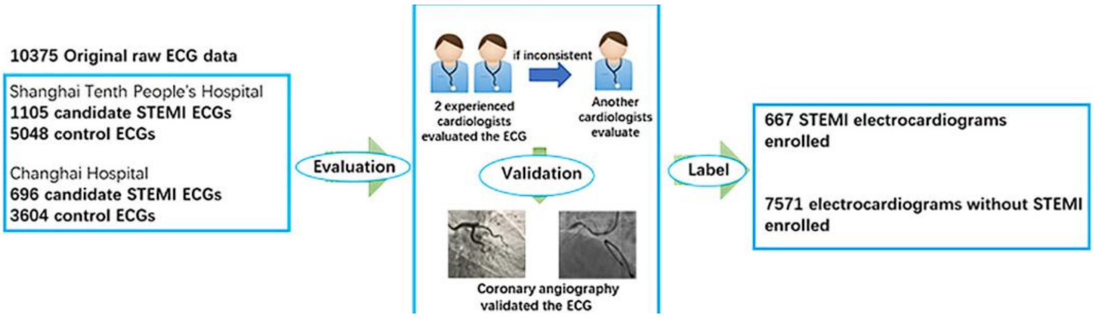

10375 Original raw ECG data
Shanghai Tenth People's Hospital
1105 candidate STEMI ECGs
5048 control ECGs
Changhai Hospital
696 candidate STEMI ECGs
3604 control ECGs
Evaluation
if inconsistent
2 experienced cardiologists evaluated the ECG
Another cardiologists evaluate
Validation
Coronary angiography validated the ECG
Label
667 STEMI electrocardiograms enrolled
7571 electrocardiograms without STEMI enrolled

Figure 2.16 Data Collection [26]

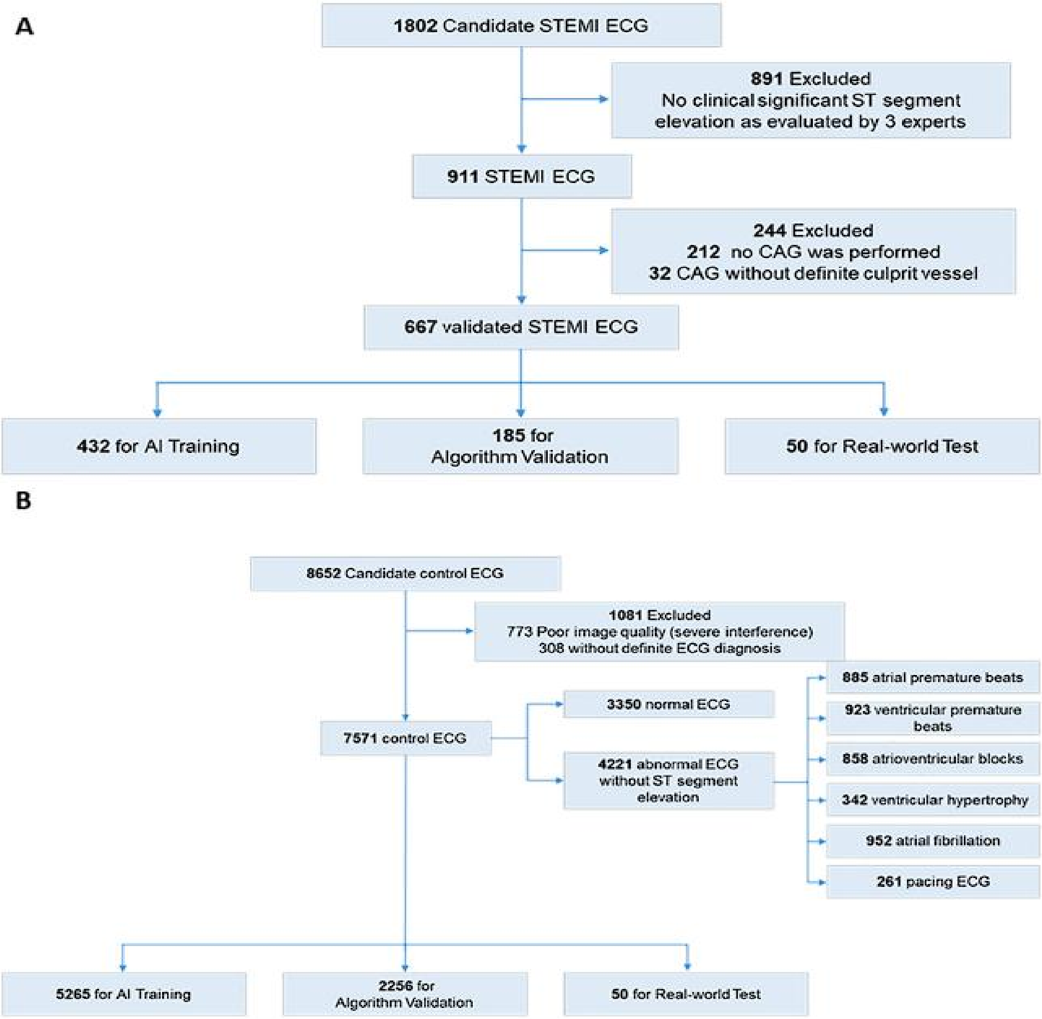

A
1802 Candidate STEMI ECG
891 Excluded
No clinical significant ST segment elevation as evaluated by 3 experts
911 STEMI ECG
244 Excluded
212 no CAG was performed
32 CAG without definite culprit vessel
667 validated STEMI ECG
432 for AI Training
185 for Algorithm Validation
50 for Real-world Test
B
8652 Candidate control ECG
1081 Excluded
773 Poor image quality (severe interference)
308 without definite ECG diagnosis
7571 control ECG
3350 normal ECG
4221 abnormal ECG without ST segment elevation
885 atrial premature beats
923 ventricular premature beats
858 atrioventricular blocks
342 ventricular hypertrophy
952 atrial fibrillation
261 pacing ECG
5265 for AI Training
2256 for Algorithm Validation
50 for Real-world Test

Figure 2.17 Data Collection [26]

The shape of each sample is (5000,12) which meant each of the 12 leads gave 5000 data points. Before feeding into 1-D Res-Net, the data went through the pre-processing steps to remove noise and baseline wandering (**Figure 2.18**) by using the finite impulse response (FIR) and bandpass filter. To increase the amount of data, the data expansion was conducted by firstly downsampling the data from 500Hz to 128Hz by using one-dimensional interpolation together with uniform sampling, in the end, to transform the data to the shape of (1280,12).

Next, they moved the starting point of the original ECG with 1280 data points by continuous 640 points to produce another 640 points in total for each original ECG. The results of expansion were 276,480 STEMI and 3,369,600 control.

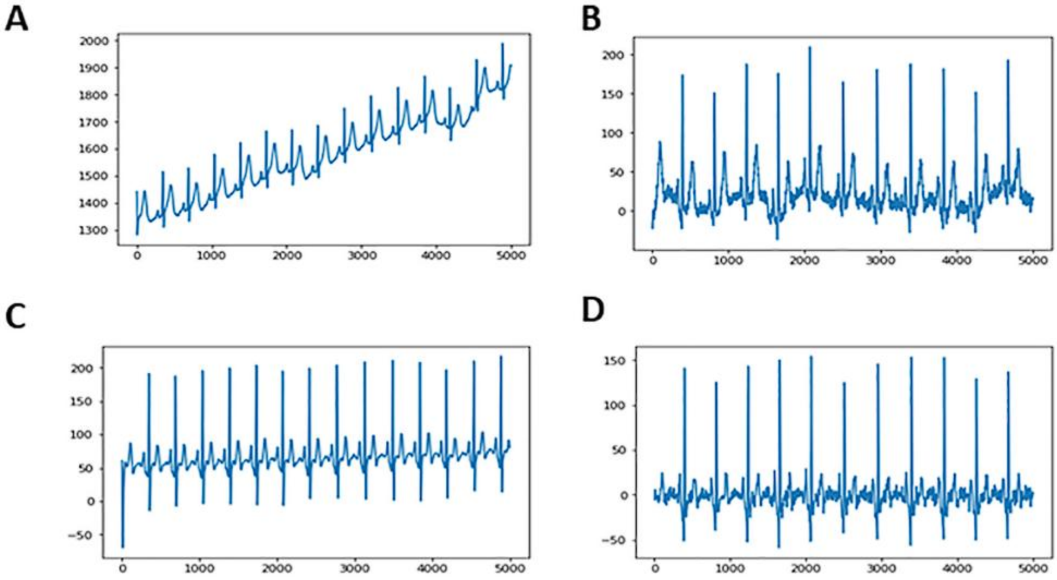


Figure 2.18 C and D are the result of pre-processing steps [26].

The 1D-CNN Res-Net architecture is shown b **Figure 2.19**. There were 16 Res-Net blocks in total and ReLu would be the activation function. The authors set the number of filters to 64 from block 1 and then doubled in blocks 3, 4, 6 and 13. The kernel size was set to 7 in only block 1 while it was 3 in all other blocks. An Adam optimizer with learning rate beta-1, beta-2 and clip norm of 0.001, 0.9, 0.999 and 1 was chosen in the training process.

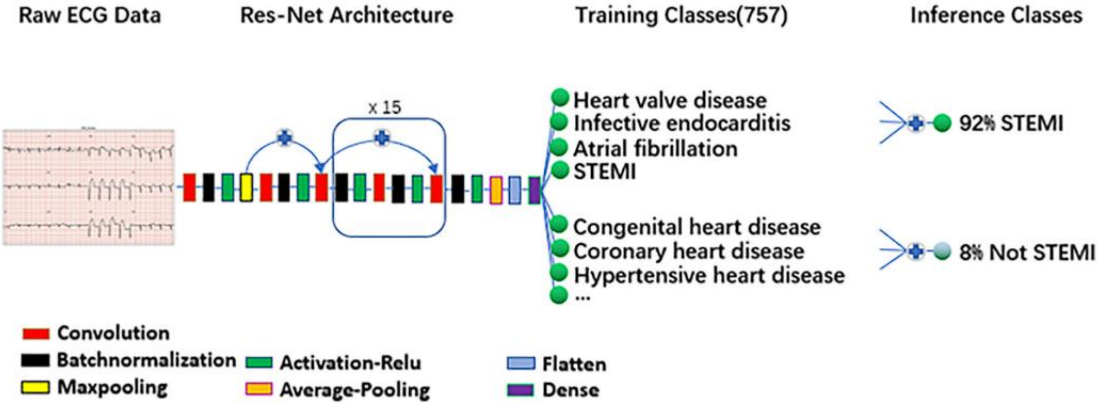


Figure 2.19 CNN architecture [26]

The training process of the 1D-ResNet took 96 hours to complete. After that, a comparative test was conducted between the classifier and 15 doctors including 5 interns, 5 internal medical residents and 5 experienced cardiologists. The research team noticed that in previous studies the ECG is often labelled by ECG experts or cardiologists which is based on human interpretation, this has proven that the result might be false in a higher probability as the patient does not have any angiography result that was applied in this study. The team built 102 different deep learning models in total with different hyperparameters, the model 1 was selected as the final algorithm. The sensitivity, specificity, accuracy and F1 score are 97.22%, 99.11%, 98.96%, 89.94% and 0.9344 respectively. The developed model has outperformed the doctor group and current commercial algorithm (**Figure 2.20**).

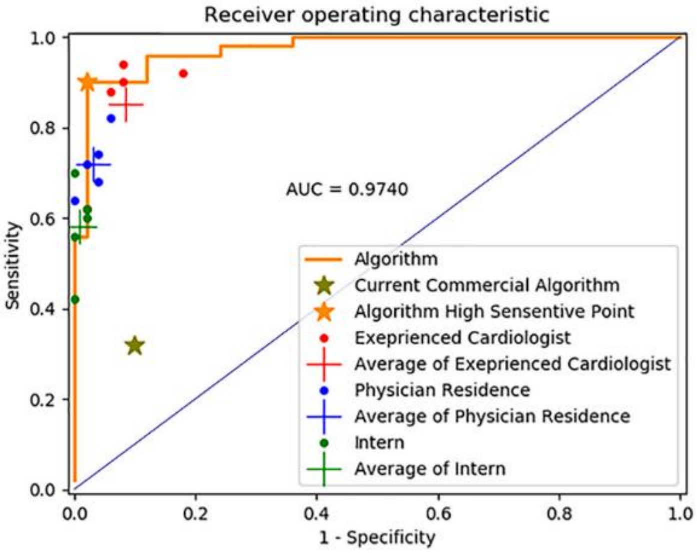


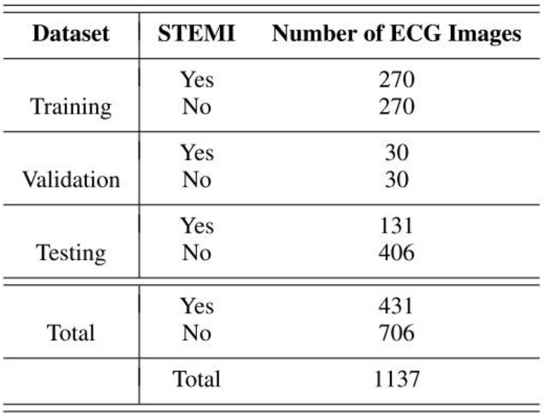
Figure 2.20 Result of comparative test [26]

2.1.5 Application of CNN for Detection and Localization of STEMI Using 12-Lead ECG Images [27]

A 2D-CNN model was proposed in this paper to perform the binary classification task of STEMI detection using ECG images. Due to the high mortality rate of STEMI, the diagnosis must be done within 10 minutes. The authors found in previous studies that 99 physicians only presented 76.9% of sensitivity and 65% of specificity and another 124 physicians presented 65% of sensitivity and 79% of specificity. The disappointing performance of humans gave the motivation to the authors to develop an AI model to automatically detect STEMI by only utilizing ECG images. On top of that, the previous studies that the authors reviewed were mostly focused on the detection or localization of MI instead of STEMI, and there is only one research team conducted a study related to STEMI at that moment. Compared to ML-based methods, DL-based methods do not require manual feature extraction. The 2D-CNN model will take ECG images as input while 1D-CNN and RNN will take ECG waveforms as input. The authors also mentioned that the previous studies mostly used raw

ECG data which is not easy to obtain, sometimes the business-to-business agreement with the hospital and ECG device company was required.

The ECG dataset containing 1137 images of this study was collected from Hualien Tzu Chi Hospital and the data was labelled by the physicians and from the patients who were correctly diagnosed as a STEMI patient. The images were then randomly distributed to the training set, validation set and testing set (**Figure 2.21**). Discussion of public datasets was also available in this paper, the authors pointed out that although many public datasets can be found from PhysioNet such as the Long-Term ST Database (LTST) and PTB Diagnostic ECG Database (PTB-ECG), the former only includes ECG data recorded on two or three leads and both of them are not labelled as STEMI, so they are not suitable for the study. The pre-processing steps of the ECG images in this study are shown in Figure. The OpenCV library together with Python was used to perform steps such as grayscale transformation, padding and thresholding. A summary of the pre-processing steps is shown in **Figure 2.22**.



Dataset	STEMI	Number of ECG Images
Training	Yes	270
	No	270
Validation	Yes	30
	No	30
Testing	Yes	131
	No	406
Total	Yes	431
	No	706
	Total	1137

Figure 2.21 Result of comparative test [27]

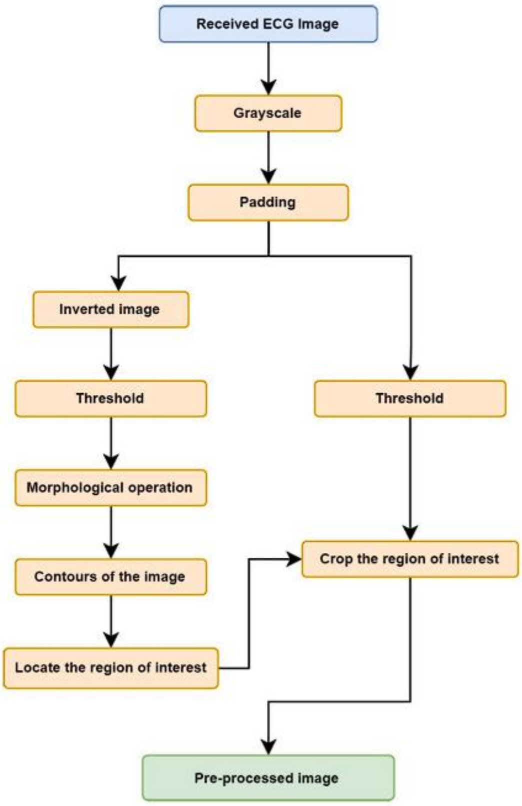
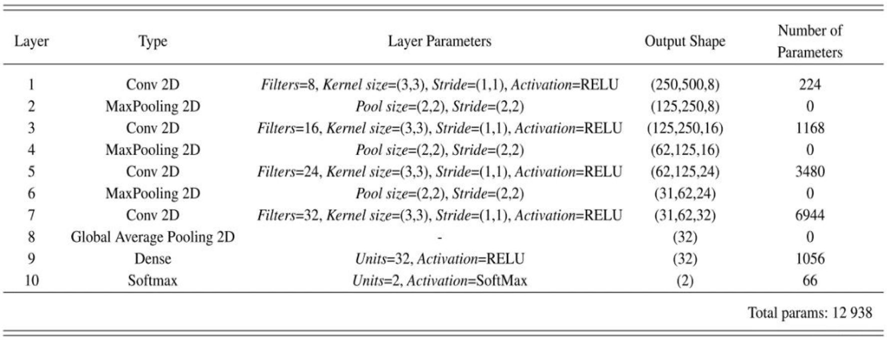


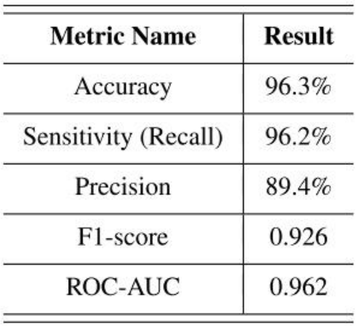
Figure 2.22 Pre-processing steps taken in [27]

For the architecture, the network consists of 10 layers and less than 13000 parameters to show that the computational complexity of the model was low and suitable to apply in low-end computers or devices, but there is no any prove in terms of time for this claim. The details of the network are shown in **Figure 2.23** while the performance is shown in **Figure 2.24**. Then a comparative test between the proposed network and the transfer learning models was experimented and the results were produced as **Figure 2.25** shows that the proposed model outperformed the ResNet, VGG, Xception and so on even though the parameters were a smaller number.



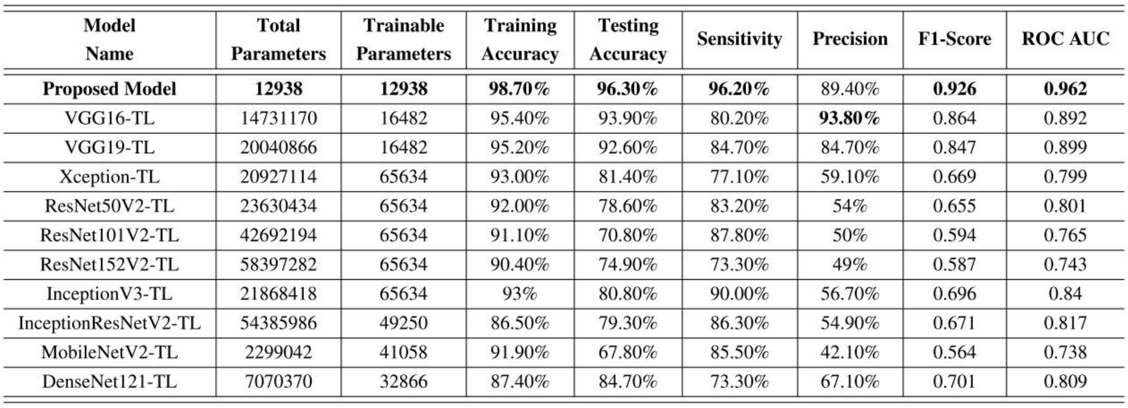
Layer	Type	Layer Parameters	Output Shape	Number of Parameters
1	Conv 2D	*Filters*=8, *Kernel size*=(3,3), *Stride*=(1,1), *Activation*=RELU	(250,500,8)	224
2	MaxPooling 2D	*Pool size*=(2,2), *Stride*=(2,2)	(125,250,8)	0
3	Conv 2D	*Filters*=16, *Kernel size*=(3,3), *Stride*=(1,1), *Activation*=RELU	(125,250,16)	1168
4	MaxPooling 2D	*Pool size*=(2,2), *Stride*=(2,2)	(62,125,16)	0
5	Conv 2D	*Filters*=24, *Kernel size*=(3,3), *Stride*=(1,1), *Activation*=RELU	(62,125,24)	3480
6	MaxPooling 2D	*Pool size*=(2,2), *Stride*=(2,2)	(31,62,24)	0
7	Conv 2D	*Filters*=32, *Kernel size*=(3,3), *Stride*=(1,1), *Activation*=RELU	(31,62,32)	6944
8	Global Average Pooling 2D	-	(32)	0
9	Dense	*Units*=32, *Activation*=RELU	(32)	1056
10	Softmax	*Units*=2, *Activation*=SoftMax	(2)	66
				Total params: 12 938

Figure 2.23 Network architecture of 2D-CNN classifier [27]



Metric Name	Result
Accuracy	96.3%
Sensitivity (Recall)	96.2%
Precision	89.4%
F1-score	0.926
ROC-AUC	0.962

Figure 2.24 Performance of 2D-CNN classifier [27]



Model Name	Total Parameters	Trainable Parameters	Training Accuracy	Testing Accuracy	Sensitivity	Precision	F1-Score	ROC AUC
Proposed Model	**12938**	**12938**	**98.70%**	**96.30%**	**96.20%**	89.40%	**0.926**	**0.962**
VGG16-TL	14731170	16482	95.40%	93.90%	80.20%	**93.80%**	0.864	0.892
VGG19-TL	20040866	16482	95.20%	92.60%	84.70%	84.70%	0.847	0.899
Xception-TL	20927114	65634	93.00%	81.40%	77.10%	59.10%	0.669	0.799
ResNet50V2-TL	23630434	65634	92.00%	78.60%	83.20%	54%	0.655	0.801
ResNet101V2-TL	42692194	65634	91.10%	70.80%	87.80%	50%	0.594	0.765
ResNet152V2-TL	58397282	65634	90.40%	74.90%	73.30%	49%	0.587	0.743
InceptionV3-TL	21868418	65634	93%	80.80%	90.00%	56.70%	0.696	0.84
InceptionResNetV2-TL	54385986	49250	86.50%	79.30%	86.30%	54.90%	0.671	0.817
MobileNetV2-TL	2299042	41058	91.90%	67.80%	85.50%	42.10%	0.564	0.738
DenseNet121-TL	7070370	32866	87.40%	84.70%	73.30%	67.10%	0.701	0.809

Figure 2.25 Result of comparative test between other transfer learning models and the proposed model [27]

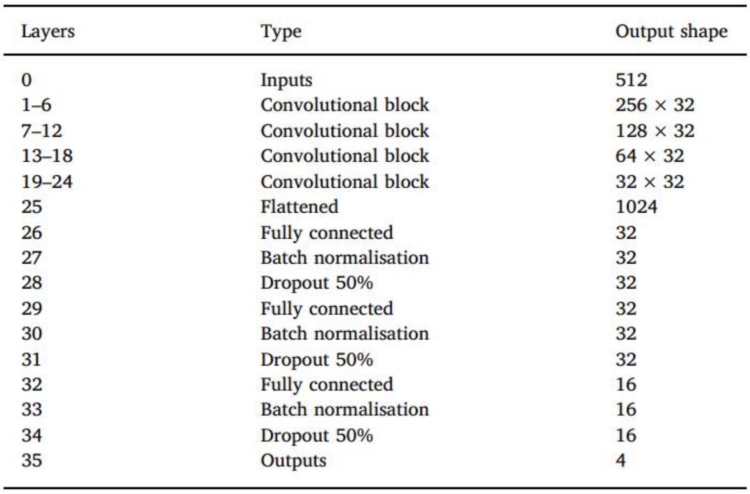
2.1.6 Multiclass classification of myocardial infarction with convolutional and recurrent neural networks for portable ECG devices [28]

In this paper, the authors built six models to have a comparison between them: Hand-crafted features MLP, CNN, CNN-LSTM, and CNN stacking decoding. CNN-LSTM stacking decoding and CNN-LSTM decoding with hand-crafted features. The authors targeted developing an MI classifier that is suitable for wearables, so using single-lead data would be appropriate.

The authors indicated the development trend of ECG devices from bulky machines to portable devices, plus the increment of data volume of ECG due to the growing population, it was not realistic and feasible to rely solely on physicians to detect MI manually. The classifier should be trained to detect MI using ECG from less number of leads. Multiclass rather than binary classification between MI and other healthy and unhealthy (not MI) subjects as well as the noise data was conducted in their research. This is because previous studies that used ECGs from the PTB diagnostic database only included MI patients and healthy subjects, but in real-world situations, some subjects might have historical or existing CVDs other than MI, thus a good classifier had to be trained to differentiate MI from not only healthy subjects but also patients with other CVDs.

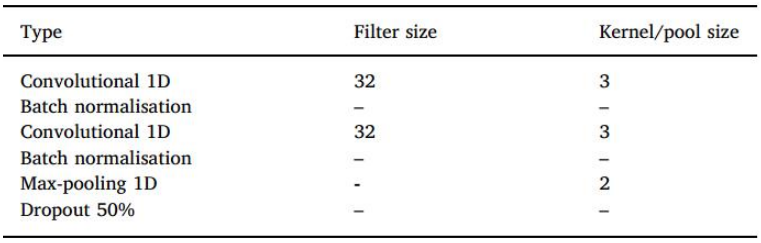
The pre-processing steps of this research comprised two Savitzky-Golay filters which were responsible for denoising and baseline wandering removal, an algorithm with a new nonlinear transform and first-order Gaussian for heartbeat segmentation as well as zero-padding to ensure a consistent length of 512 data point. Moreover, normalization was also applied to the input data to shift the amplitude range from -2500 (min) and 2500 (max) to 0 (min) and 1 (max).

The architecture of the CNN classifier and its convolutional block is shown in **Figure 2.26** and **Figure 2.27.** ReLu activation function was used in the convolutional block and the softmax function was used in the output layer. The CNN-LSTM classifier was obtained by modifying layer 29 of CNN architecture from fully connected to LSTM (**Figure 2.28**). Instead of only using the result produced from four units of output layer to perform multiclass classification, the authors deployed the stacking decoding technique. Multiple binary classifiers will be trained for each pair of available classes, then a meta-classifier can be produced and trained (**Figure 2.29**) using the two inputs of each binary classifier (total input =12).



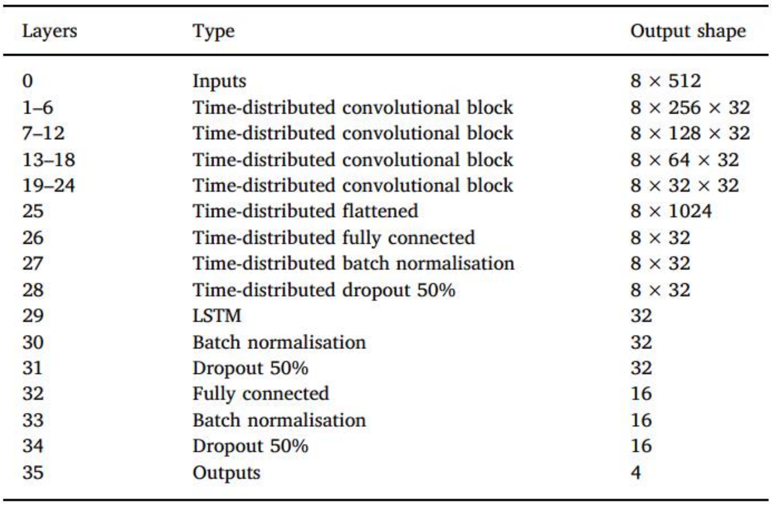
Layers	Type	Output shape
0	Inputs	512
1–6	Convolutional block	256 × 32
7–12	Convolutional block	128 × 32
13–18	Convolutional block	64 × 32
19–24	Convolutional block	32 × 32
25	Flattened	1024
26	Fully connected	32
27	Batch normalisation	32
28	Dropout 50%	32
29	Fully connected	32
30	Batch normalisation	32
31	Dropout 50%	32
32	Fully connected	16
33	Batch normalisation	16
34	Dropout 50%	16
35	Outputs	4

Figure 2.26 CNN architecture [28]



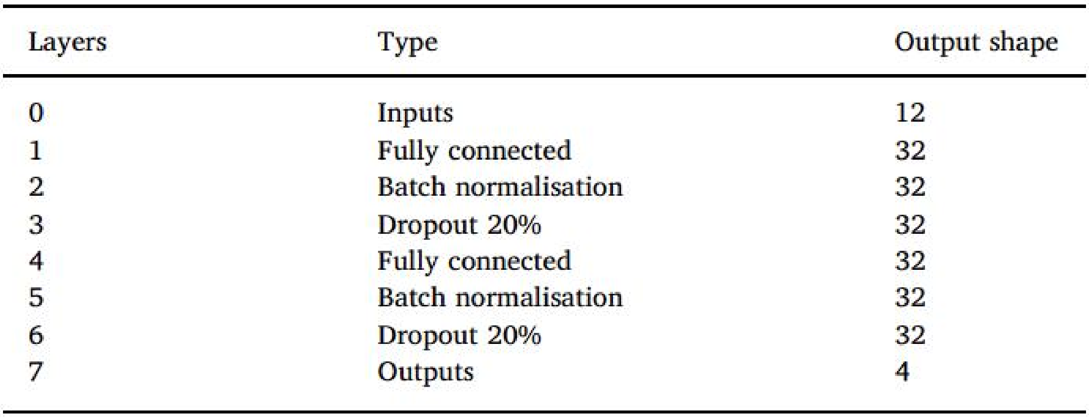
Type	Filter size	Kernel/pool size
Convolutional 1D	32	3
Batch normalisation	–	–
Convolutional 1D	32	3
Batch normalisation	–	–
Max-pooling 1D	-	2
Dropout 50%	–	–

Figure 2.27 Convolutional block architecture [28]



Layers	Type	Output shape
0	Inputs	8 × 512
1–6	Time-distributed convolutional block	8 × 256 × 32
7–12	Time-distributed convolutional block	8 × 128 × 32
13–18	Time-distributed convolutional block	8 × 64 × 32
19–24	Time-distributed convolutional block	8 × 32 × 32
25	Time-distributed flattened	8 × 1024
26	Time-distributed fully connected	8 × 32
27	Time-distributed batch normalisation	8 × 32
28	Time-distributed dropout 50%	8 × 32
29	LSTM	32
30	Batch normalisation	32
31	Dropout 50%	32
32	Fully connected	16
33	Batch normalisation	16
34	Dropout 50%	16
35	Outputs	4

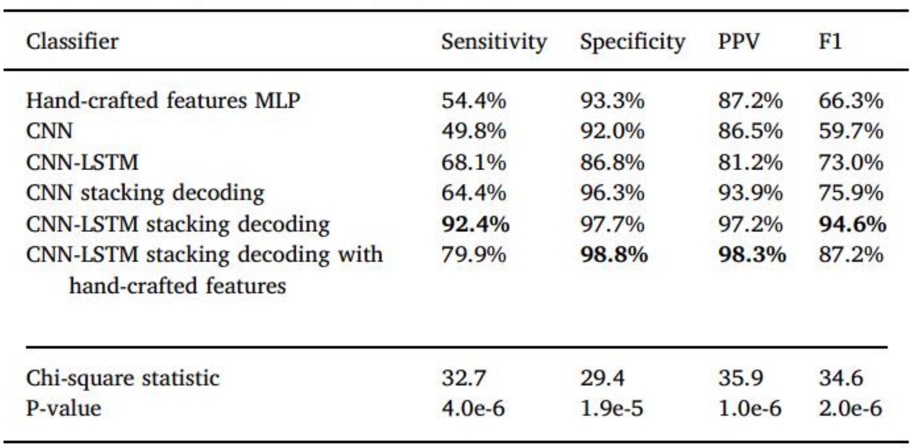
Figure 2.28 CNN-LSTM architecture [28]



Layers	Type	Output shape
0	Inputs	12
1	Fully connected	32
2	Batch normalisation	32
3	Dropout 20%	32
4	Fully connected	32
5	Batch normalisation	32
6	Dropout 20%	32
7	Outputs	4

Figure 2.29 Meta-classifier architecture [28]

For the purpose of comparison, the authors also developed a traditional model of using an MLP classifier with hand-crafted features and its architecture was the same as a meta-classifier. In the end, the performance metrics of each classifier are shown in **Figure 2.30**.

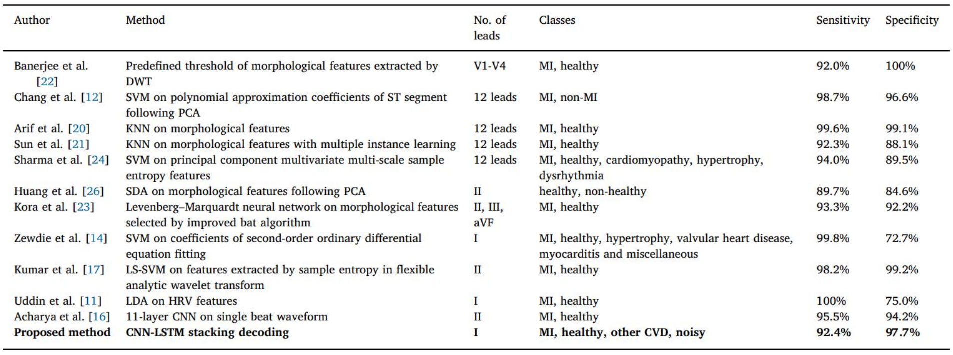


Classifier	Sensitivity	Specificity	PPV	F1
Hand-crafted features MLP	54.4%	93.3%	87.2%	66.3%
CNN	49.8%	92.0%	86.5%	59.7%
CNN-LSTM	68.1%	86.8%	81.2%	73.0%
CNN stacking decoding	64.4%	96.3%	93.9%	75.9%
CNN-LSTM stacking decoding	**92.4%**	97.7%	97.2%	**94.6%**
CNN-LSTM stacking decoding with hand-crafted features	79.9%	**98.8%**	**98.3%**	87.2%
Chi-square statistic	32.7	29.4	35.9	34.6
P-value	4.0e-6	1.9e-5	1.0e-6	2.0e-6

Figure 2.30 Performance of each classifier [28]

The authors mentioned that the computational time for MI detection of the best-performed model, CNN-LSTM with stacking decoding required 350ms.

In the comparison with models proposed by other studies (**Figure 2.31**), the authors stated that many of the models might not have the ability to differentiate MI from other CVDs. For the models developed using multiclass ECG, although their sensitivity outperformed the proposed CNN-LSTM with stacking decoding, their specificity was lower. Specificity becomes important when the classification task is changed from binary class to multiclass as this performance metric shows the ability of the model to differentiate MI from other CVDs. Another advantage of this research was they collected noisy data to train their proposed model to ensure the model is applicable in real-world scenes.



Author	Method	No. of leads	Classes	Sensitivity	Specificity
Banerjee et al. [22]	Predefined threshold of morphological features extracted by DWT	V1-V4	MI, healthy	92.0%	100%
Chang et al. [12]	SVM on polynomial approximation coefficients of ST segment following PCA	12 leads	MI, non-MI	98.7%	96.6%
Arif et al. [20]	KNN on morphological features	12 leads	MI, healthy	99.6%	99.1%
Sun et al. [21]	KNN on morphological features with multiple instance learning	12 leads	MI, healthy	92.3%	88.1%
Sharma et al. [24]	SVM on principal component multivariate multi-scale sample entropy features	12 leads	MI, healthy, cardiomyopathy, hypertrophy, dysrhythmia	94.0%	89.5%
Huang et al. [26]	SDA on morphological features following PCA	II	healthy, non-healthy	89.7%	84.6%
Kora et al. [23]	Levenberg–Marquardt neural network on morphological features selected by improved bat algorithm	II, III, aVF	MI, healthy	93.3%	92.2%
Zewdie et al. [14]	SVM on coefficients of second-order ordinary differential equation fitting	I	MI, healthy, hypertrophy, valvular heart disease, myocarditis and miscellaneous	99.8%	72.7%
Kumar et al. [17]	LS-SVM on features extracted by sample entropy in flexible analytic wavelet transform	II	MI, healthy	98.2%	99.2%
Uddin et al. [11]	LDA on HRV features	I	MI, healthy	100%	75.0%
Acharya et al. [16]	11-layer CNN on single beat waveform	II	MI, healthy	95.5%	94.2%
Proposed method	**CNN-LSTM stacking decoding**	**I**	**MI, healthy, other CVD, noisy**	**92.4%**	**97.7%**

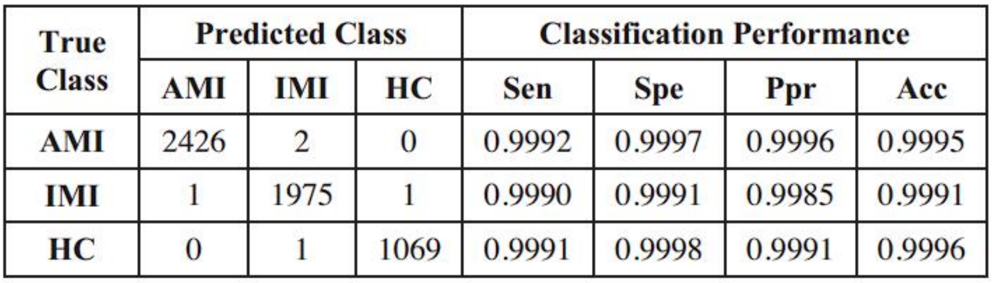
Figure 2.31 Comparison of performance between the proposed model and other models [28]

2.1.7 A New Automatic Approach to Distinguish Myocardial Infarction Based on LSTM [29]

In order to lower the misdiagnosis rate of MI which resulted from human interpretation, an LSTM model was proposed in this research. The authors mentioned that in previous studies that focused on traditional machine-learning-based classification, the design of the algorithm was largely dependent on the knowledge of the designer while the deep-learning-based classifier can be trained by learning features from the data automatically itself. The authors agreed that using single-lead ECG data could speed up the classification speed but it sacrificed the accuracy as the model suffered from useful information loss.

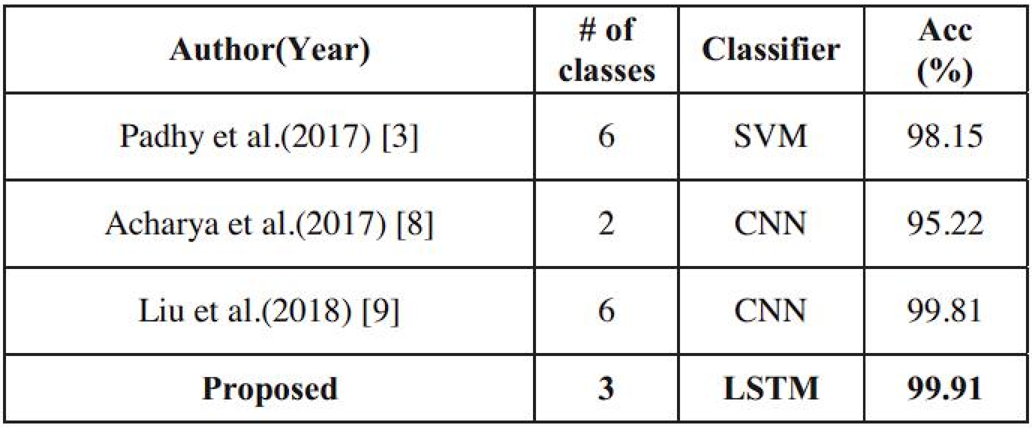
This research only utilized data from 8 leads and MI was classified as Anterior MI and Inferior MI. The pre-processing steps used in this paper included median filtering for baseline wandering removal, a notch filter and Chebyshev digital low-pass filter for denoise (power-frequency interference), and the Pan-Tompkins algorithm for QRS complex detection. The signals were then segmented from 250ms left and 400ms right of the R-peak location.

In this paper, the network architecture for LSTM was not shown. The performance of the classifier and the result of the comparative test are shown in **Figure 2.32** and **Figure 2.33.**



True Class	Predicted Class			Classification Performance			
	AMI	IMI	HC	Sen	Spe	Ppr	Acc
AMI	2426	2	0	0.9992	0.9997	0.9996	0.9995
IMI	1	1975	1	0.9990	0.9991	0.9985	0.9991
HC	0	1	1069	0.9991	0.9998	0.9991	0.9996

Figure 2.32 Performance of the LSTM model [29]



Author(Year)	# of classes	Classifier	Acc (%)
Padhy et al.(2017) [3]	6	SVM	98.15
Acharya et al.(2017) [8]	2	CNN	95.22
Liu et al.(2018) [9]	6	CNN	99.81
Proposed	**3**	**LSTM**	**99.91**

Figure 2.33 Comparison of performance between proposed LSTM and other proposed models [29]

2.1.8 Deep Learning with Long Short-Term Memory for Enhancement Myocardial Infarction Classification [30]

The authors found that the main weakness of traditional machine learning was the application of hand-engineered features, so they decided to propose a deep-learning-based algorithm as it will automatically enable the feature learning process without human intervention. In this paper, the authors utilized balanced accuracy (BAcc) and Matthew’s Correlation Coefficient (MCC) to tackle with the data imbalance problem.

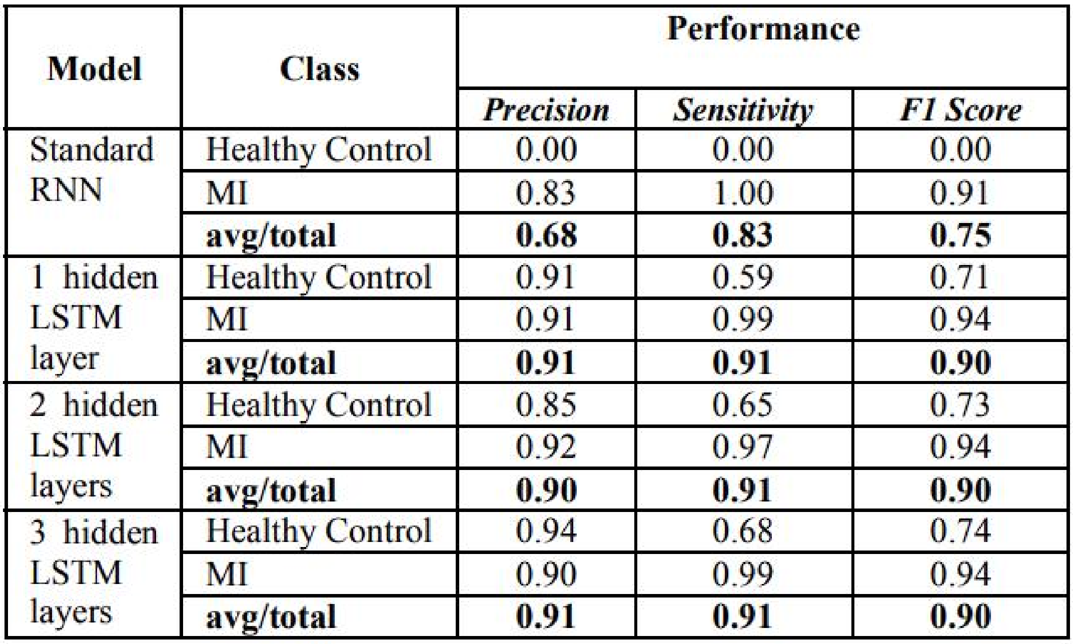
$$BAcc = \frac{(\frac{TP}{P}+\frac{TN}{N})}{2}$$

$$MCC = \frac{(TP\ x\ TN)-(FP\ x\ FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

Figure 2.34 Formula of balanced accuracy and Matthew's Correlation Coefficient [30]

The data source of this research was also the PTB diagnostic ECG database, the pre-processing steps were not mentioned in the paper but the segmentation was conducted to get data with a consistent length of 4000 data points. The authors then explained the principle behind the RNN model to highlight the vanishing gradient problem of RNN. Other than that, the authors also showed the details of the forward pass and backward pass of the LSTM network.

The total data that could be used for training and testing were 12,359 with 10,144 of them being MI and the rest being healthy subjects. There were four models had been developed: Standard RNN and 1-/2-/3-layered LSTM. It can be observed that all of the models were weak in detecting healthy subjects (**Figure 2.35**).



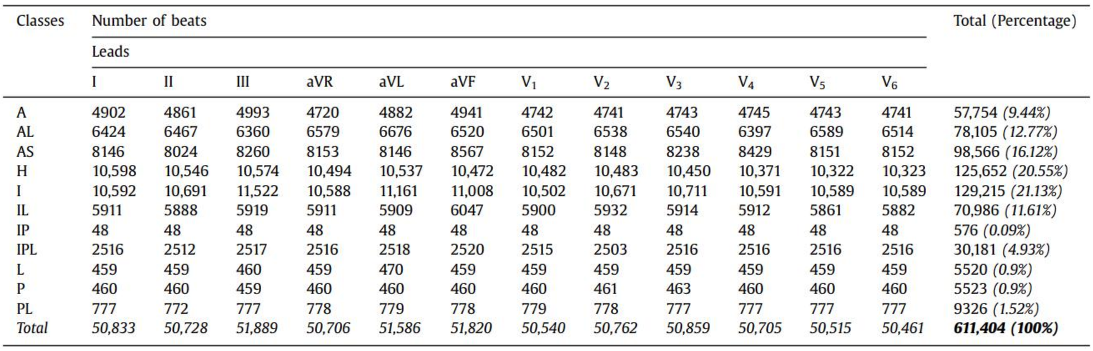
Model	**Class**	**Performance**		
		Precision	***Sensitivity***	***F1 Score***
Standard RNN	Healthy Control	0.00	0.00	0.00
	MI	0.83	1.00	0.91
	avg/total	**0.68**	**0.83**	**0.75**
1 hidden LSTM layer	Healthy Control	0.91	0.59	0.71
	MI	0.91	0.99	0.94
	avg/total	**0.91**	**0.91**	**0.90**
2 hidden LSTM layers	Healthy Control	0.85	0.65	0.73
	MI	0.92	0.97	0.94
	avg/total	**0.90**	**0.91**	**0.90**
3 hidden LSTM layers	Healthy Control	0.94	0.68	0.74
	MI	0.90	0.99	0.94
	avg/total	**0.91**	**0.91**	**0.90**

Figure 2.35 Performance of each model in classification [30]

2.1.9 Classification of myocardial infarction with multi-lead ECG signals and deep CNN [31]

Similarly, the motivation of this research was to propose an end-to-end deep-learning-based model that could be implemented on portable healthcare devices to eliminate the step of manual feature extraction. In this research, the authors also care about the MI locations.

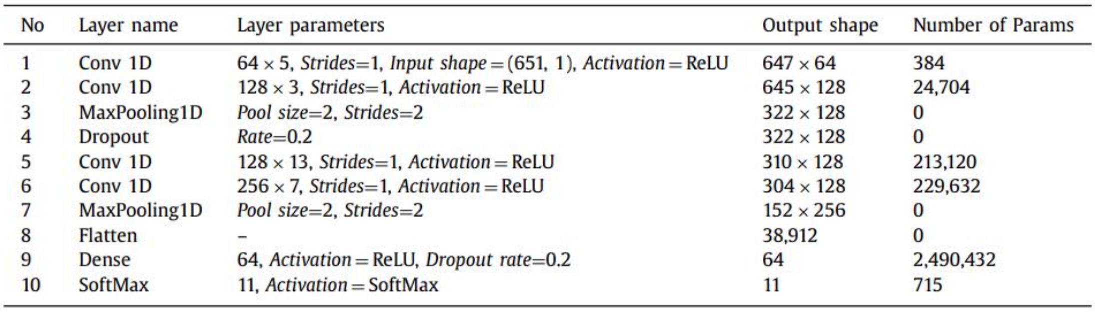
The data source of this research was still the PTB Diagnostic ECG database. The authors applied a wavelet transform algorithm to denoise and remove baseline wandering. The data was then segmented into heartbeats, the length of the sequence was 651 for each sample (250 data points to the left and 400 to the right according to R-peak). The total number of heartbeats was 611,404 while 20.55% of them were healthy group. The distribution of classes is shown in **Figure 2.33**: anterior (A), anterior lateral (AL), anterior septal (AS), inferior (I), inferior lateral (IL), inferior posterior (IP), inferior posterior lateral (IPL), lateral (L), posterior (P) and posterior lateral (PL).



Classes	Number of beats												Total (Percentage)
	Leads												
	I	II	III	aVR	aVL	aVF	V_1	V_2	V_3	V_4	V_5	V_6	
A	4902	4861	4993	4720	4882	4941	4742	4741	4743	4745	4743	4741	57,754 *(9.44%)*
AL	6424	6467	6360	6579	6676	6520	6501	6538	6540	6397	6589	6514	78,105 *(12.77%)*
AS	8146	8024	8260	8153	8146	8567	8152	8148	8238	8429	8151	8152	98,566 *(16.12%)*
H	10,598	10,546	10,574	10,494	10,537	10,472	10,482	10,483	10,450	10,371	10,322	10,323	125,652 *(20.55%)*
I	10,592	10,691	11,522	10,588	11,161	11,008	10,502	10,671	10,711	10,591	10,589	10,589	129,215 *(21.13%)*
IL	5911	5888	5919	5911	5909	6047	5900	5932	5914	5912	5861	5882	70,986 *(11.61%)*
IP	48	48	48	48	48	48	48	48	48	48	48	48	576 *(0.09%)*
IPL	2516	2512	2517	2516	2518	2520	2515	2503	2516	2516	2516	2516	30,181 *(4.93%)*
L	459	459	460	459	470	459	459	459	459	459	459	459	5520 *(0.9%)*
P	460	460	459	460	460	460	460	461	463	460	460	460	5523 *(0.9%)*
PL	777	772	777	778	779	778	779	778	777	777	777	777	9326 *(1.52%)*
Total	*50,833*	*50,728*	*51,889*	*50,706*	*51,586*	*51,820*	*50,540*	*50,762*	*50,859*	*50,705*	*50,515*	*50,461*	***611,404 (100%)***

Figure 2.36 Distribution of classes in each lead [31]

A 10-layered CNN was built according to the architecture in **Figure 2.37** and **Figure 2.38** to perform the multiclass classification task and thus categorical cross-entropy loss function was used. Adam optimizer with a learning rate of 10^{-3} and 10^{-4} was used during the training phase. To prevent the overfitting issue, the model was not trained using a large number of epochs.



No	Layer name	Layer parameters	Output shape	Number of Params
1	Conv 1D	64 × 5, *Strides*=1, *Input shape* = (651, 1), *Activation* = ReLU	647 × 64	384
2	Conv 1D	128 × 3, *Strides*=1, *Activation* = ReLU	645 × 128	24,704
3	MaxPooling1D	*Pool size*=2, *Strides*=2	322 × 128	0
4	Dropout	*Rate*=0.2	322 × 128	0
5	Conv 1D	128 × 13, *Strides*=1, *Activation* = ReLU	310 × 128	213,120
6	Conv 1D	256 × 7, *Strides*=1, *Activation* = ReLU	304 × 128	229,632
7	MaxPooling1D	*Pool size*=2, *Strides*=2	152 × 256	0
8	Flatten	–	38,912	0
9	Dense	64, *Activation* = ReLU, *Dropout rate*=0.2	64	2,490,432
10	SoftMax	11, *Activation* = SoftMax	11	715

Figure 2.37 CNN architecture [31]

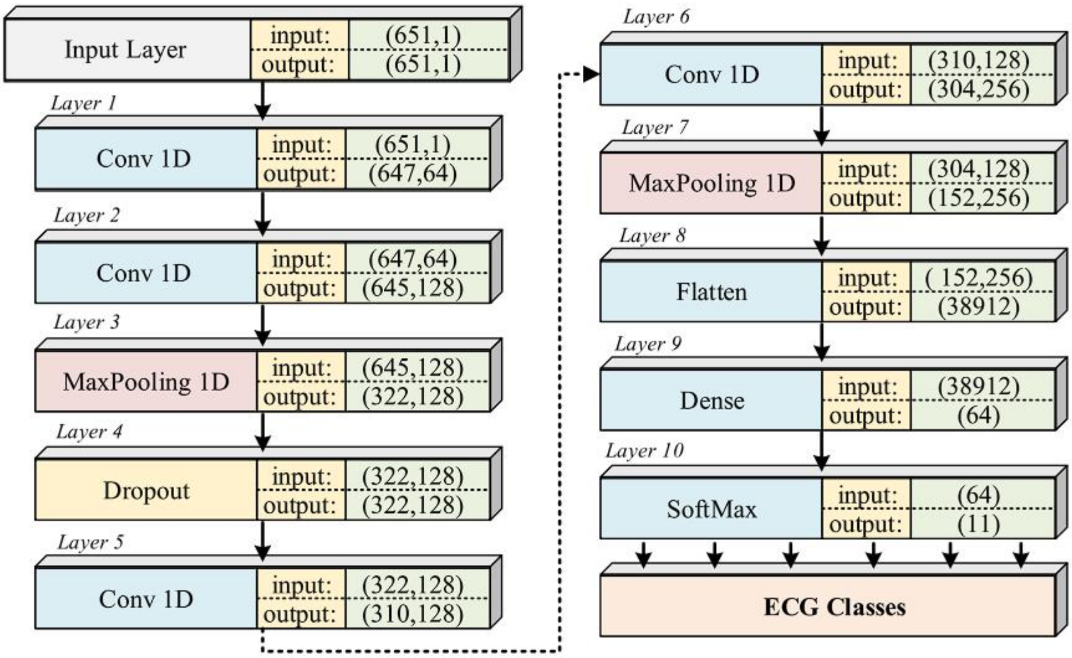
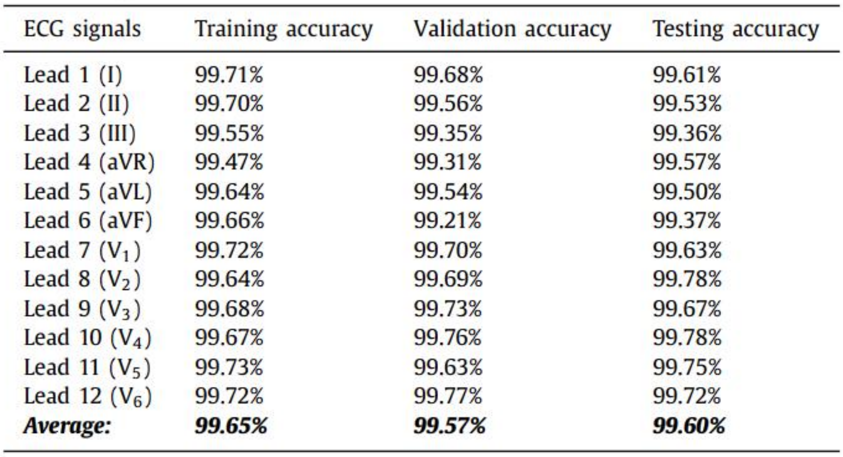


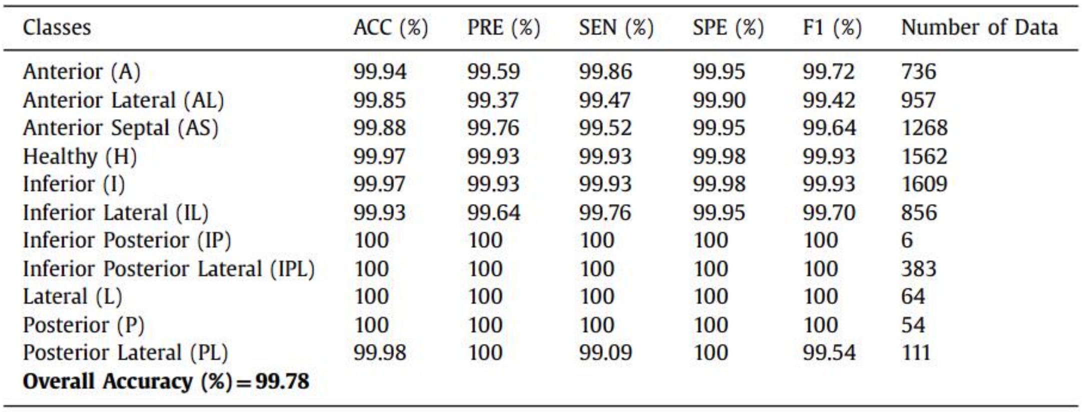
Figure 2.38 CNN architecture [31]

Overall, the CNN model achieved a very good result in every lead. (**Figure 2.39**). The model was able to differentiate MI in different locations and also the healthy subjects. Lead V4 performed the best in this research (**Figure 2.40**).



ECG signals	Training accuracy	Validation accuracy	Testing accuracy
Lead 1 (I)	99.71%	99.68%	99.61%
Lead 2 (II)	99.70%	99.56%	99.53%
Lead 3 (III)	99.55%	99.35%	99.36%
Lead 4 (aVR)	99.47%	99.31%	99.57%
Lead 5 (aVL)	99.64%	99.54%	99.50%
Lead 6 (aVF)	99.66%	99.21%	99.37%
Lead 7 (V_1)	99.72%	99.70%	99.63%
Lead 8 (V_2)	99.64%	99.69%	99.78%
Lead 9 (V_3)	99.68%	99.73%	99.67%
Lead 10 (V_4)	99.67%	99.76%	99.78%
Lead 11 (V_5)	99.73%	99.63%	99.75%
Lead 12 (V_6)	99.72%	99.77%	99.72%
Average:	***99.65%***	***99.57%***	***99.60%***

Figure 2.39 Accuracy achieved by CNN in each lead [31]



Classes	ACC (%)	PRE (%)	SEN (%)	SPE (%)	F1 (%)	Number of Data
Anterior (A)	99.94	99.59	99.86	99.95	99.72	736
Anterior Lateral (AL)	99.85	99.37	99.47	99.90	99.42	957
Anterior Septal (AS)	99.88	99.76	99.52	99.95	99.64	1268
Healthy (H)	99.97	99.93	99.93	99.98	99.93	1562
Inferior (I)	99.97	99.93	99.93	99.98	99.93	1609
Inferior Lateral (IL)	99.93	99.64	99.76	99.95	99.70	856
Inferior Posterior (IP)	100	100	100	100	100	6
Inferior Posterior Lateral (IPL)	100	100	100	100	100	383
Lateral (L)	100	100	100	100	100	64
Posterior (P)	100	100	100	100	100	54
Posterior Lateral (PL)	99.98	100	99.09	100	99.54	111
Overall Accuracy (%) = 99.78						

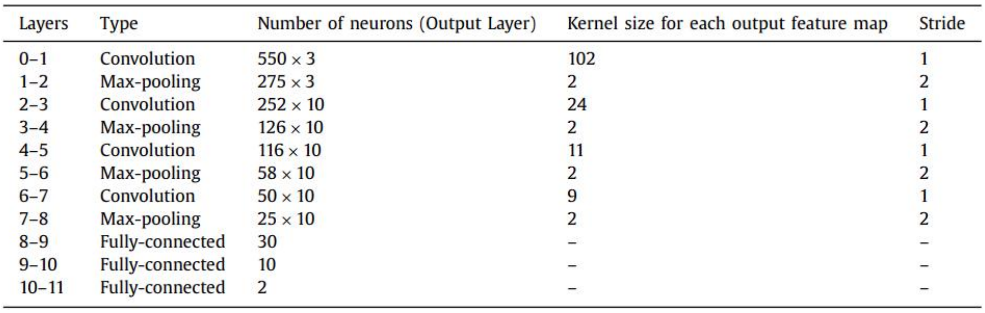
Figure 2.40 Performance metrics achieved by lead V4 [31]

2.1.10 Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals [32]

In this research, the authors claimed that compared to previous studies, no denoising operation was required as their algorithm can detect MI in the presence of noise. A comparative test was conducted using different sets of ECG (denoised / not denoised).

The authors collected the data from the PTB Diagnostic ECG database. The data is then split into two sets. One of them would undergo denoise and baseline wandering removal by using Daubechies wavelet 6 mother wavelet function while the noises were kept in the other set. Pan-Tompkins algorithm was applied to both datasets to segment the heartbeats with a consistent length of 651 data points.

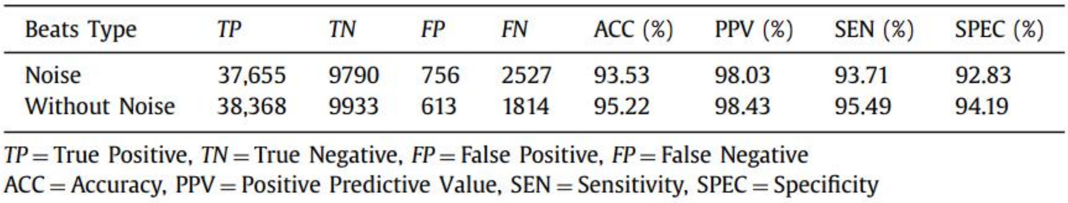
The CNN architecture is shown in **Figure 2.41**. LeakyRelu was used in the convolutional layer and Softmax was used in the last FC layer. The type of the optimizer was unclear in this research but not its parameters: regularization (avoid overfitting), momentum (speed of learning), and learning rate (speed of convergence) parameters were set to 0.2, 3×10^{4}, and 0.7 respectively.



Layers	Type	Number of neurons (Output Layer)	Kernel size for each output feature map	Stride
0–1	Convolution	550 × 3	102	1
1–2	Max-pooling	275 × 3	2	2
2–3	Convolution	252 × 10	24	1
3–4	Max-pooling	126 × 10	2	2
4–5	Convolution	116 × 10	11	1
5–6	Max-pooling	58 × 10	2	2
6–7	Convolution	50 × 10	9	1
7–8	Max-pooling	25 × 10	2	2
8–9	Fully-connected	30	–	–
9–10	Fully-connected	10	–	–
10–11	Fully-connected	2	–	–

Figure 2.41 CNN architecture [32]

From Figure 2.39, it can be observed that the performance of the model using a dataset with noise dropped by a small degree. This proved that the model was robust to noise.



Beats Type	*TP*	*TN*	*FP*	*FN*	ACC (%)	PPV (%)	SEN (%)	SPEC (%)
Noise	37,655	9790	756	2527	93.53	98.03	93.71	92.83
Without Noise	38,368	9933	613	1814	95.22	98.43	95.49	94.19

TP = True Positive, *TN* = True Negative, *FP* = False Positive, *FP* = False Negative
ACC = Accuracy, PPV = Positive Predictive Value, SEN = Sensitivity, SPEC = Specificity

Figure 2.42 Performance metrics of each dataset [32]

2.2 Summary of previous 10 works

Table 2.1 Summary of previous works

	Model	**Lead**	**Data Size**	**Inter/Intra**	**Performance**
[23]	LSTM	II	Heartbeats: 50732 MI 10123 HC	Inter	ACC = 0.89 SEN = 0.91 SPEC = 0.80
[16]	-1D-CNN -LSTM -CNN-LSTM	12-lead	Subjects **Set 1** 315 STEMI 478 Control **Set 2** 62 STEMI 28 Control	Not mentioned	**CNN-LSTM** ACC = 0.98 SEN = 0.97 SPEC = 0.97 F1 = 0.98 **LSTM** ACC = 0.83 SEN = 0.78 SPEC = 0.80 F1 = 0.84 **CNN** ACC = 0.87 SEN = 0.90 SPEC = 0.91 F1 = 0.87
[25]	1D-CNN	12-lead	Heartbeats:	Intra	**12-lead**

		V2	46,056 STEMI 23,301 Abnormal 21,235 Normal		ACC = 0.96 SEN = 0.96 SPEC = 0.96 **Lead V2** ACC = 0.90 SEN = 0.86 SPEC = 0.94
[26]	1D-CNN (Res-Net)	12-lead	276,480 STEMI 3,369,600 Normal	Not mentioned	ACC = 0.99 SEN = 0.96 SPEC = 0.99 F1 = 0.9372
[27]	2D-CNN	12-lead	Images: 540 Training 537 Testing	Not mentioned	ACC = 0.96 SEN = 0.96 PRE = 0.89 F1 = 0.926
[28]	-MLP -CNN -CNN stacking decoding -CNN-LSTM -CNN-LSTM stacking decoding -CNN-LSTM stacking decoding with hand-crafted features	I	PTB: 549 records Cardiology Challenge 2017 database (AF-Challenge): 8528 records	Not mentioned	Due to space constraints, can refer to **Figure 2.27**
[29]	LSTM	8-lead	Heartbeats: 54,753	Not mentioned	ACC = 0.99

[30]	RNN LSTM	Not mentioned	Signals: 12,359	Not mentioned	BAcc = 0.83 MCC = 0.75 PRE = 0.91 SEN = 0.91 F1 = 0.90
[31]	1D-CNN	12-leads	Heartbeats: 611,404	Not mentioned	V4 ACC = 0.99
[32]	1D-CNN	II	Heartbeats: 10,546 normal 40,182 MI	Intra	Noise ACC = 0.93 SEN = 0.93 SPEC = 0.92 Wihout Noise ACC = 0.95 SEN = 0.95 SPEC = 0.94

Chapter 3

System Design & Model

3.1 System Design Diagram

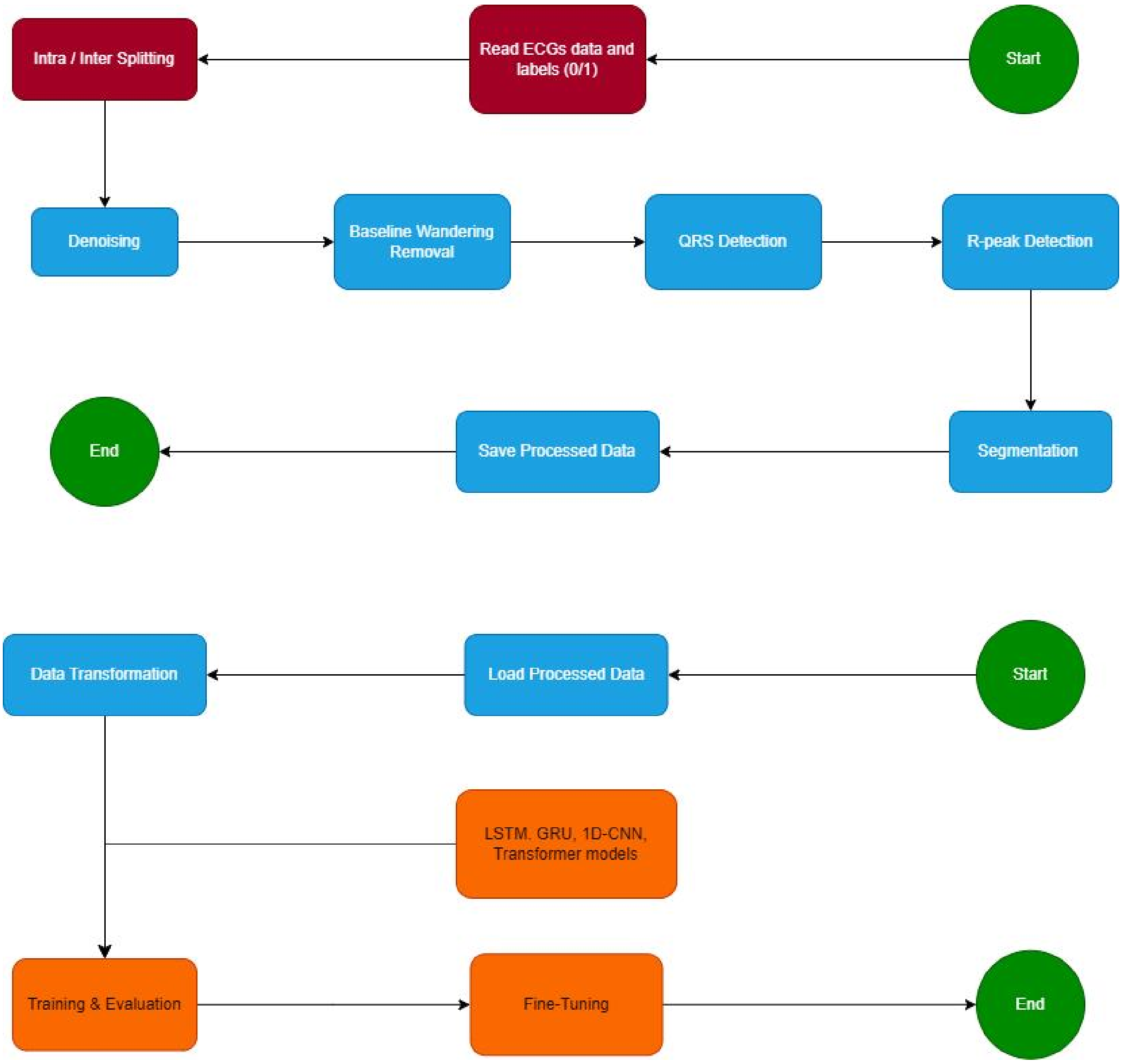


Figure 3.1 System design of this project

As shown in **Figure 3.1**, first and foremost, the dataset is downloaded from the open-source database, PTB Diagnostic ECG Database which is available on the PhysioNet website [24]. The labelling of each record will be automatically executed by reading the header file which contains a section indicating the diagnosis of the record. Besides, the header file also included age, gender, data on medical history, etc. The dataset consists of 148 MI patients and 52 normal subjects and one subject might contribute more than one record to form a total number of 448 ECG records. The sampling rate of all the records of this dataset was 1000 Hz frequency which means there would be 1000 data points recorded at 1-second intervals. A higher sampling rate represents more information about the heart's activities that can be described and recorded, thus more information is available for models to learn a complex decision function.

After that, to ensure the inter-patient splitting, the splitting operation to produce the 10-fold cross-validation of training set, validation set and testing set must be performed at the patient level instead of at the heartbeat level. Instead, the intra-patient splitting can be done if the operation is performed at the heartbeat level. To simulate the real-world settings and according to the documentation of the PhysioNet website [24], it was assumed that most of the ECGs suffered from noises and baseline wandering in this project possibly due to the movement of the patient during the skin resistance, measurement, patient respiration or poor electrode (lead) contact [24, 32]. Thus, some crucial pre-processing steps were necessary. Then, two Savitzky-Golay (SG) filters [36] were applied to raw signals in each fold to perform denoise and baseline wandering removal operations. Next, the Pan-Tompkins algorithm [37] was utilized to locate the QRS region in the filtered ECG signal in an effort to eventually find the exact R-peak location for the segmentation purpose. Due to the difference in length of the signal in each record, segmentation is a necessary action as the model cannot be trained using signals with inconsistent lengths. Therefore, an accurate R-peak location can be used as a standard to segment the continuous signal into heartbeats with a consistent sequence length of 800 data points. From the R-peak, 300 data points before and 500 data points after would compose a heartbeat containing a PQRST complex and its visualization would be presented in Chapter 4. The 800 points of a signal The SG filter and Pan-Tompkins algorithm were chosen to perform the respective pre-processing tasks because of their popularity in other studies [22, 28, 29, 32] and it works well in this research (Figure 4.7 & Figure 4.8). During the implementation, it was expected to take a long time to perform all actions mentioned above to become a proper training set, validation set and testing set, thus it was necessary to save the processed heartbeats somewhere such as Google Drive.

Every time the project starts, the processed heartbeats of each fold can be loaded without undergoing any pre-processing steps. A simple transformation was required to encapsulate the processed heartbeats into batches. The heartbeats will then be fed into the LSTM, GRU, 1D-CNN and Transformer batches by batches for training, validation and testing to observe the performance. After this, a random search method was used in the fine-tuning process to search for the best hyperparameters of each model that helped solve the underfitting and overfitting problem with the final goal of getting the best performance in the testing phase. The hyperparameters list included learning rate, weight decay (regularization), dropout rate, number of neural units in fully connected layers, etc.

3.2 System Architecture Diagram

The following diagrams solely show the architecture and parameters of each model/classifier. An FC network consisting of three FC layers was included in each type of model. The default activation function for the Fully Connected (FC) network in this project was set to ReLU. A sigmoid function was applied in the output layer to produce a binary classification result, 0 or 1. The number of units of the first FC layer was set to 512 and became half for each following layer, 256 and 128.

3.2.1 LSTM classifier

The input of the first LSTM [39] layer should be in the shape of a 3D tensor (Batch Size, Sequence Length, Dimensionality) so it would be for example, (128, 800, 1) where the dimensionality depends on how many leads were used in the research. The LSTM block contains 256 units (hidden_size = 256) and 2 layers. There were two LSTM networks built by enabling the bi-direction option. **Figure 3.2**, **Figure 3.3 and Figure 3.4** show the architecture of the LSTM classifier, together with the number of parameters within each layer.

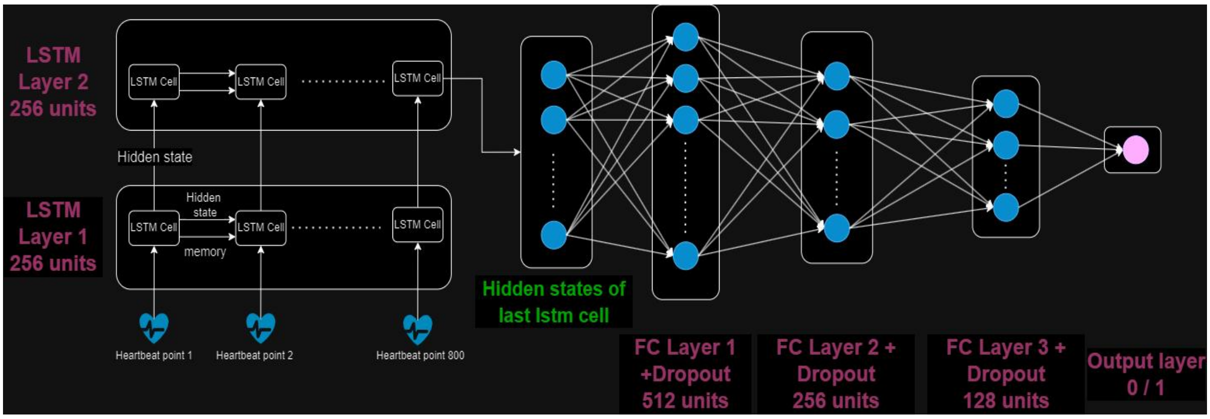
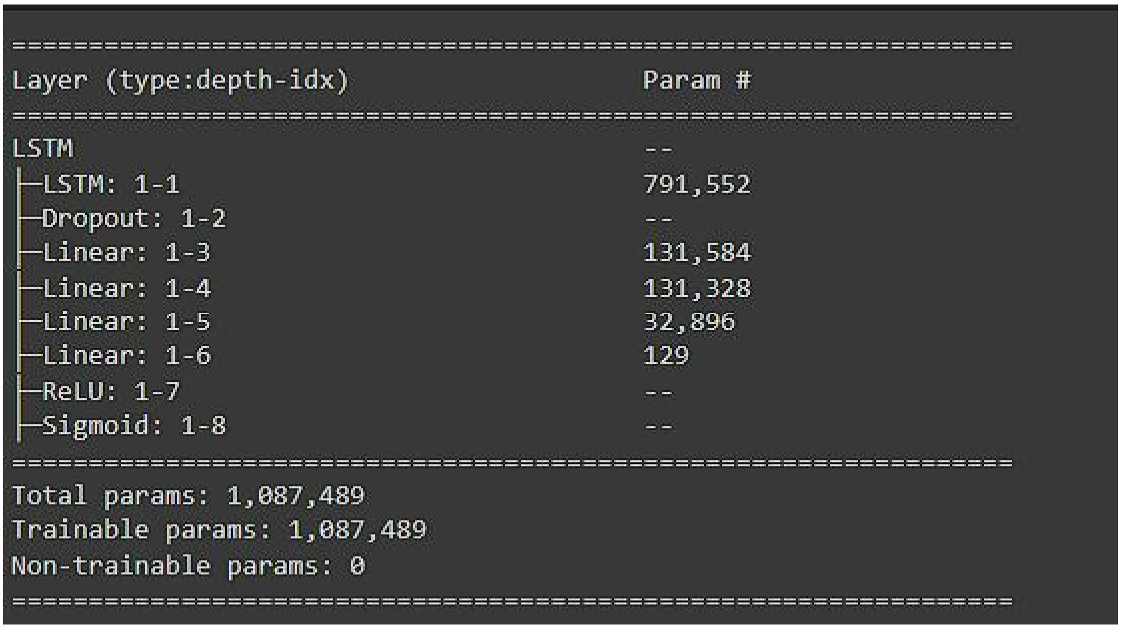
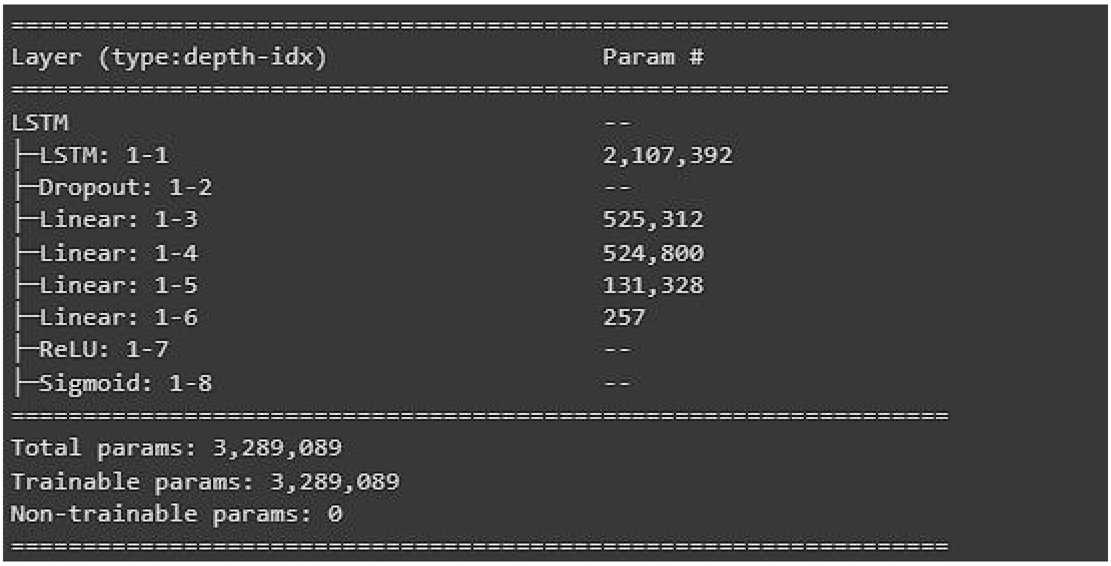


Figure 3.2 Network architecture of LSTM classifier



```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
LSTM                                     --
├─LSTM: 1-1                              791,552
├─Dropout: 1-2                           --
├─Linear: 1-3                            131,584
├─Linear: 1-4                            131,328
├─Linear: 1-5                            32,896
├─Linear: 1-6                            129
├─ReLU: 1-7                              --
├─Sigmoid: 1-8                           --
=================================================================
Total params: 1,087,489
Trainable params: 1,087,489
Non-trainable params: 0
=================================================================
```

Figure 3.3 Parameters of uni-directional LSTM classifier



```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
LSTM                                     --
├─LSTM: 1-1                              2,107,392
├─Dropout: 1-2                           --
├─Linear: 1-3                            525,312
├─Linear: 1-4                            524,800
├─Linear: 1-5                            131,328
├─Linear: 1-6                            257
├─ReLU: 1-7                              --
├─Sigmoid: 1-8                           --
=================================================================
Total params: 3,289,089
Trainable params: 3,289,089
Non-trainable params: 0
=================================================================
```

Figure 3.4 Parameters of bi-directional LSTM classifier

3.2.2 GRU Classifier

The input of the GRU [40] classifer should have the same shape as the LSTM's input. The network architecture is similar to the LSTM, which has 2 layers and 256 units in the GRU block and then the output is passed to four FC layers as shown in **Figure 3.5**. Identically, the bi-direction option is also available for GRU, so in total, two GRU models were trained for both data-split methods. **Figure 3.6** and **Figure 3.7** indicate the respective number of parameters for uni-directional and bi-directional GRU classifiers.

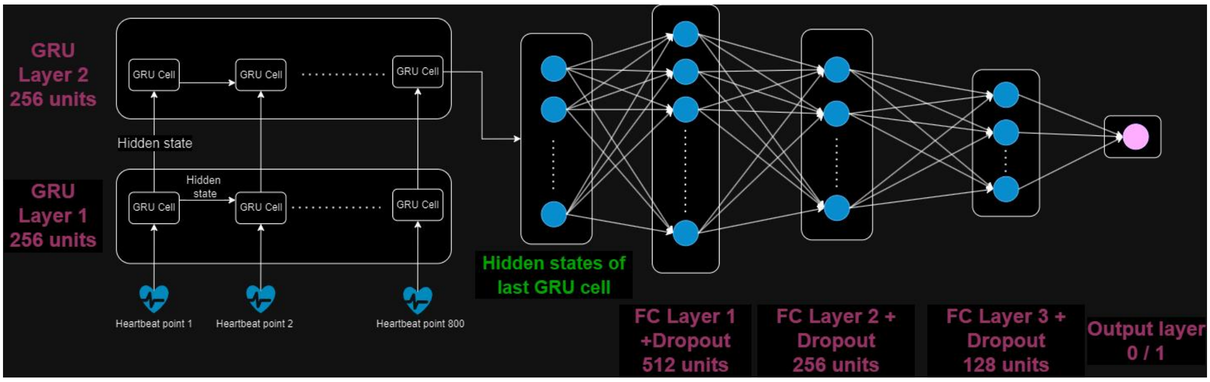
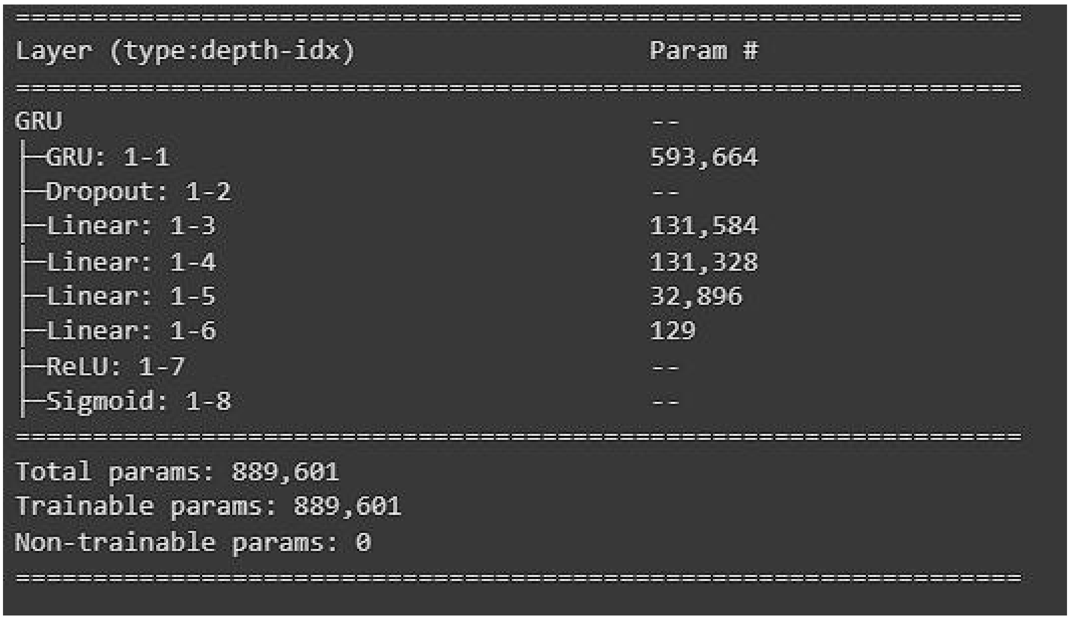
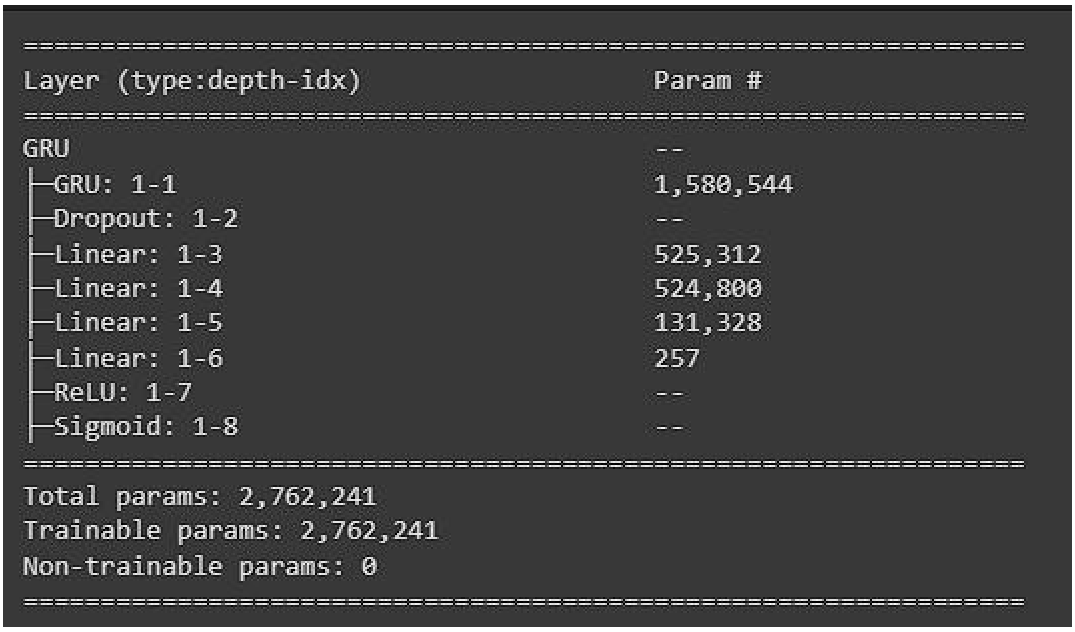


Figure 3.5 Network architecture of GRU classifier



```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
GRU                                      --
├─GRU: 1-1                               593,664
├─Dropout: 1-2                           --
├─Linear: 1-3                            131,584
├─Linear: 1-4                            131,328
├─Linear: 1-5                            32,896
├─Linear: 1-6                            129
├─ReLU: 1-7                              --
├─Sigmoid: 1-8                           --
=================================================================
Total params: 889,601
Trainable params: 889,601
Non-trainable params: 0
=================================================================
```

Figure 3.6 Parameters of uni-directional GRU classifier



```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
GRU                                      --
├─GRU: 1-1                               1,580,544
├─Dropout: 1-2                           --
├─Linear: 1-3                            525,312
├─Linear: 1-4                            524,800
├─Linear: 1-5                            131,328
├─Linear: 1-6                            257
├─ReLU: 1-7                              --
├─Sigmoid: 1-8                           --
=================================================================
Total params: 2,762,241
Trainable params: 2,762,241
Non-trainable params: 0
=================================================================
```

Figure 3.7 Parameters of bi-directional GRU classifier

3.2.3 1D-CNN Classifier

The input of the 1D-CNN [41] classifier was different from GRU and LSTM classifiers, it had a shape of (Batch Size, Dimensionality, Sequence Length). From **Figure 3.8**, it was easy to notice that the convolutional block consisting of 4 convolutional layers was built with a max pooling layer after each convolutional layer. Relatively, the number of parameters of 1D-CNN was significantly lower than LSTM and GRU as shown in **Figure 3.9**.

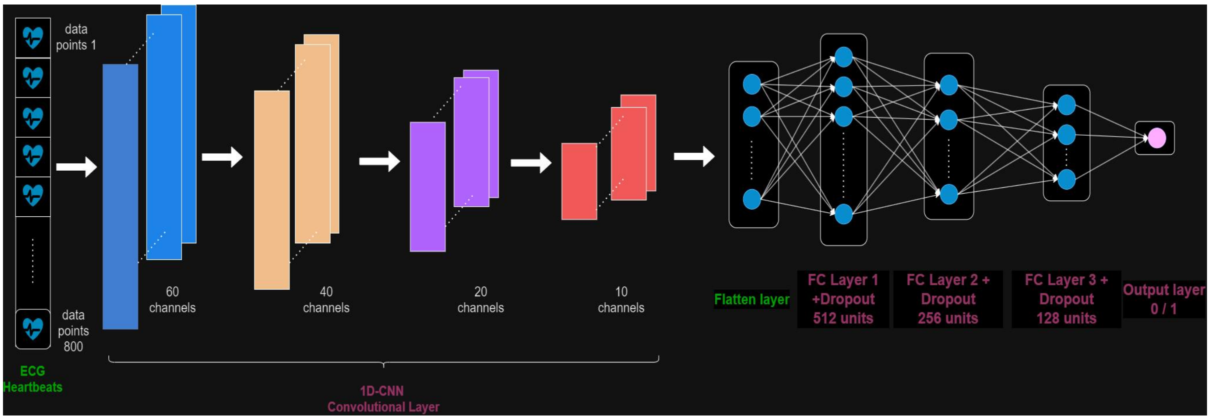
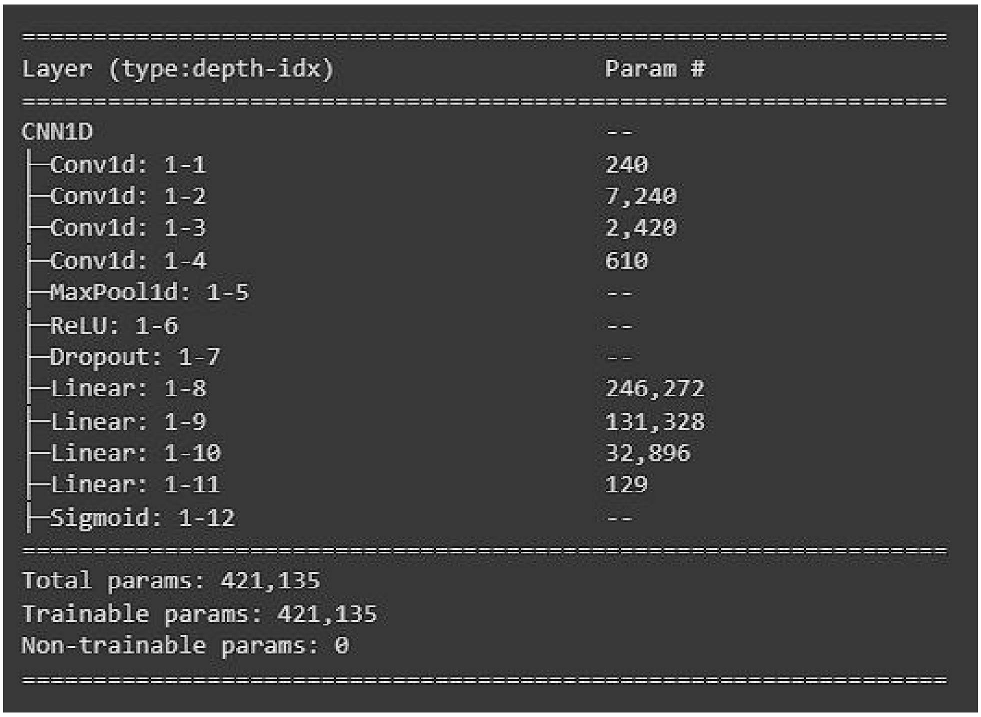


Figure 3.8 Network architecture of 1D-CNN classifier



```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
CNN1D                                    --
├─Conv1d: 1-1                            240
├─Conv1d: 1-2                            7,240
├─Conv1d: 1-3                            2,420
├─Conv1d: 1-4                            610
├─MaxPool1d: 1-5                         --
├─ReLU: 1-6                              --
├─Dropout: 1-7                           --
├─Linear: 1-8                            246,272
├─Linear: 1-9                            131,328
├─Linear: 1-10                           32,896
├─Linear: 1-11                           129
├─Sigmoid: 1-12                          --
=================================================================
Total params: 421,135
Trainable params: 421,135
Non-trainable params: 0
=================================================================
```

Figure 3.9 Parameters of 1D-CNN classifier

3.2.4 Transformer Classifier

Due to the nature of transformer architecture, the transformer is able to carry the multi-head attention [42] without considering the ordering of the data points. Ordering of data points is important to the ECG signal, therefore the input heartbeat sequence had to undergo a positional embedding to maintain its ordering. Next, the heartbeats with positions entered an encoder network comprising 6 encoder layers and for each of the layers, multi-head attention, batch normalization and feedforward network will produce learned information from the entire sequence. After that, the output of the encoder block would fit into the FC network.

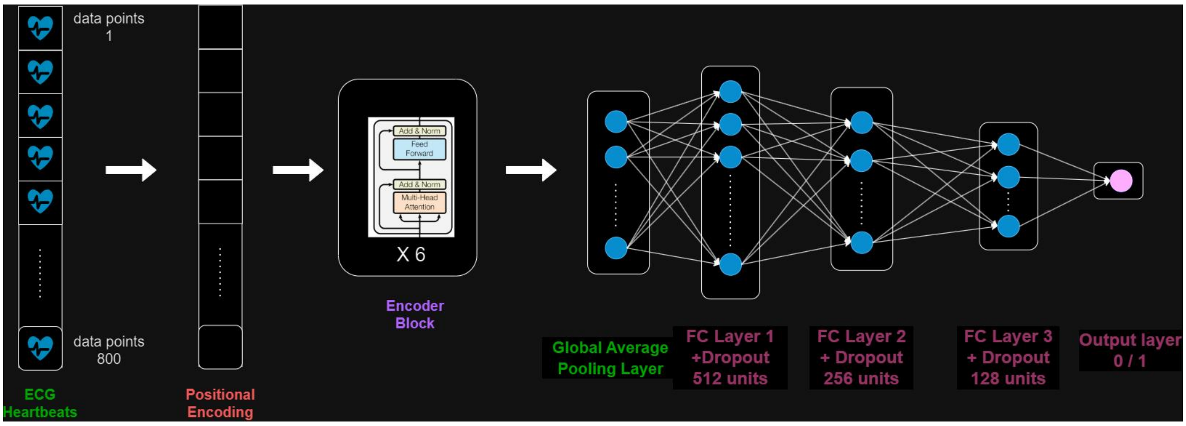
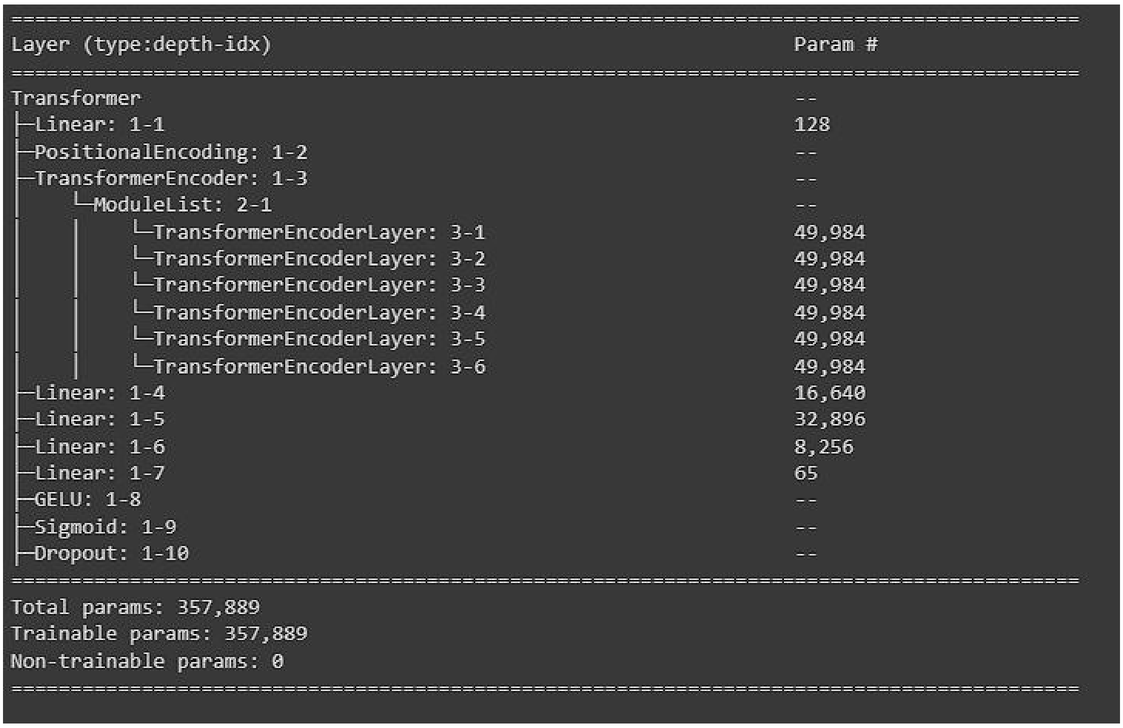


Figure 3.10 Network architecture of Transformer classifier

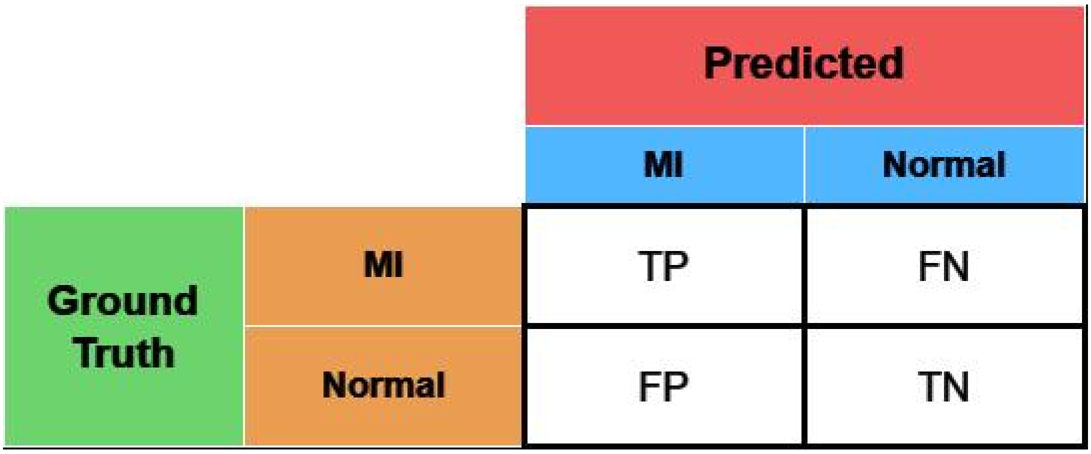


```
==========================================================================================
Layer (type:depth-idx)                                                 Param #
==========================================================================================
Transformer                                                            --
├─Linear: 1-1                                                          128
├─PositionalEncoding: 1-2                                              --
├─TransformerEncoder: 1-3                                              --
│    └─ModuleList: 2-1                                                 --
│    │    └─TransformerEncoderLayer: 3-1                               49,984
│    │    └─TransformerEncoderLayer: 3-2                               49,984
│    │    └─TransformerEncoderLayer: 3-3                               49,984
│    │    └─TransformerEncoderLayer: 3-4                               49,984
│    │    └─TransformerEncoderLayer: 3-5                               49,984
│    │    └─TransformerEncoderLayer: 3-6                               49,984
├─Linear: 1-4                                                          16,640
├─Linear: 1-5                                                          32,896
├─Linear: 1-6                                                          8,256
├─Linear: 1-7                                                          65
├─GELU: 1-8                                                            --
├─Sigmoid: 1-9                                                         --
├─Dropout: 1-10                                                        --
==========================================================================================
Total params: 357,889
Trainable params: 357,889
Non-trainable params: 0
==========================================================================================
```

Figure 3.11 Parameters of Transformer classifier

3.3 Performance Metrics

Performance metrics or standards must be set up to quantify the performance of the model/classifier. Four metrics were chosen for this research as they are commonly used in other studies [22] therefore the comparison could be made in a more intuitive way. A confusion matrix is a good way to measure the performance of the classifier as shown in **Figure 3.12**. In this research, the performance metrics included 4 types of measurements: accuracy [20], sensitivity [20], precision [20] and F1 score [20].



		Predicted	
		MI	Normal
Ground Truth	MI	TP	FN
	Normal	FP	TN

Figure 3.12 Example of confusion matrix where TP is a true positive, TN is a true negative, FP is a false positive, and FN is a false negative

3.3.1 Accuracy

The equation of accuracy is as shown in equation 1.

$$(ACC) = (TP+TN) / (TP+TN+FP+ FN) \tag{1}$$

Undoubtedly, a good model/classifier should achieve high accuracy as this is the most general measurement and the easiest concept for people to be confident in its performance and definitely reliability. Nonetheless, solely depending on accuracy might end up in some misleading conclusions when the dataset is imbalanced [20], especially in a binary classification task. If one of the categories (For example, too many MI patients) got too high a ratio to the other, it is still easy to produce a good accuracy result even though the classifier classifies all the ECGs to be MI.

3.3.2 Sensitivity

The equation of sensitivity is shown in Equation 2.

$$(SEN) = (TP) / (TP+FN)$$

$$(2)$$

Where Sensitivity (SEN) = true positive rate (TPR) = recall (RC)

On top of accuracy, sensitivity is also an important tool especially the application in the medical field. This is, according to the formula, a low sensitivity means that the model will easily classify an MI ECG as a normal ECG, therefore it is dangerous to delay the presentation time of the patients.

3.3.3 Precision

The equation of precision is shown in Equation 3.

$$(PRE) = (TP) / (TP+FP) \tag{3}$$

Where Formula of Precision (PRE) = positive predictive value (PPV)

It is also a waste of medical attention and resources if the model always classifies a normal ECG as an MI ECG. Hence, a good model should also achieve high precision to avoid this error in real-world applications.

3.3.4 F1 score

The equation of accuracy is shown in Equation 4.

$$(ACC) = (2TP) / (2TP+FP+ FN) \tag{4}$$

F1 score is a good way to eliminate the bad effect caused by accuracy in an imbalanced dataset [22]. It takes sensitivity and precision into consideration at the same time.

Chapter 4

Implementation & Experimental Results

4.1 Hardware & Software Setup

4.1.1 Hardware

The hardware involved in this project is only a laptop. A computer was issued to carry out implementation/coding for data collection, data cleaning, and data transformation as well as designing network architecture and helper functions. Specification of the laptop is shown in **Table 4.1**, it was more than enough to complete all tasks mentioned above.

Table 4.1 Specifications of laptop

Description	Specifications
Model	Asus TUF Gaming A15
Processor	AMD Ryzen 5 4600H
Operating System	Windows 11
Graphic	NVIDIA GeForce GTX 1650 Ti
Memory	16GB DDR4 RAM
Storage	512GB SATA HDD

4.1.2 Software

Due to the limitation of the hardware's computing power, the training, validation and testing processes were completed on online software.

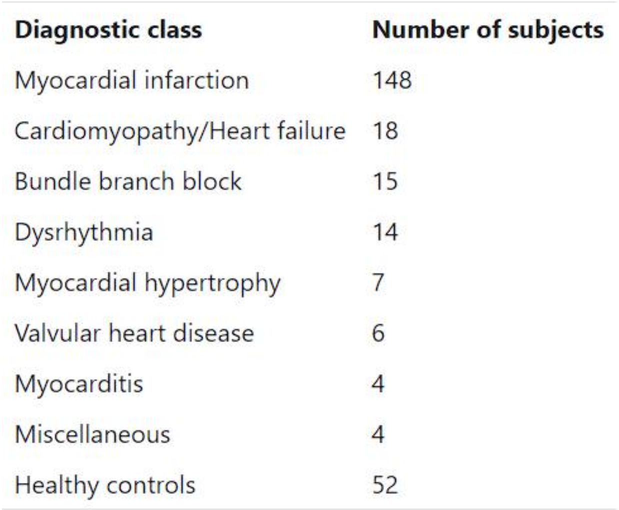
1. Deep Learning Framework: Pytorch
2. IDE: Google Colab Pro
3. Available CPU: Unspecified model with 51.0 GB RAM
4. Available TPU/GPUs: TPUv2, A100, T4, L4 with GPU RAM from 15.0 GB to above 50.0 GB
5. Python waveform-database (WFDB) package

4.2 Data Collection

The data source of this project is the PTB Diagnostic ECG Database [24]. It is an open-source database published in 2004 to obtain free ECG signals and at the same time, it makes the signal anonymous in order to protect the privacy of patients. A collection of 549 high-quality ECG signals, most of them are encapsulated with their respective clinical summary together in the organized form. There are ECGs from 268 subjects available for the research and this project would only need the ECGs collected from MI patients and also the normal subjects. **Figure 4.1** indicates that there are a total of 148 MI patients and 52 normal subjects for this project to utilize their signals. **Figure 4.2** shows all the technical descriptions of the recorder used to obtain the signals. The sampling rate was 1000Hz frequency and the noise was included during the recording.

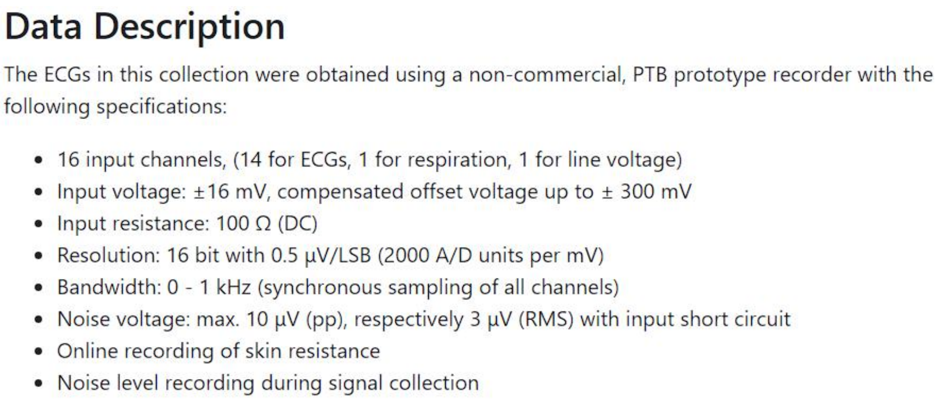
The ECGs are well-organized in the database as shown in **Figure 4.3** and **Figure 4.4** by using a patient-split. There are 3 types of files inside each patient folder: header, data, and xyz. To extract the raw signal data, the header and data files play an important role in acting as the parameters in WFDB's 'rdrecord' function.

URL: https://physionet.org/content/ptbdb/1.0.0/



Diagnostic class	Number of subjects
Myocardial infarction	148
Cardiomyopathy/Heart failure	18
Bundle branch block	15
Dysrhythmia	14
Myocardial hypertrophy	7
Valvular heart disease	6
Myocarditis	4
Miscellaneous	4
Healthy controls	52

Figure 4.1 Number of subjects in each diagnostic class

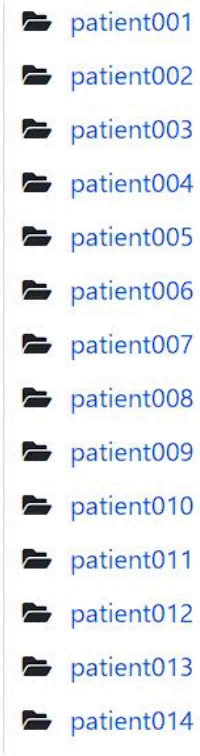


Data Description

The ECGs in this collection were obtained using a non-commercial, PTB prototype recorder with the following specifications:

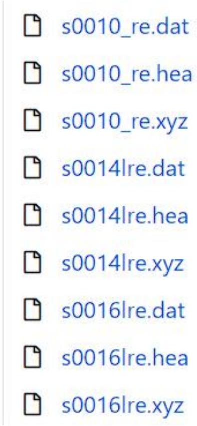
- 16 input channels, (14 for ECGs, 1 for respiration, 1 for line voltage)
- Input voltage: ±16 mV, compensated offset voltage up to ± 300 mV
- Input resistance: 100 Ω (DC)
- Resolution: 16 bit with 0.5 µV/LSB (2000 A/D units per mV)
- Bandwidth: 0 - 1 kHz (synchronous sampling of all channels)
- Noise voltage: max. 10 µV (pp), respectively 3 µV (RMS) with input short circuit
- Online recording of skin resistance
- Noise level recording during signal collection

Figure 4.2 Technical description of the signal recorder



patient001
patient002
patient003
patient004
patient005
patient006
patient007
patient008
patient009
patient010
patient011
patient012
patient013
patient014

Figure 4.3 Organization of patient data in the PTB database



s0010_re.dat
s0010_re.hea
s0010_re.xyz
s0014lre.dat
s0014lre.hea
s0014lre.xyz
s0016lre.dat
s0016lre.hea
s0016lre.xyz

Figure 4.4 Contents of patient folder

4.3 Data Pre-Processing

After we extracted the raw ECG signals from the data file by using the WFDB function, we had to perform pre-processing on the signals as they were inconsistent in sequence length and unavoidably suffered from noise and baseline wandering. A common solution [20] was performing a filtering to smoothen the signal and a segmentation to ensure a consistent length of signals. Moreover, to train the models, we split the original signal collection into the training, validation, and testing sets. To ensure that our testing result is not naïve and not a coincidence, stratified 10-fold cross-validation [43] such as **Figure 4.5** was applied to produce 10 training sets and 10 validation sets, but only one testing set to monitor the validation results. From **Figure 4.1**, it is obvious that the number of healthy subjects is almost three times less than the number of MI patients, thus we also performed data augmentation in a conservative way which is time-shifting [26] as shown in **Figure 4.6** to reduce the data imbalance.

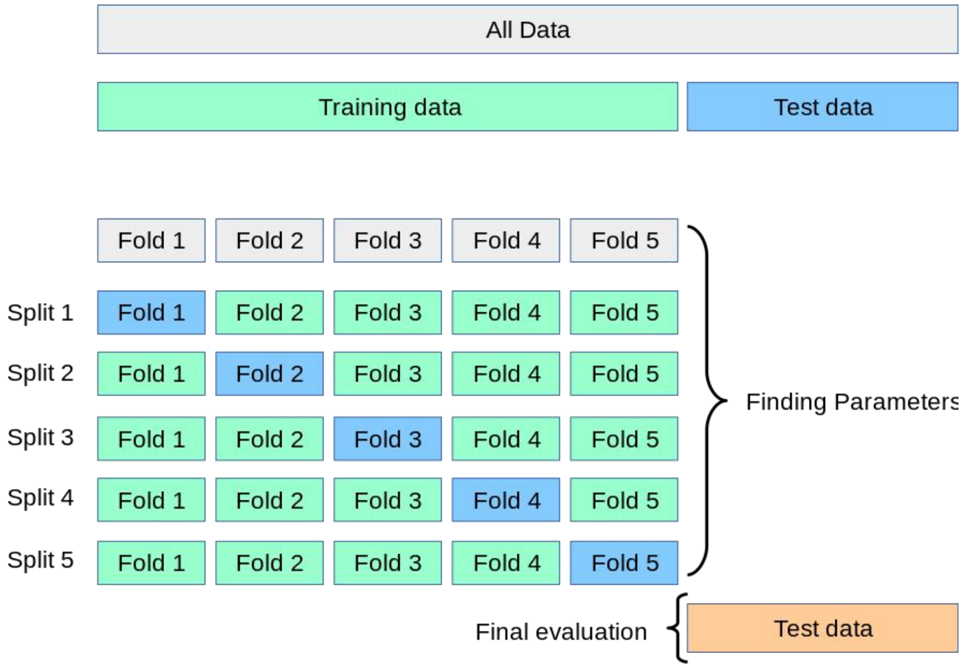


Figure 4.5 Example of 10-fold cross-validation [44]

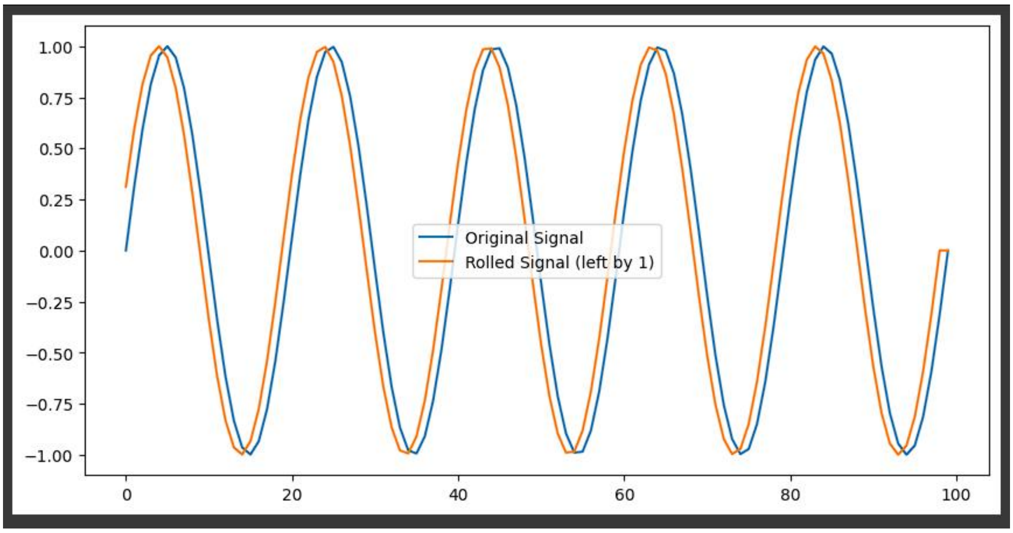
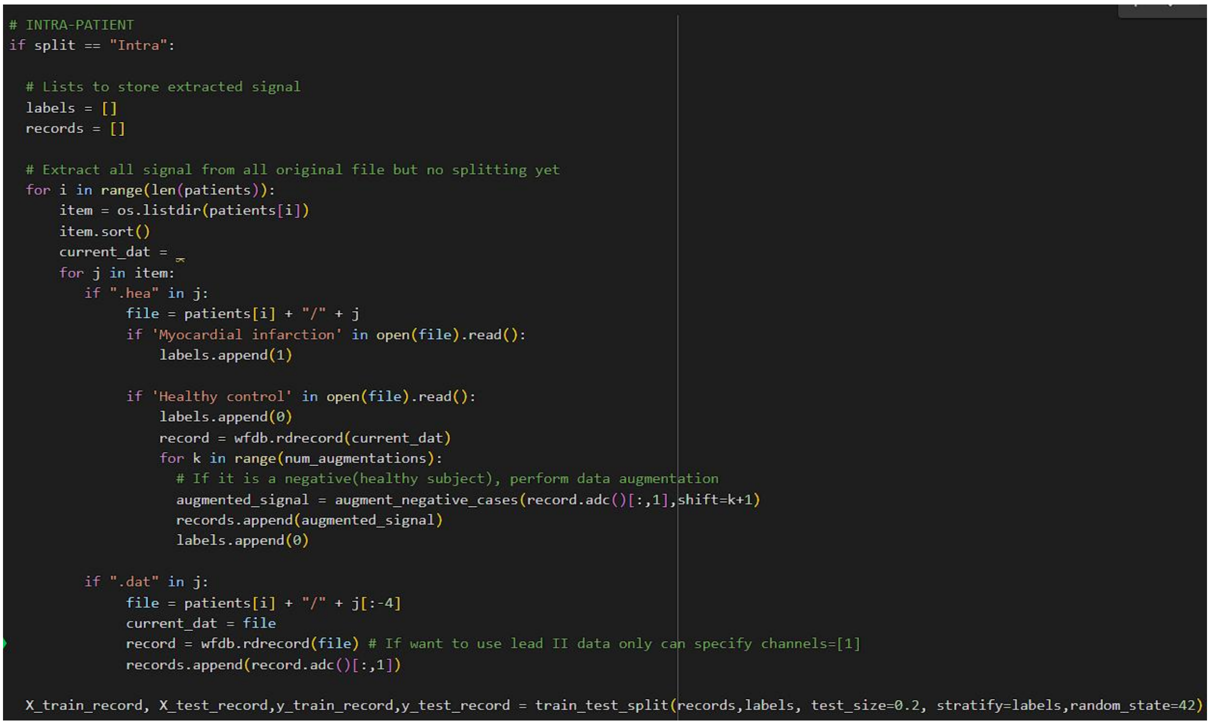


Figure 4.6 Time-shift Data Augmentation

4.3.1 Intra-Patient Splitting

Intra-patient splitting is the so-called beat-split, the splitting operation would not be executed on the patient level but only on the heartbeat level. In **Figure 4.7**, the WFDB function was executed first to extract the signals from the data file in each patient folder. Then the Scikit-learn's function, train_test_split was performed to randomly assign the signals either to the training set or the testing set. All the signals in both sets would undergo the following pre-processing steps before they are arranged into 10 folds.

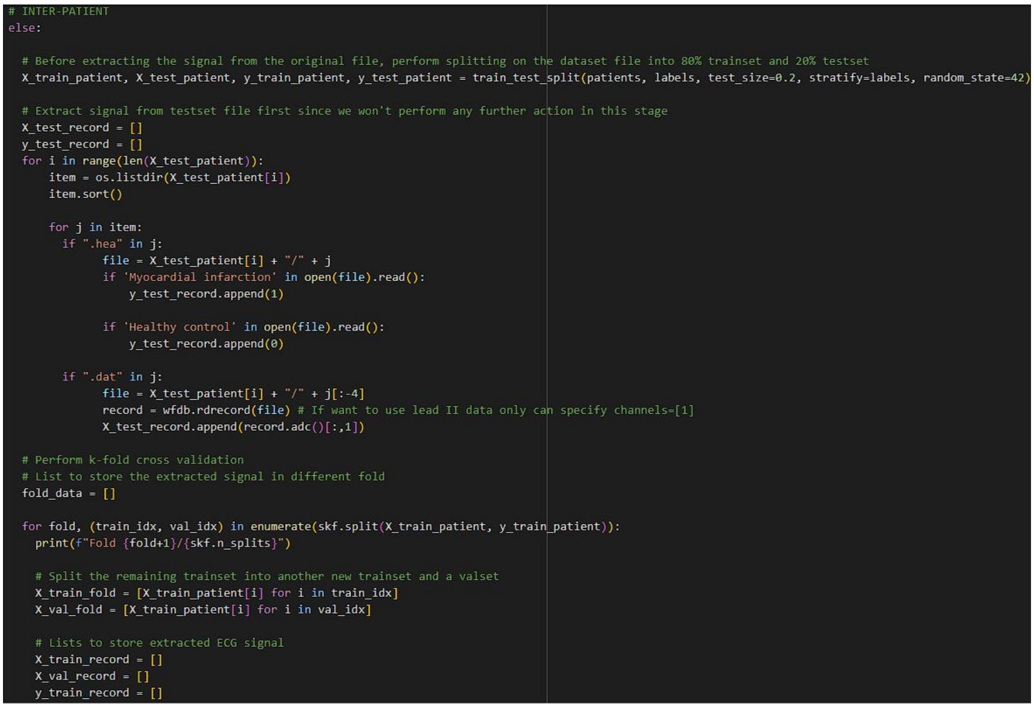


```
# INTRA-PATIENT
if split == "Intra":

  # Lists to store extracted signal
  labels = []
  records = []

  # Extract all signal from all original file but no splitting yet
  for i in range(len(patients)):
      item = os.listdir(patients[i])
      item.sort()
      current_dat =
      for j in item:
         if ".hea" in j:
              file = patients[i] + "/" + j
              if 'Myocardial infarction' in open(file).read():
                  labels.append(1)

              if 'Healthy control' in open(file).read():
                  labels.append(0)
                  record = wfdb.rdrecord(current_dat)
                  for k in range(num_augmentations):
                    # If it is a negative(healthy subject), perform data augmentation
                    augmented_signal = augment_negative_cases(record.adc()[:,1],shift=k+1)
                    records.append(augmented_signal)
                    labels.append(0)

         if ".dat" in j:
              file = patients[i] + "/" + j[:-4]
              current_dat = file
              record = wfdb.rdrecord(file) # If want to use lead II data only can specify channels=[1]
              records.append(record.adc()[:,1])

  X_train_record, X_test_record,y_train_record,y_test_record = train_test_split(records,labels, test_size=0.2, stratify=labels,random_state=42)
```

Figure 4.7 Intra-Patient Splitting

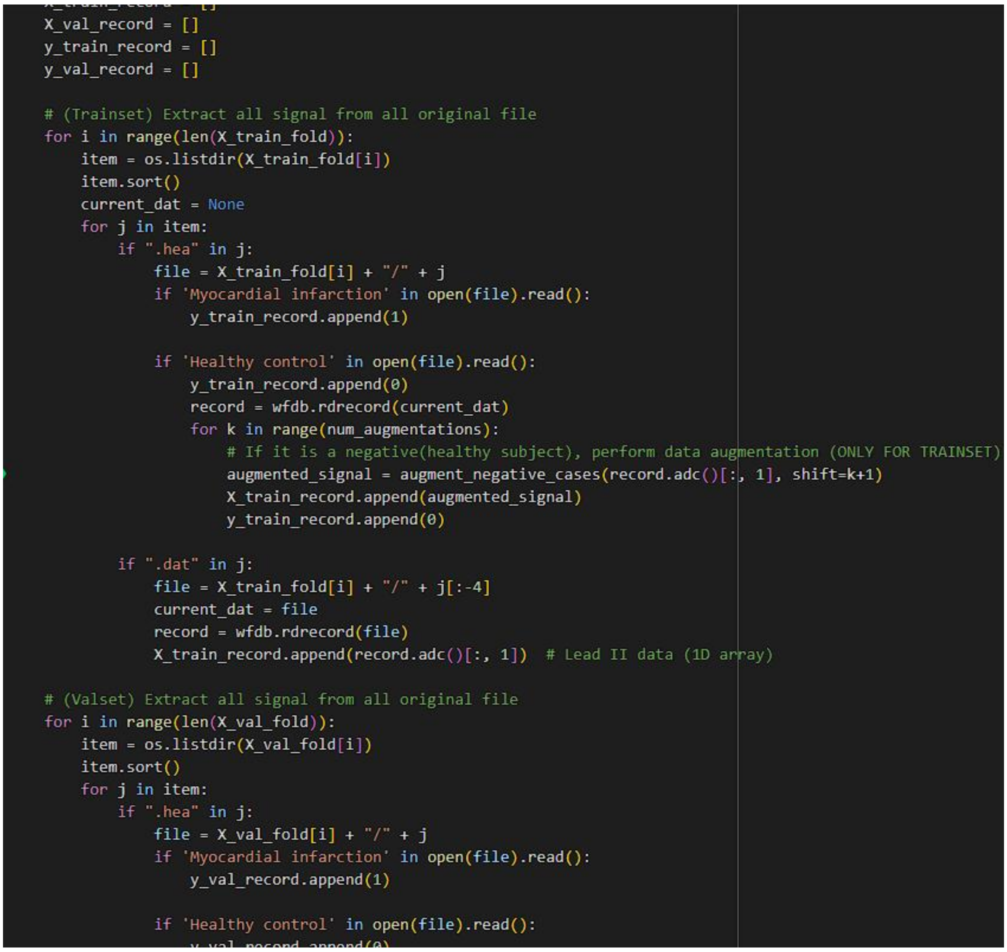
4.3.2 Inter-Patient Splitting

Instead, to ensure inter-patient splitting or patient split, the patient folders must be split into the training and testing set first. Then the folders in the training set must be split again into a new training set and validation set in each loop of a 10-fold cross-validation. This was because it can ensure that the heartbeats from patients produced from later pre-processing steps would only present in either the training set, validation set or testing set.



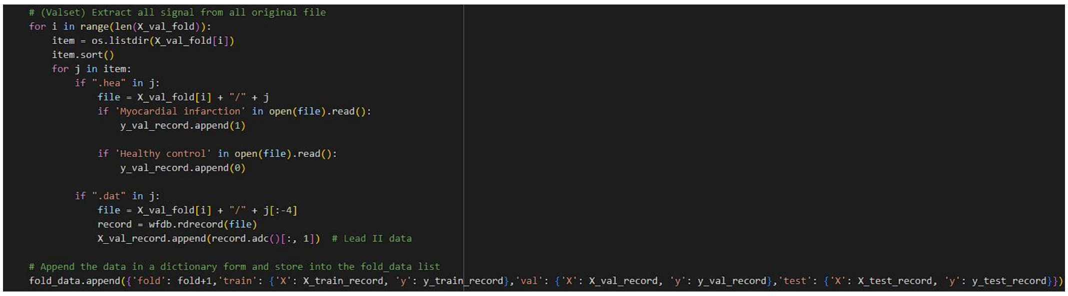
```
# INTER-PATIENT
else:

  # Before extracting the signal from the original file, perform splitting on the dataset file into 80% trainset and 20% testset
  X_train_patient, X_test_patient, y_train_patient, y_test_patient = train_test_split(patients, labels, test_size=0.2, stratify=labels, random_state=42)

  # Extract signal from testset file first since we won't perform any further action in this stage
  X_test_record = []
  y_test_record = []
  for i in range(len(X_test_patient)):
      item = os.listdir(X_test_patient[i])
      item.sort()

      for j in item:
        if ".hea" in j:
              file = X_test_patient[i] + "/" + j
              if 'Myocardial infarction' in open(file).read():
                  y_test_record.append(1)

              if 'Healthy control' in open(file).read():
                  y_test_record.append(0)

        if ".dat" in j:
              file = X_test_patient[i] + "/" + j[:-4]
              record = wfdb.rdrecord(file) # If want to use lead II data only can specify channels=[1]
              X_test_record.append(record.adc()[:,1])

  # Perform k-fold cross validation
  # List to store the extracted signal in different fold
  fold_data = []

  for fold, (train_idx, val_idx) in enumerate(skf.split(X_train_patient, y_train_patient)):
    print(f"Fold {fold+1}/{skf.n_splits}")

    # Split the remaining trainset into another new trainset and a valset
    X_train_fold = [X_train_patient[i] for i in train_idx]
    X_val_fold = [X_train_patient[i] for i in val_idx]

    # Lists to store extracted ECG signal
    X_train_record = []
    X_val_record = []
    y_train_record = []
```

Figure 4.8 Inter-Patient Splitting Part 1



```
    X_val_record = []
    y_train_record = []
    y_val_record = []

    # (Trainset) Extract all signal from all original file
    for i in range(len(X_train_fold)):
        item = os.listdir(X_train_fold[i])
        item.sort()
        current_dat = None
        for j in item:
            if ".hea" in j:
                file = X_train_fold[i] + "/" + j
                if 'Myocardial infarction' in open(file).read():
                    y_train_record.append(1)

                if 'Healthy control' in open(file).read():
                    y_train_record.append(0)
                    record = wfdb.rdrecord(current_dat)
                    for k in range(num_augmentations):
                        # If it is a negative(healthy subject), perform data augmentation (ONLY FOR TRAINSET)
                        augmented_signal = augment_negative_cases(record.adc()[:, 1], shift=k+1)
                        X_train_record.append(augmented_signal)
                        y_train_record.append(0)

            if ".dat" in j:
                file = X_train_fold[i] + "/" + j[:-4]
                current_dat = file
                record = wfdb.rdrecord(file)
                X_train_record.append(record.adc()[:, 1])  # Lead II data (1D array)

    # (Valset) Extract all signal from all original file
    for i in range(len(X_val_fold)):
        item = os.listdir(X_val_fold[i])
        item.sort()
        for j in item:
            if ".hea" in j:
                file = X_val_fold[i] + "/" + j
                if 'Myocardial infarction' in open(file).read():
                    y_val_record.append(1)

                if 'Healthy control' in open(file).read():
```

Figure 4.9 Inter-Patient Splitting Part 2

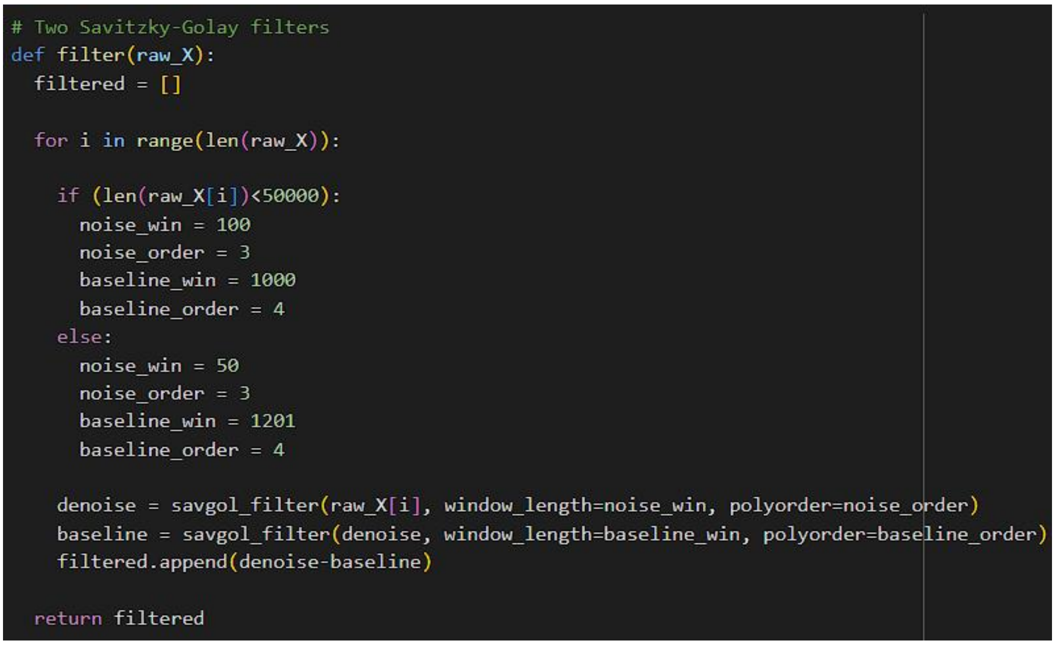


```
    # (Valset) Extract all signal from all original file
    for i in range(len(X_val_fold)):
        item = os.listdir(X_val_fold[i])
        item.sort()
        for j in item:
            if ".hea" in j:
                file = X_val_fold[i] + "/" + j
                if 'Myocardial infarction' in open(file).read():
                    y_val_record.append(1)

                if 'Healthy control' in open(file).read():
                    y_val_record.append(0)

            if ".dat" in j:
                file = X_val_fold[i] + "/" + j[:-4]
                record = wfdb.rdrecord(file)
                X_val_record.append(record.adc()[:, 1])  # Lead II data

    # Append the data in a dictionary form and store into the fold_data list
    fold_data.append({'fold': fold+1,'train': {'X': X_train_record, 'y': y_train_record},'val': {'X': X_val_record, 'y': y_val_record},'test': {'X': X_test_record, 'y': y_test_record}})
```

Figure 4.10 Inter-Patient Splitting Part 3

4.3.3 Denoise & Baseline Wandering Removal

There were in total of two Savitzky-Golay filters [36] applied in this project for the purpose of denoise and baseline wandering removal operations. Through a series of manual experiments, a good enough parameter setting was captured for both filters. The implementation of the filtering is shown in Figure 4.11 and an example of a successful filtering is shown in Figure 4.12.



```python
# Two Savitzky-Golay filters
def filter(raw_X):
  filtered = []

  for i in range(len(raw_X)):

    if (len(raw_X[i])<50000):
      noise_win = 100
      noise_order = 3
      baseline_win = 1000
      baseline_order = 4
    else:
      noise_win = 50
      noise_order = 3
      baseline_win = 1201
      baseline_order = 4

    denoise = savgol_filter(raw_X[i], window_length=noise_win, polyorder=noise_order)
    baseline = savgol_filter(denoise, window_length=baseline_win, polyorder=baseline_order)
    filtered.append(denoise-baseline)

  return filtered
```

Figure 4.11 Implementation of SG filter

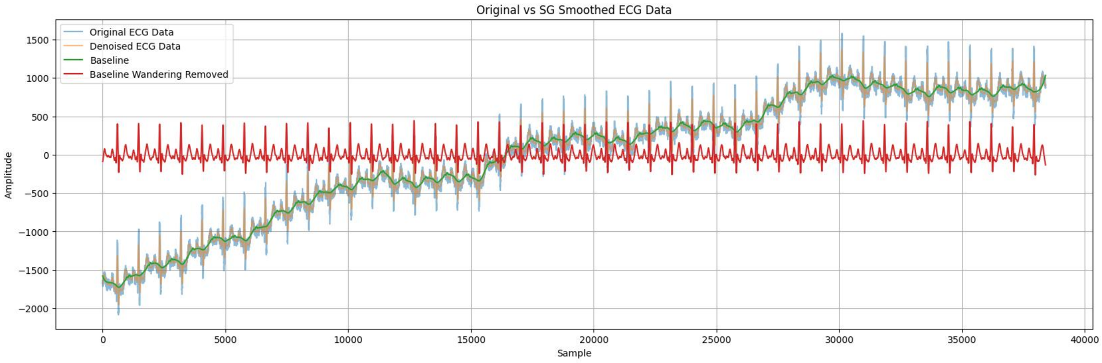
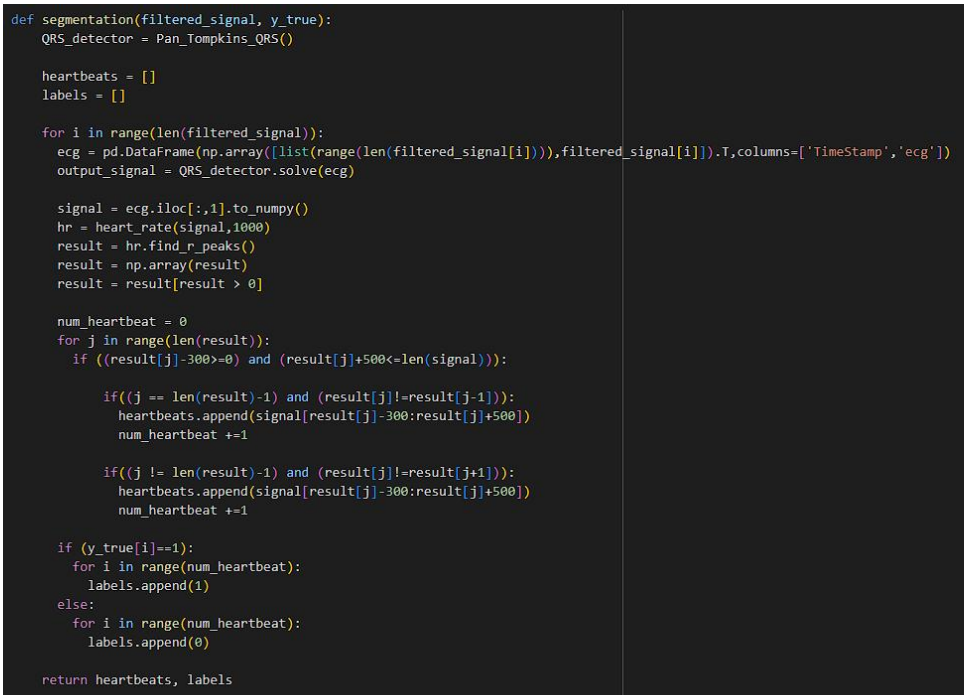


Figure 4.12 Result of successful filtering, the red line represents the filtered signal, while the green line represents the baseline

4.3.4 R-Peak Detection & Segmentation

Pan-Tompkins algorithm [37] is able to locate the R peak as illustrated in **Figure 4.14** accurately by searching for the QRS region for each heartbeat cycle. After R peak locations are captured, then index slicing (segmentation) of the NumPy array is performed to get separate distinct heartbeats with a consistent length of the sequence of 800 as shown in **Figure 4.15**. In FYP1, the total number of heartbeats is 52096 (8953 normal and 42,867 MI) in the training set as well as 13251 (1910 normal and 11341 MI) in the testing set. Through the deployment of a data augmentation technique, the data imbalance issue was resolved as shown in **Figure 4.16**. Nevertheless, noted that in **Figure 4.17**, clearly shows that each set has a different number of MI ECGs and normal ECGs from fold to fold due to the earlier train_test_split operation and the data augmentation only acted on the signals in the training set.



```
def segmentation(filtered_signal, y_true):
    QRS_detector = Pan_Tompkins_QRS()

    heartbeats = []
    labels = []

    for i in range(len(filtered_signal)):
      ecg = pd.DataFrame(np.array([list(range(len(filtered_signal[i]))),filtered_signal[i]]).T,columns=['TimeStamp','ecg'])
      output_signal = QRS_detector.solve(ecg)

      signal = ecg.iloc[:,1].to_numpy()
      hr = heart_rate(signal,1000)
      result = hr.find_r_peaks()
      result = np.array(result)
      result = result[result > 0]

      num_heartbeat = 0
      for j in range(len(result)):
        if ((result[j]-300>=0) and (result[j]+500<=len(signal))):

            if((j == len(result)-1) and (result[j]!=result[j-1])):
              heartbeats.append(signal[result[j]-300:result[j]+500])
              num_heartbeat +=1

            if((j != len(result)-1) and (result[j]!=result[j+1])):
              heartbeats.append(signal[result[j]-300:result[j]+500])
              num_heartbeat +=1

      if (y_true[i]==1):
        for i in range(num_heartbeat):
          labels.append(1)
      else:
        for i in range(num_heartbeat):
          labels.append(0)

    return heartbeats, labels
```

Figure 4.13 Implementation of R-peak detection and segmentation

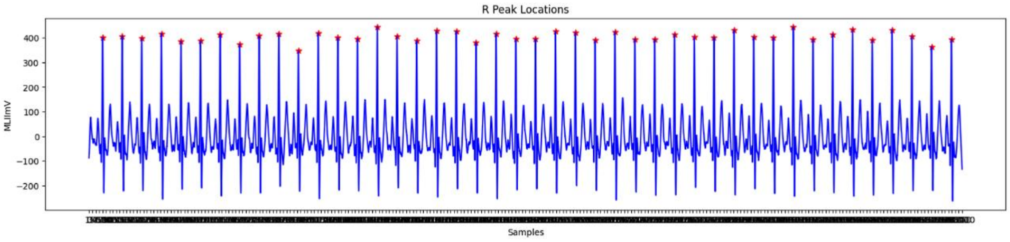


Figure 4.14 R-peaks were detected

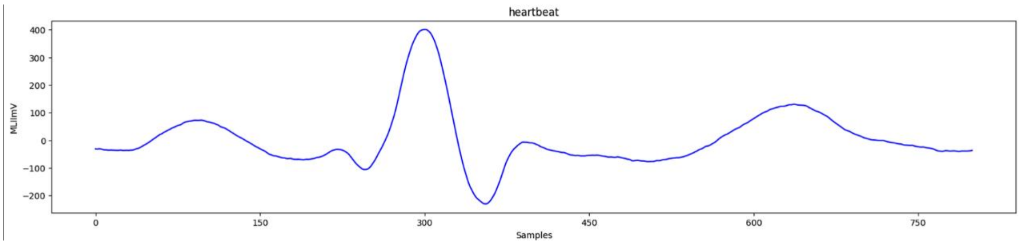
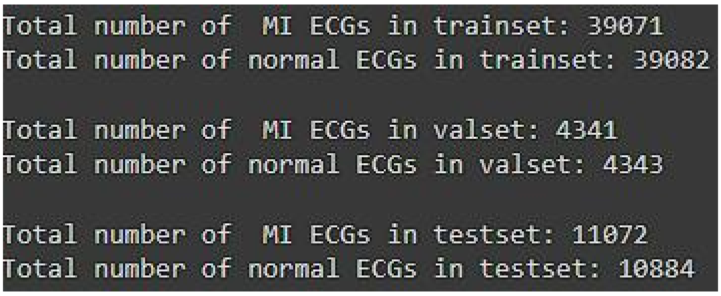
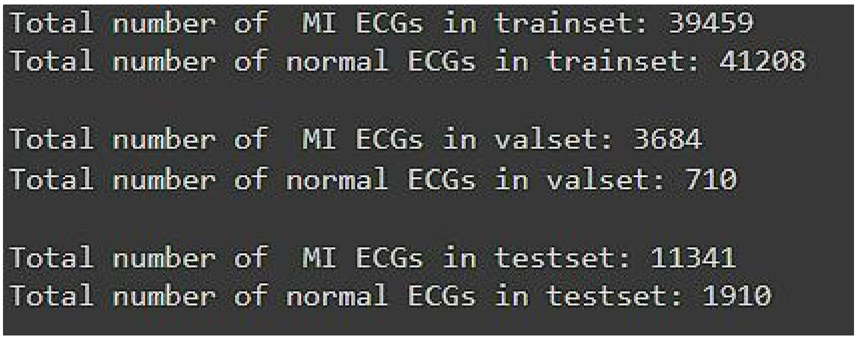


Figure 4.15 Segmentation from 300 points before R-peak and 500 points after R-peak



```
Total number of  MI ECGs in trainset: 39071
Total number of normal ECGs in trainset: 39082

Total number of  MI ECGs in valset: 4341
Total number of normal ECGs in valset: 4343

Total number of  MI ECGs in testset: 11072
Total number of normal ECGs in testset: 10884
```

Figure 4.16 Number of MI ECGs and normal ECGs in each set of each fold (Intra-Patient)

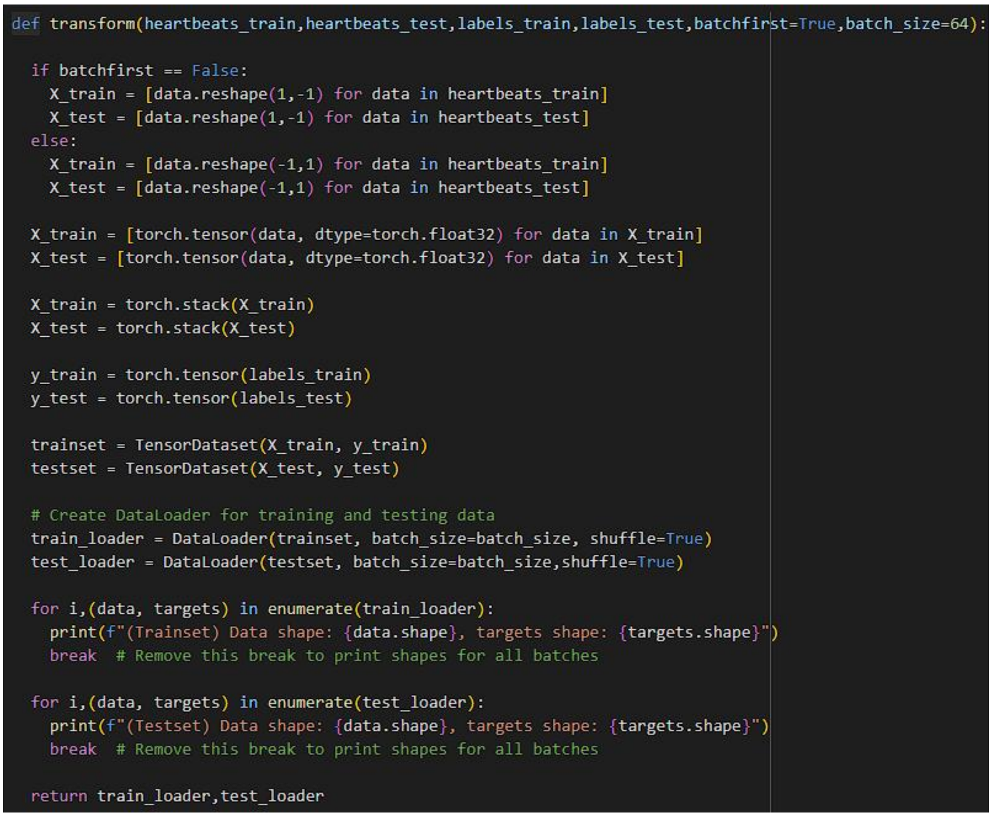


```
Total number of  MI ECGs in trainset: 39459
Total number of normal ECGs in trainset: 41208

Total number of  MI ECGs in valset: 3684
Total number of normal ECGs in valset: 710

Total number of  MI ECGs in testset: 11341
Total number of normal ECGs in testset: 1910
```

Figure 4.17 Number of MI ECGs and normal ECGs in each set (Inter-Patient)

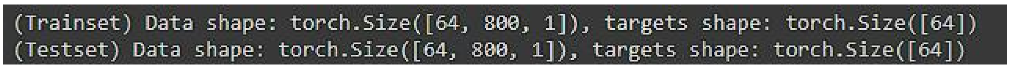
4.3.5 Transformation

The final step of the pre-processing stage was to transform the processed heartbeats in each set of each fold into data loader format for the ease of training, validation and testing. The ‘batchfirst’ parameter was prepared for the 1D-CNN model to change the heartbeats into a desirable format. As PyTorch was the deep learning framework deployed in the project, the heartbeats that were initially in NumPy type must be transformed to PyTorch’s tensor type. Then stack operation was executed to transform the 2D tensors into 3D tensors. The DataLoader function is capable of creating mini-batches of heartbeat data to allow faster training and frequent updates on the parameters as well as more efficient memory management compared to training the model using the whole dataset at once. The implementation is shown in **Figure 4.18** and **Figure 4.19** shows the example of a batch of heartbeats in the dataloader.



```
def transform(heartbeats_train,heartbeats_test,labels_train,labels_test,batchfirst=True,batch_size=64):

  if batchfirst == False:
    X_train = [data.reshape(1,-1) for data in heartbeats_train]
    X_test = [data.reshape(1,-1) for data in heartbeats_test]
  else:
    X_train = [data.reshape(-1,1) for data in heartbeats_train]
    X_test = [data.reshape(-1,1) for data in heartbeats_test]

  X_train = [torch.tensor(data, dtype=torch.float32) for data in X_train]
  X_test = [torch.tensor(data, dtype=torch.float32) for data in X_test]

  X_train = torch.stack(X_train)
  X_test = torch.stack(X_test)

  y_train = torch.tensor(labels_train)
  y_test = torch.tensor(labels_test)

  trainset = TensorDataset(X_train, y_train)
  testset = TensorDataset(X_test, y_test)

  # Create DataLoader for training and testing data
  train_loader = DataLoader(trainset, batch_size=batch_size, shuffle=True)
  test_loader = DataLoader(testset, batch_size=batch_size,shuffle=True)

  for i,(data, targets) in enumerate(train_loader):
    print(f"(Trainset) Data shape: {data.shape}, targets shape: {targets.shape}")
    break  # Remove this break to print shapes for all batches

  for i,(data, targets) in enumerate(test_loader):
    print(f"(Testset) Data shape: {data.shape}, targets shape: {targets.shape}")
    break  # Remove this break to print shapes for all batches

  return train_loader,test_loader
```

Figure 4.18 Implementation of transformation

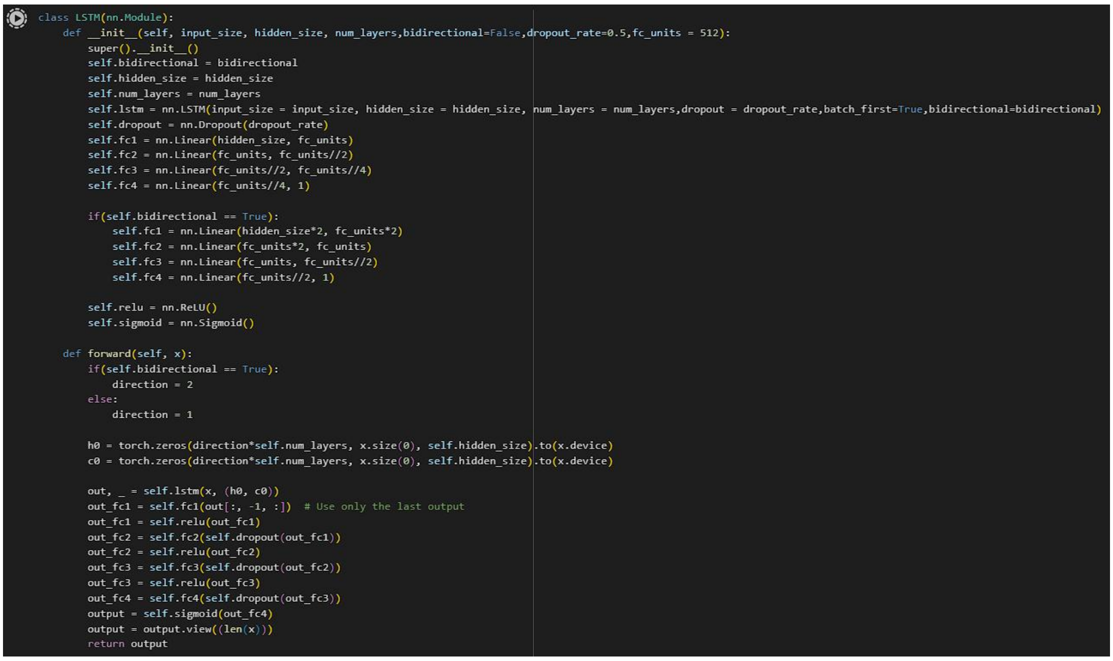


```
(Trainset) Data shape: torch.Size([64, 800, 1]), targets shape: torch.Size([64])
(Testset) Data shape: torch.Size([64, 800, 1]), targets shape: torch.Size([64])
```

Figure 4.19 The training and testing set is encapsulated in the data loader

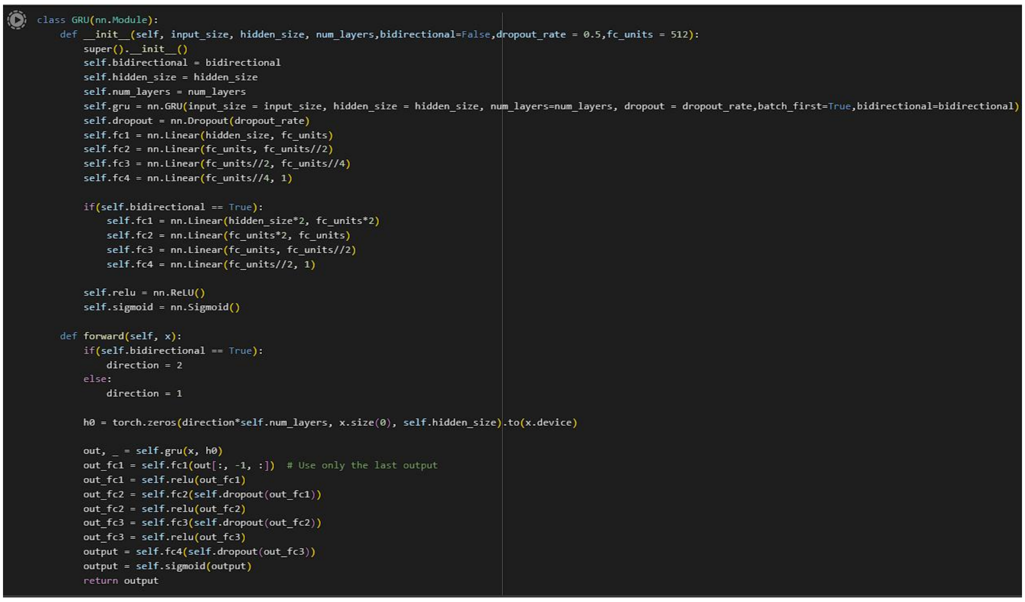
4.4 Network Architecture

The implementation of each network architecture is shown from **Figure 4.20** until **Figure 4.23**. According to the designs as shown in Chapter 3, the class of each type of classifier was built.



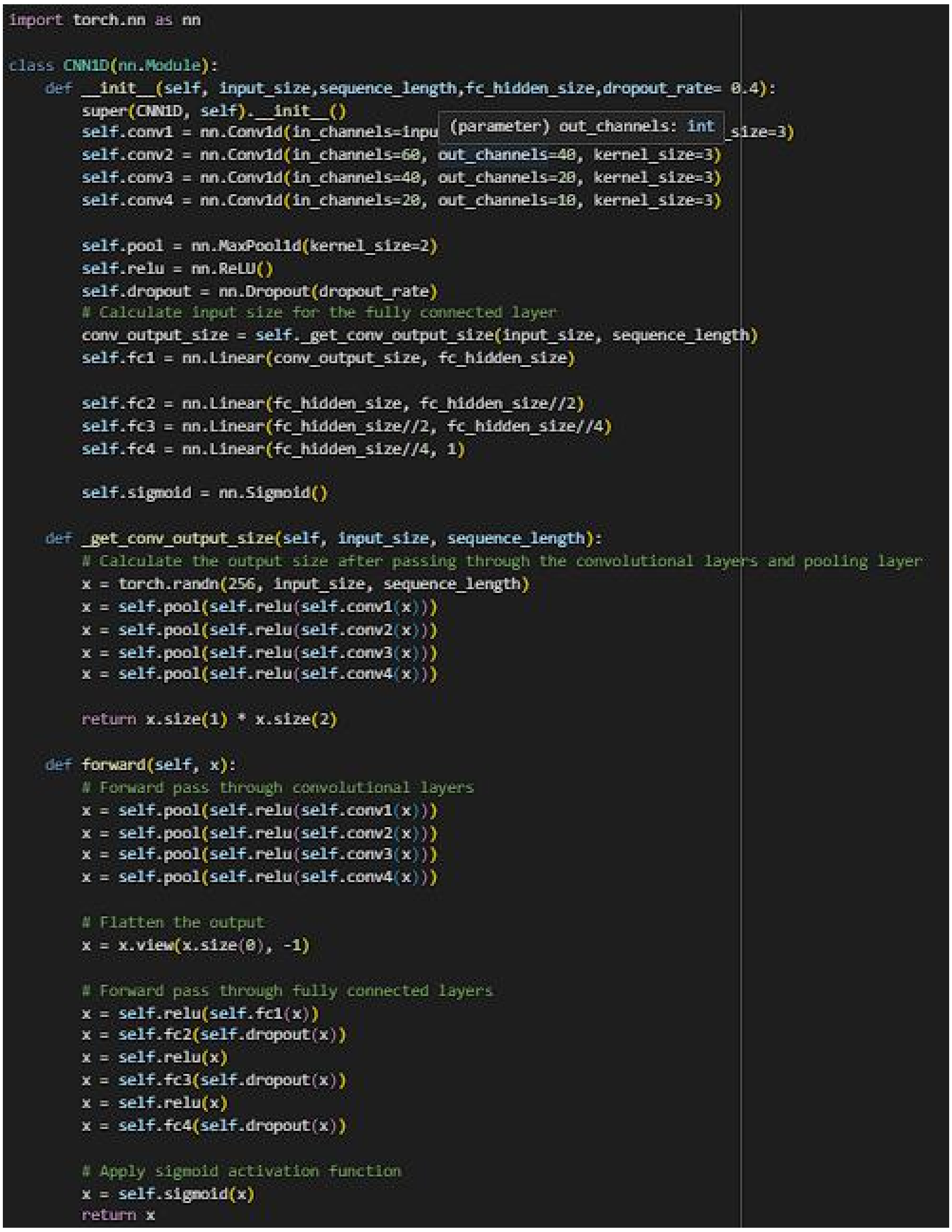
```python
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers,bidirectional=False,dropout_rate=0.5,fc_units = 512):
        super().__init__()
        self.bidirectional = bidirectional
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size = input_size, hidden_size = hidden_size, num_layers = num_layers,dropout = dropout_rate,batch_first=True,bidirectional=bidirectional)
        self.dropout = nn.Dropout(dropout_rate)
        self.fc1 = nn.Linear(hidden_size, fc_units)
        self.fc2 = nn.Linear(fc_units, fc_units//2)
        self.fc3 = nn.Linear(fc_units//2, fc_units//4)
        self.fc4 = nn.Linear(fc_units//4, 1)

        if(self.bidirectional == True):
            self.fc1 = nn.Linear(hidden_size*2, fc_units*2)
            self.fc2 = nn.Linear(fc_units*2, fc_units)
            self.fc3 = nn.Linear(fc_units, fc_units//2)
            self.fc4 = nn.Linear(fc_units//2, 1)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        if(self.bidirectional == True):
            direction = 2
        else:
            direction = 1

        h0 = torch.zeros(direction*self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(direction*self.num_layers, x.size(0), self.hidden_size).to(x.device)

        out, _ = self.lstm(x, (h0, c0))
        out_fc1 = self.fc1(out[:, -1, :])  # Use only the last output
        out_fc1 = self.relu(out_fc1)
        out_fc2 = self.fc2(self.dropout(out_fc1))
        out_fc2 = self.relu(out_fc2)
        out_fc3 = self.fc3(self.dropout(out_fc2))
        out_fc3 = self.relu(out_fc3)
        out_fc4 = self.fc4(self.dropout(out_fc3))
        output = self.sigmoid(out_fc4)
        output = output.view((len(x)))
        return output
```

Figure 4.20 Implementation of LSTM networks



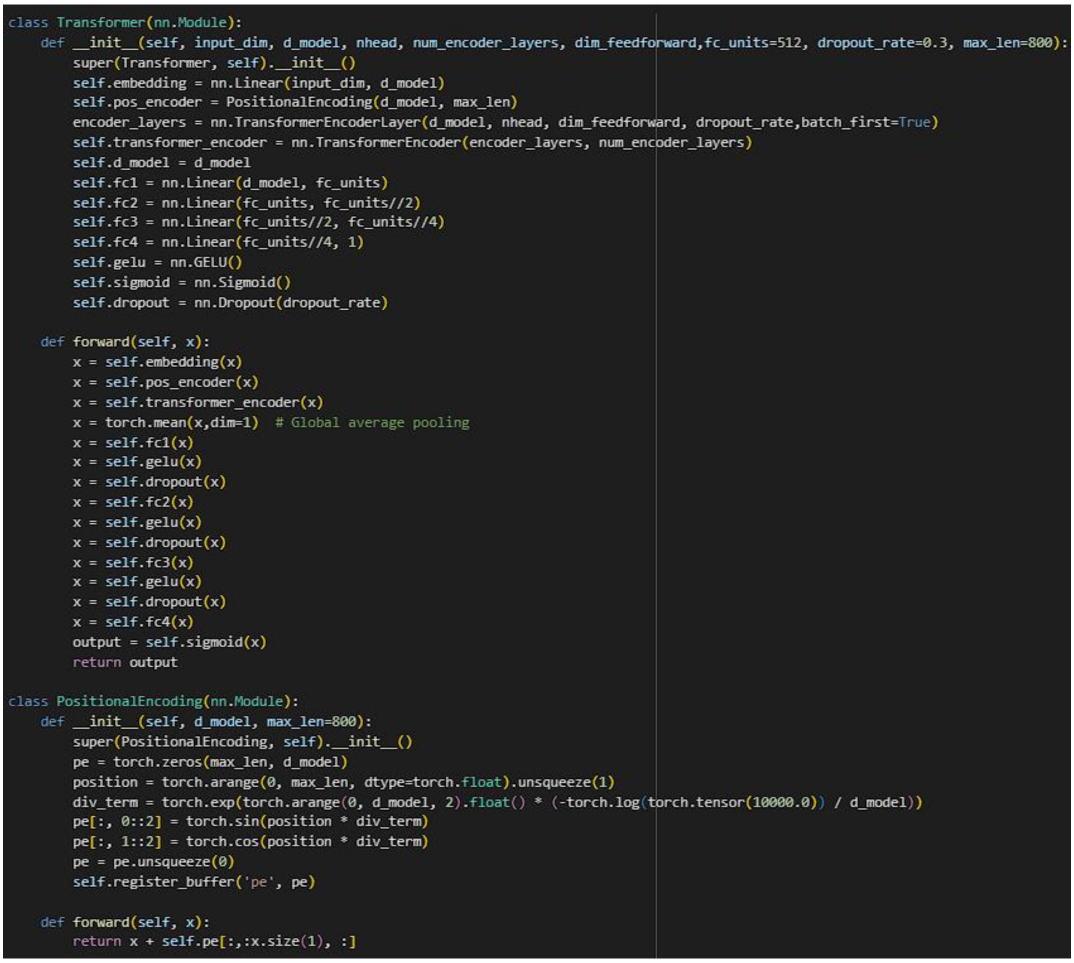
```python
class GRU(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers,bidirectional=False,dropout_rate = 0.5,fc_units = 512):
        super().__init__()
        self.bidirectional = bidirectional
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.gru = nn.GRU(input_size = input_size, hidden_size = hidden_size, num_layers=num_layers, dropout = dropout_rate,batch_first=True,bidirectional=bidirectional)
        self.dropout = nn.Dropout(dropout_rate)
        self.fc1 = nn.Linear(hidden_size, fc_units)
        self.fc2 = nn.Linear(fc_units, fc_units//2)
        self.fc3 = nn.Linear(fc_units//2, fc_units//4)
        self.fc4 = nn.Linear(fc_units//4, 1)

        if(self.bidirectional == True):
            self.fc1 = nn.Linear(hidden_size*2, fc_units*2)
            self.fc2 = nn.Linear(fc_units*2, fc_units)
            self.fc3 = nn.Linear(fc_units, fc_units//2)
            self.fc4 = nn.Linear(fc_units//2, 1)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        if(self.bidirectional == True):
            direction = 2
        else:
            direction = 1

        h0 = torch.zeros(direction*self.num_layers, x.size(0), self.hidden_size).to(x.device)

        out, _ = self.gru(x, h0)
        out_fc1 = self.fc1(out[:, -1, :])  # Use only the last output
        out_fc1 = self.relu(out_fc1)
        out_fc2 = self.fc2(self.dropout(out_fc1))
        out_fc2 = self.relu(out_fc2)
        out_fc3 = self.fc3(self.dropout(out_fc2))
        out_fc3 = self.relu(out_fc3)
        output = self.fc4(self.dropout(out_fc3))
        output = self.sigmoid(output)
        return output
```

Figure 4.21 Implementation of GRU networks



```
import torch.nn as nn

class CNN1D(nn.Module):
    def __init__(self, input_size,sequence_length,fc_hidden_size,dropout_rate= 0.4):
        super(CNN1D, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=inpu (parameter) out_channels: int _size=3)
        self.conv2 = nn.Conv1d(in_channels=60, out_channels=40, kernel_size=3)
        self.conv3 = nn.Conv1d(in_channels=40, out_channels=20, kernel_size=3)
        self.conv4 = nn.Conv1d(in_channels=20, out_channels=10, kernel_size=3)

        self.pool = nn.MaxPool1d(kernel_size=2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout_rate)
        # Calculate input size for the fully connected layer
        conv_output_size = self._get_conv_output_size(input_size, sequence_length)
        self.fc1 = nn.Linear(conv_output_size, fc_hidden_size)

        self.fc2 = nn.Linear(fc_hidden_size, fc_hidden_size//2)
        self.fc3 = nn.Linear(fc_hidden_size//2, fc_hidden_size//4)
        self.fc4 = nn.Linear(fc_hidden_size//4, 1)

        self.sigmoid = nn.Sigmoid()

    def _get_conv_output_size(self, input_size, sequence_length):
        # Calculate the output size after passing through the convolutional layers and pooling layer
        x = torch.randn(256, input_size, sequence_length)
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))
        x = self.pool(self.relu(self.conv4(x)))

        return x.size(1) * x.size(2)

    def forward(self, x):
        # Forward pass through convolutional layers
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))
        x = self.pool(self.relu(self.conv4(x)))

        # Flatten the output
        x = x.view(x.size(0), -1)

        # Forward pass through fully connected layers
        x = self.relu(self.fc1(x))
        x = self.fc2(self.dropout(x))
        x = self.relu(x)
        x = self.fc3(self.dropout(x))
        x = self.relu(x)
        x = self.fc4(self.dropout(x))

        # Apply sigmoid activation function
        x = self.sigmoid(x)
        return x
```

Figure 4.22 Implementation of 1D-CNN network



```python
class Transformer(nn.Module):
    def __init__(self, input_dim, d_model, nhead, num_encoder_layers, dim_feedforward,fc_units=512, dropout_rate=0.3, max_len=800):
        super(Transformer, self).__init__()
        self.embedding = nn.Linear(input_dim, d_model)
        self.pos_encoder = PositionalEncoding(d_model, max_len)
        encoder_layers = nn.TransformerEncoderLayer(d_model, nhead, dim_feedforward, dropout_rate,batch_first=True)
        self.transformer_encoder = nn.TransformerEncoder(encoder_layers, num_encoder_layers)
        self.d_model = d_model
        self.fc1 = nn.Linear(d_model, fc_units)
        self.fc2 = nn.Linear(fc_units, fc_units//2)
        self.fc3 = nn.Linear(fc_units//2, fc_units//4)
        self.fc4 = nn.Linear(fc_units//4, 1)
        self.gelu = nn.GELU()
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(dropout_rate)

    def forward(self, x):
        x = self.embedding(x)
        x = self.pos_encoder(x)
        x = self.transformer_encoder(x)
        x = torch.mean(x,dim=1)  # Global average pooling
        x = self.fc1(x)
        x = self.gelu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.gelu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        x = self.gelu(x)
        x = self.dropout(x)
        x = self.fc4(x)
        output = self.sigmoid(x)
        return output

class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=800):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-torch.log(torch.tensor(10000.0)) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)

    def forward(self, x):
        return x + self.pe[:,:x.size(1), :]
```

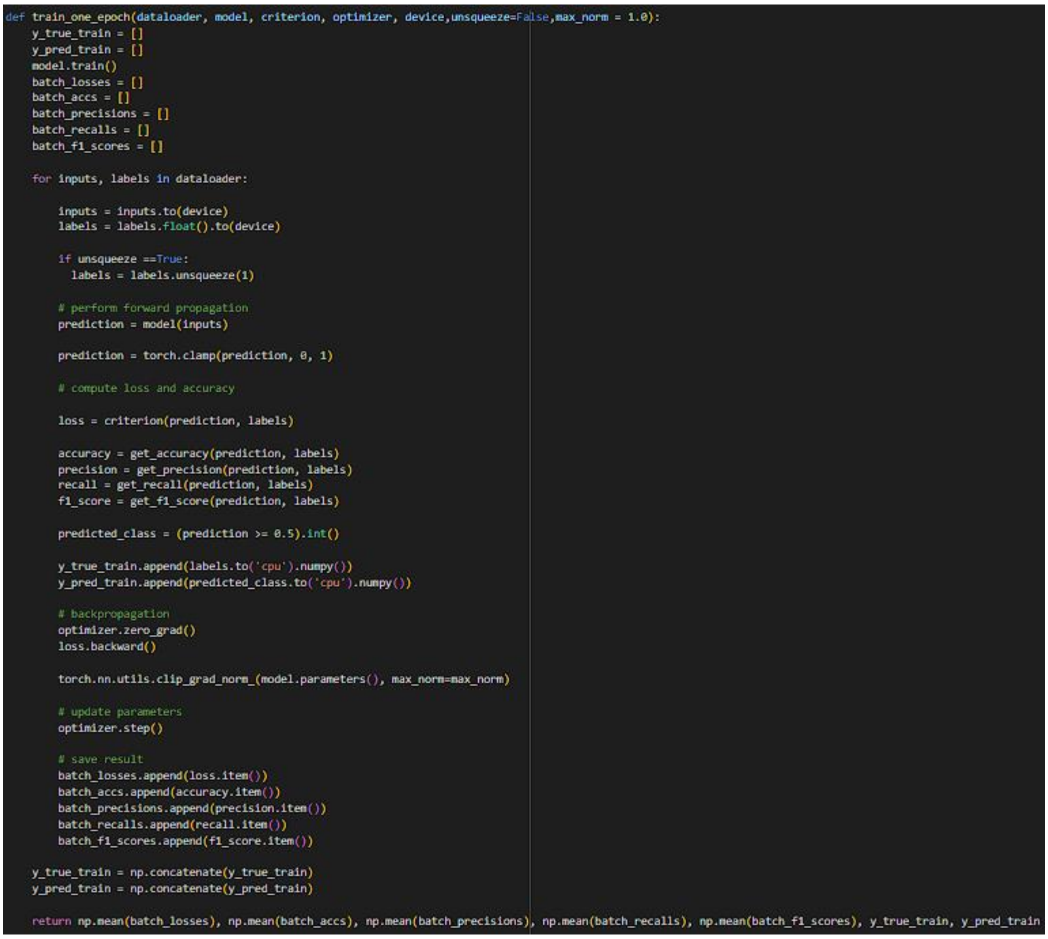
Figure 4.23 Implementation of Transformer classifier

4.5 Helpers

Multiple helper functions were built into this project to assist in the training, validation and testing of the classifiers. Besides, visualizations such as graph plotting and confusion matrix were also established to present the performance metrics in a user-friendly way to increase readability.

4.5.1 Training Loop

The training loop helper function in **Figure 4.24** will first create multiple new variables for each epoch to record and track the performance metrics as well as the true label and the predictions made. The true label and predictions will be utilized in creating a confusion matrix in the visualization part. Performance metrics of the trained models of each fold must be tracked in order to record the related performance metrics of the best-performing model.

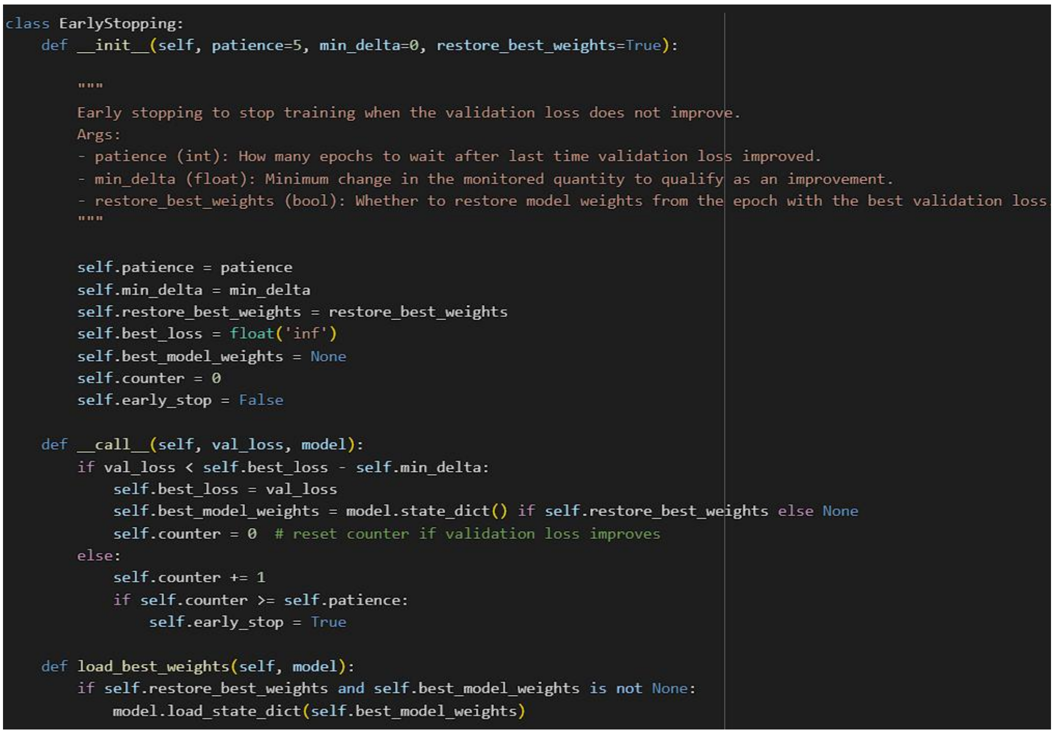


```python
def train_one_epoch(dataloader, model, criterion, optimizer, device,unsqueeze=False,max_norm = 1.0):
    y_true_train = []
    y_pred_train = []
    model.train()
    batch_losses = []
    batch_accs = []
    batch_precisions = []
    batch_recalls = []
    batch_f1_scores = []

    for inputs, labels in dataloader:

        inputs = inputs.to(device)
        labels = labels.float().to(device)

        if unsqueeze ==True:
          labels = labels.unsqueeze(1)

        # perform forward propagation
        prediction = model(inputs)

        prediction = torch.clamp(prediction, 0, 1)

        # compute loss and accuracy

        loss = criterion(prediction, labels)

        accuracy = get_accuracy(prediction, labels)
        precision = get_precision(prediction, labels)
        recall = get_recall(prediction, labels)
        f1_score = get_f1_score(prediction, labels)

        predicted_class = (prediction >= 0.5).int()

        y_true_train.append(labels.to('cpu').numpy())
        y_pred_train.append(predicted_class.to('cpu').numpy())

        # backpropagation
        optimizer.zero_grad()
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=max_norm)

        # update parameters
        optimizer.step()

        # save result
        batch_losses.append(loss.item())
        batch_accs.append(accuracy.item())
        batch_precisions.append(precision.item())
        batch_recalls.append(recall.item())
        batch_f1_scores.append(f1_score.item())

    y_true_train = np.concatenate(y_true_train)
    y_pred_train = np.concatenate(y_pred_train)

    return np.mean(batch_losses), np.mean(batch_accs), np.mean(batch_precisions), np.mean(batch_recalls), np.mean(batch_f1_scores), y_true_train, y_pred_train
```

4.24 Training function

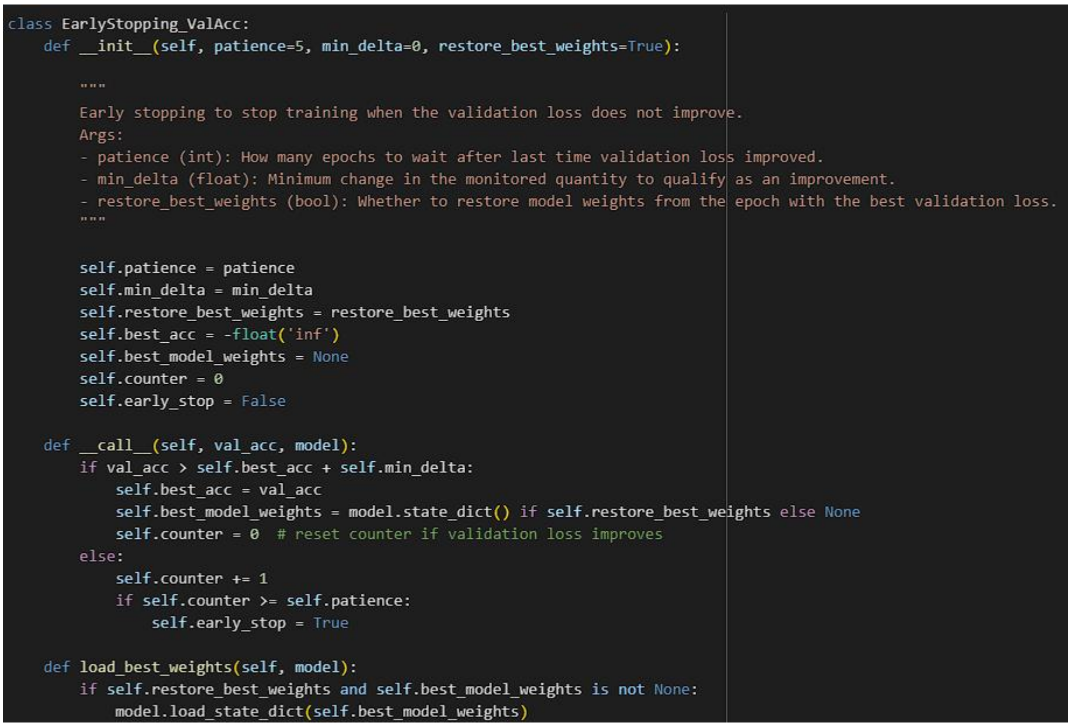
4.5.2 Early Stop Mechanism

In this project, in order to reduce the waste of computational resources and time, two early stopping mechanisms as shown in **Figure 4.25** and **Figure 4.26** were introduced into the training process. A suitable parameter setting including patience (how many epochs to wait after the last improvement of validation loss/accuracy) and min_delta (minimum changes of figures of validation loss/accuracy can be considered improvement) can help track back to the best-performing model's parameters and its performance metrics. An example of how early stopping works is shown in **Figure 4.27**. In the inter-patient training and validation process, the early stopping used validation accuracy as the standard to decide whether to perform early stopping while validation loss was used in the intra-patient process.



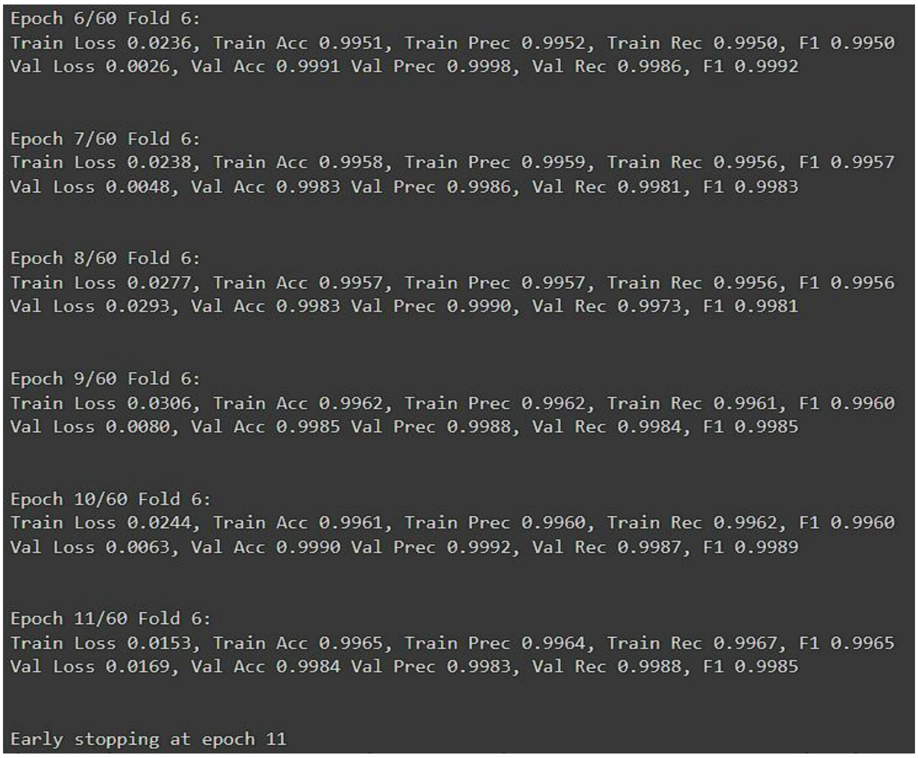
```
class EarlyStopping:
    def __init__(self, patience=5, min_delta=0, restore_best_weights=True):

        """
        Early stopping to stop training when the validation loss does not improve.
        Args:
        - patience (int): How many epochs to wait after last time validation loss improved.
        - min_delta (float): Minimum change in the monitored quantity to qualify as an improvement.
        - restore_best_weights (bool): Whether to restore model weights from the epoch with the best validation loss
        """

        self.patience = patience
        self.min_delta = min_delta
        self.restore_best_weights = restore_best_weights
        self.best_loss = float('inf')
        self.best_model_weights = None
        self.counter = 0
        self.early_stop = False

    def __call__(self, val_loss, model):
        if val_loss < self.best_loss - self.min_delta:
            self.best_loss = val_loss
            self.best_model_weights = model.state_dict() if self.restore_best_weights else None
            self.counter = 0  # reset counter if validation loss improves
        else:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True

    def load_best_weights(self, model):
        if self.restore_best_weights and self.best_model_weights is not None:
            model.load_state_dict(self.best_model_weights)
```

Figure 4.25 Early Stop Mechanism (Validation Loss)



```
class EarlyStopping_ValAcc:
    def __init__(self, patience=5, min_delta=0, restore_best_weights=True):

        """
        Early stopping to stop training when the validation loss does not improve.
        Args:
        - patience (int): How many epochs to wait after last time validation loss improved.
        - min_delta (float): Minimum change in the monitored quantity to qualify as an improvement.
        - restore_best_weights (bool): Whether to restore model weights from the epoch with the best validation loss.
        """

        self.patience = patience
        self.min_delta = min_delta
        self.restore_best_weights = restore_best_weights
        self.best_acc = -float('inf')
        self.best_model_weights = None
        self.counter = 0
        self.early_stop = False

    def __call__(self, val_acc, model):
        if val_acc > self.best_acc + self.min_delta:
            self.best_acc = val_acc
            self.best_model_weights = model.state_dict() if self.restore_best_weights else None
            self.counter = 0  # reset counter if validation loss improves
        else:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True

    def load_best_weights(self, model):
        if self.restore_best_weights and self.best_model_weights is not None:
            model.load_state_dict(self.best_model_weights)
```

Figure 4.26 Early Stop Mechanism (Validation Accuracy)

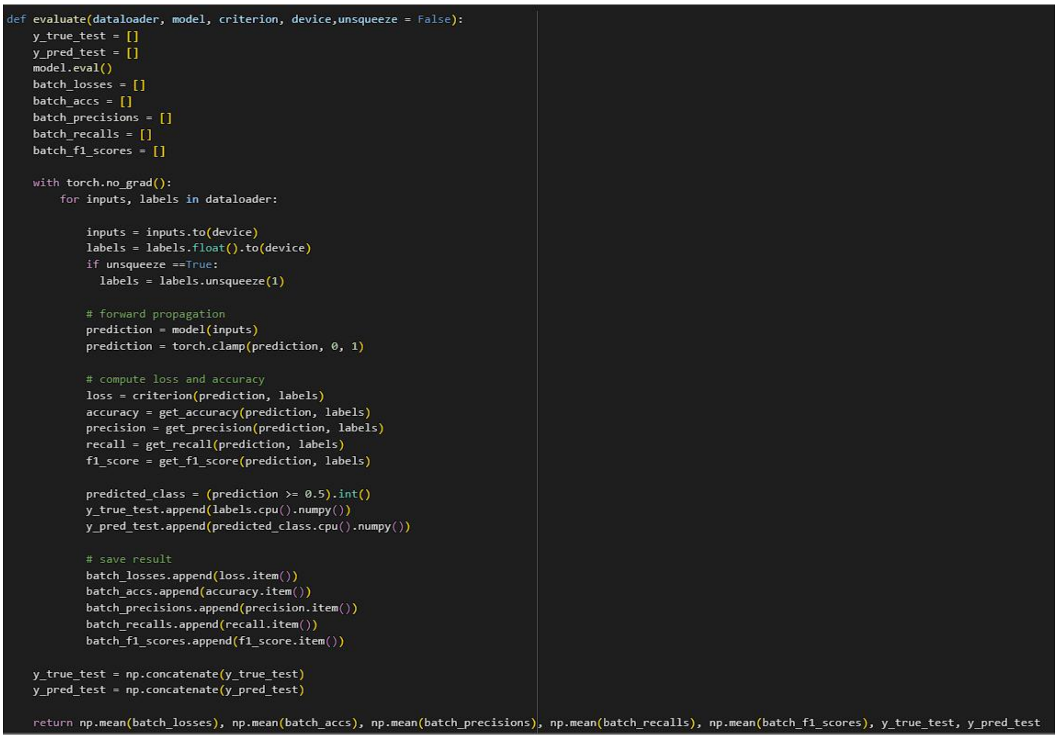


```
Epoch 6/60 Fold 6:
Train Loss 0.0236, Train Acc 0.9951, Train Prec 0.9952, Train Rec 0.9950, F1 0.9950
Val Loss 0.0026, Val Acc 0.9991 Val Prec 0.9998, Val Rec 0.9986, F1 0.9992


Epoch 7/60 Fold 6:
Train Loss 0.0238, Train Acc 0.9958, Train Prec 0.9959, Train Rec 0.9956, F1 0.9957
Val Loss 0.0048, Val Acc 0.9983 Val Prec 0.9986, Val Rec 0.9981, F1 0.9983


Epoch 8/60 Fold 6:
Train Loss 0.0277, Train Acc 0.9957, Train Prec 0.9957, Train Rec 0.9956, F1 0.9956
Val Loss 0.0293, Val Acc 0.9983 Val Prec 0.9990, Val Rec 0.9973, F1 0.9981


Epoch 9/60 Fold 6:
Train Loss 0.0306, Train Acc 0.9962, Train Prec 0.9962, Train Rec 0.9961, F1 0.9960
Val Loss 0.0080, Val Acc 0.9985 Val Prec 0.9988, Val Rec 0.9984, F1 0.9985


Epoch 10/60 Fold 6:
Train Loss 0.0244, Train Acc 0.9961, Train Prec 0.9960, Train Rec 0.9962, F1 0.9960
Val Loss 0.0063, Val Acc 0.9990 Val Prec 0.9992, Val Rec 0.9987, F1 0.9989


Epoch 11/60 Fold 6:
Train Loss 0.0153, Train Acc 0.9965, Train Prec 0.9964, Train Rec 0.9967, F1 0.9965
Val Loss 0.0169, Val Acc 0.9984 Val Prec 0.9983, Val Rec 0.9988, F1 0.9985


Early stopping at epoch 11
```

4.27 An example of early stopping

4.5.3 Testing

The testing helper function in **Figure 4.28** was doing the same job as the training helper function but without the backpropagation process. Identically, the performance metrics will be recorded for visualization through a graph and a confusion matrix.

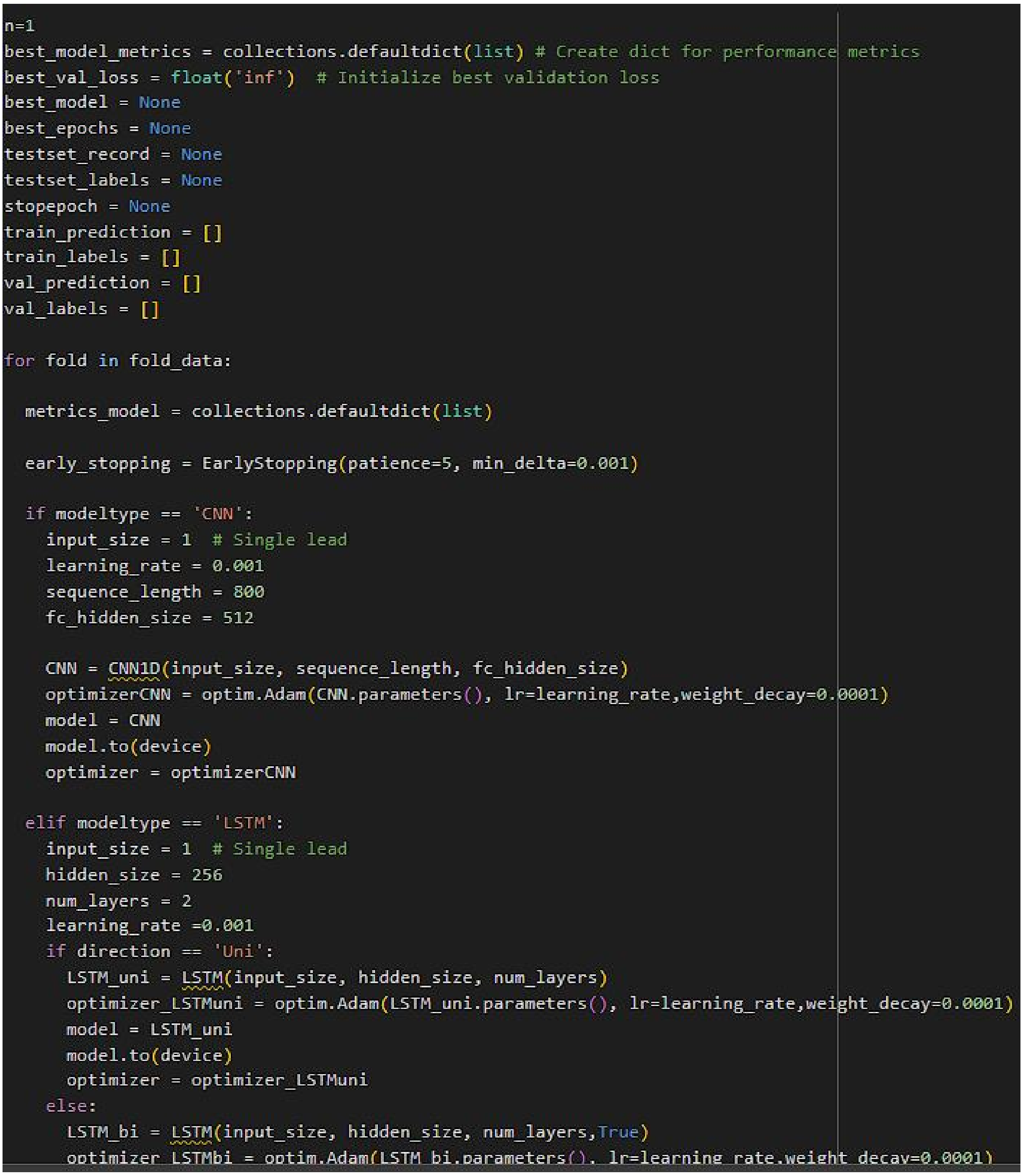


```
def evaluate(dataloader, model, criterion, device,unsqueeze = False):
    y_true_test = []
    y_pred_test = []
    model.eval()
    batch_losses = []
    batch_accs = []
    batch_precisions = []
    batch_recalls = []
    batch_f1_scores = []

    with torch.no_grad():
        for inputs, labels in dataloader:

            inputs = inputs.to(device)
            labels = labels.float().to(device)
            if unsqueeze ==True:
              labels = labels.unsqueeze(1)

            # forward propagation
            prediction = model(inputs)
            prediction = torch.clamp(prediction, 0, 1)

            # compute loss and accuracy
            loss = criterion(prediction, labels)
            accuracy = get_accuracy(prediction, labels)
            precision = get_precision(prediction, labels)
            recall = get_recall(prediction, labels)
            f1_score = get_f1_score(prediction, labels)

            predicted_class = (prediction >= 0.5).int()
            y_true_test.append(labels.cpu().numpy())
            y_pred_test.append(predicted_class.cpu().numpy())

            # save result
            batch_losses.append(loss.item())
            batch_accs.append(accuracy.item())
            batch_precisions.append(precision.item())
            batch_recalls.append(recall.item())
            batch_f1_scores.append(f1_score.item())

    y_true_test = np.concatenate(y_true_test)
    y_pred_test = np.concatenate(y_pred_test)

    return np.mean(batch_losses), np.mean(batch_accs), np.mean(batch_precisions), np.mean(batch_recalls), np.mean(batch_f1_scores), y_true_test, y_pred_test
```

4.28 Testing function

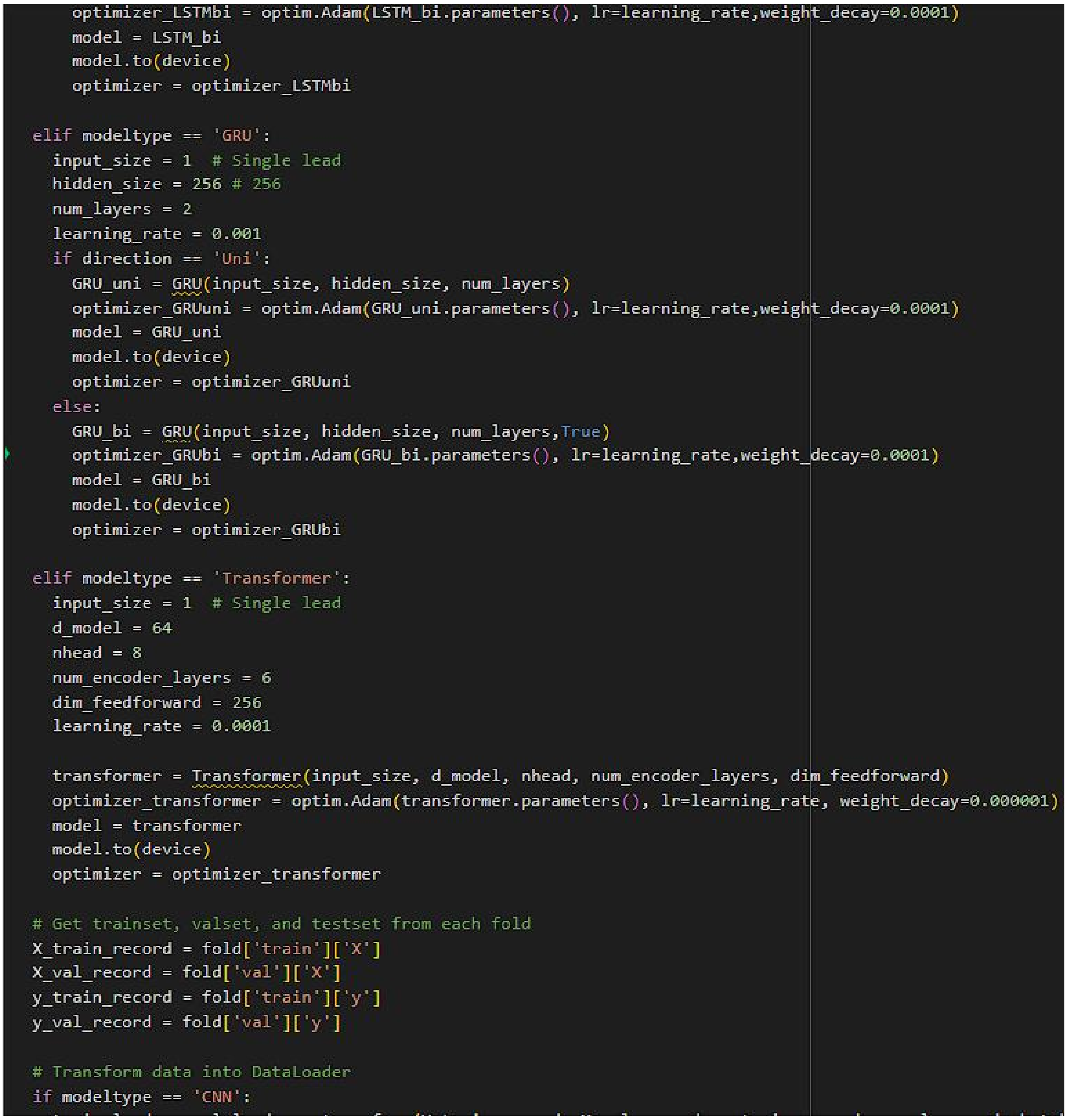
4.6 Main function

Multiple variables were created to track the best-performing classifier, the number of epochs to get the greatest performance and the pair of labels and predictions made. In every fold, the classifier, early stopping and performance metrics will be reinitialized to re-train a new classifier to compare with the performance metrics of the current best-performing classifier. All the implementations are shown in **Figure 4.29** until **Figure 4.32.**



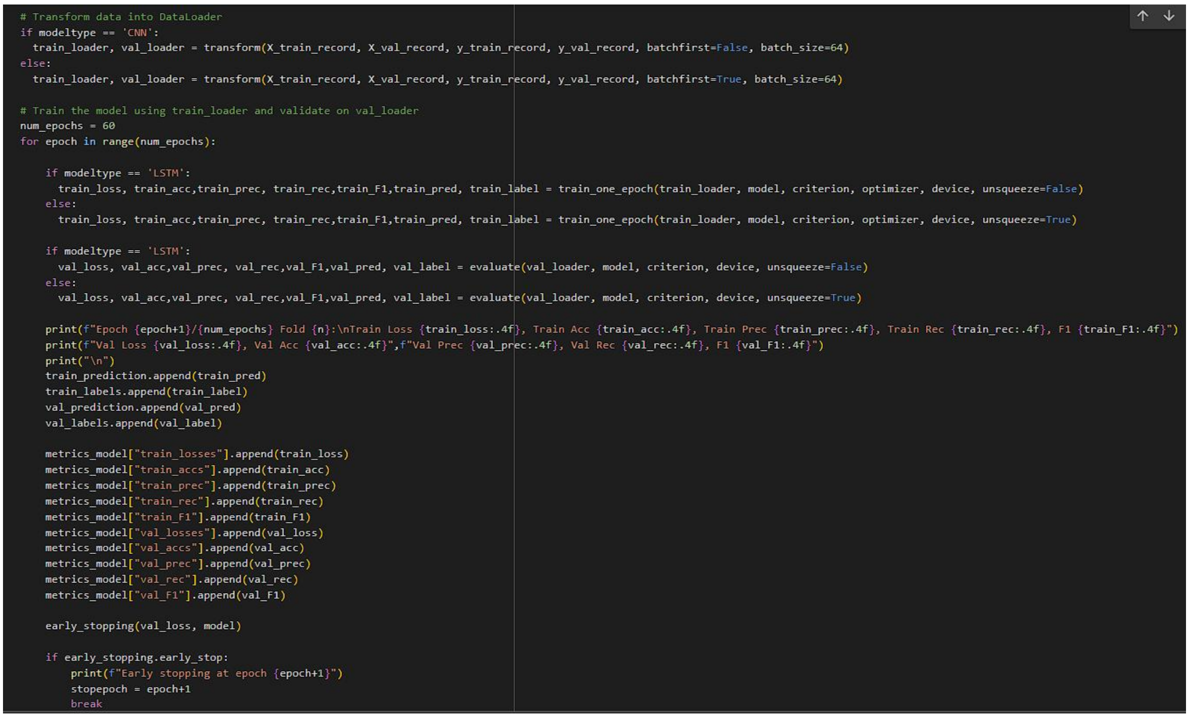
```
n=1
best_model_metrics = collections.defaultdict(list) # Create dict for performance metrics
best_val_loss = float('inf')  # Initialize best validation loss
best_model = None
best_epochs = None
testset_record = None
testset_labels = None
stopepoch = None
train_prediction = []
train_labels = []
val_prediction = []
val_labels = []

for fold in fold_data:

  metrics_model = collections.defaultdict(list)

  early_stopping = EarlyStopping(patience=5, min_delta=0.001)

  if modeltype == 'CNN':
    input_size = 1  # Single lead
    learning_rate = 0.001
    sequence_length = 800
    fc_hidden_size = 512

    CNN = CNN1D(input_size, sequence_length, fc_hidden_size)
    optimizerCNN = optim.Adam(CNN.parameters(), lr=learning_rate,weight_decay=0.0001)
    model = CNN
    model.to(device)
    optimizer = optimizerCNN

  elif modeltype == 'LSTM':
    input_size = 1  # Single lead
    hidden_size = 256
    num_layers = 2
    learning_rate =0.001
    if direction == 'Uni':
      LSTM_uni = LSTM(input_size, hidden_size, num_layers)
      optimizer_LSTMuni = optim.Adam(LSTM_uni.parameters(), lr=learning_rate,weight_decay=0.0001)
      model = LSTM_uni
      model.to(device)
      optimizer = optimizer_LSTMuni
    else:
      LSTM_bi = LSTM(input_size, hidden_size, num_layers,True)
      optimizer_LSTMbi = optim.Adam(LSTM_bi.parameters(), lr=learning_rate,weight_decay=0.0001)
```

Figure 4.29 Main code part 1



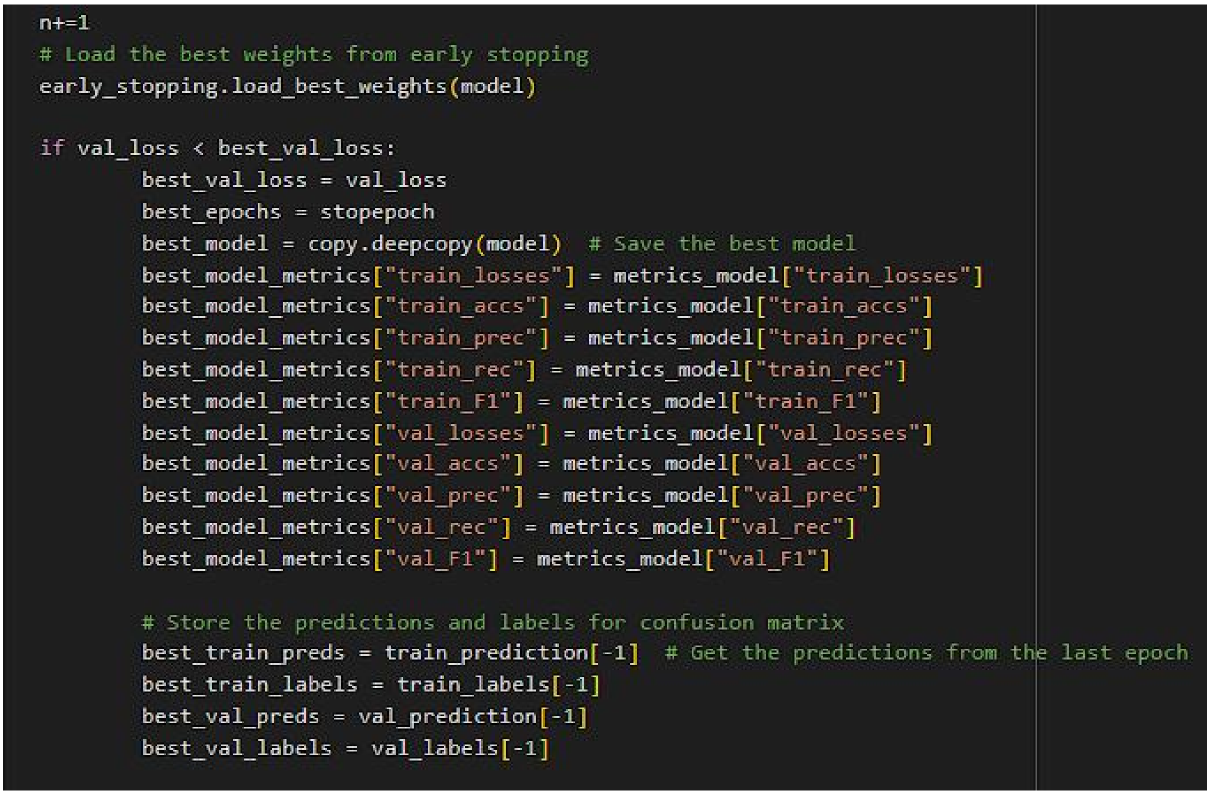
```
      optimizer_LSTMbi = optim.Adam(LSTM_bi.parameters(), lr=learning_rate,weight_decay=0.0001)
      model = LSTM_bi
      model.to(device)
      optimizer = optimizer_LSTMbi

  elif modeltype == 'GRU':
    input_size = 1  # Single lead
    hidden_size = 256 # 256
    num_layers = 2
    learning_rate = 0.001
    if direction == 'Uni':
      GRU_uni = GRU(input_size, hidden_size, num_layers)
      optimizer_GRUuni = optim.Adam(GRU_uni.parameters(), lr=learning_rate,weight_decay=0.0001)
      model = GRU_uni
      model.to(device)
      optimizer = optimizer_GRUuni
    else:
      GRU_bi = GRU(input_size, hidden_size, num_layers,True)
      optimizer_GRUbi = optim.Adam(GRU_bi.parameters(), lr=learning_rate,weight_decay=0.0001)
      model = GRU_bi
      model.to(device)
      optimizer = optimizer_GRUbi

  elif modeltype == 'Transformer':
    input_size = 1  # Single lead
    d_model = 64
    nhead = 8
    num_encoder_layers = 6
    dim_feedforward = 256
    learning_rate = 0.0001

    transformer = Transformer(input_size, d_model, nhead, num_encoder_layers, dim_feedforward)
    optimizer_transformer = optim.Adam(transformer.parameters(), lr=learning_rate, weight_decay=0.000001)
    model = transformer
    model.to(device)
    optimizer = optimizer_transformer

  # Get trainset, valset, and testset from each fold
  X_train_record = fold['train']['X']
  X_val_record = fold['val']['X']
  y_train_record = fold['train']['y']
  y_val_record = fold['val']['y']

  # Transform data into DataLoader
  if modeltype == 'CNN':
```

Figure 4.30 Main code part 2



```
# Transform data into DataLoader
if modeltype == 'CNN':
  train_loader, val_loader = transform(X_train_record, X_val_record, y_train_record, y_val_record, batchfirst=False, batch_size=64)
else:
  train_loader, val_loader = transform(X_train_record, X_val_record, y_train_record, y_val_record, batchfirst=True, batch_size=64)

# Train the model using train_loader and validate on val_loader
num_epochs = 60
for epoch in range(num_epochs):

    if modeltype == 'LSTM':
      train_loss, train_acc,train_prec, train_rec,train_F1,train_pred, train_label = train_one_epoch(train_loader, model, criterion, optimizer, device, unsqueeze=False)
    else:
      train_loss, train_acc,train_prec, train_rec,train_F1,train_pred, train_label = train_one_epoch(train_loader, model, criterion, optimizer, device, unsqueeze=True)

    if modeltype == 'LSTM':
      val_loss, val_acc,val_prec, val_rec,val_F1,val_pred, val_label = evaluate(val_loader, model, criterion, device, unsqueeze=False)
    else:
      val_loss, val_acc,val_prec, val_rec,val_F1,val_pred, val_label = evaluate(val_loader, model, criterion, device, unsqueeze=True)

    print(f"Epoch {epoch+1}/{num_epochs} Fold {n}:\nTrain Loss {train_loss:.4f}, Train Acc {train_acc:.4f}, Train Prec {train_prec:.4f}, Train Rec {train_rec:.4f}, F1 {train_F1:.4f}")
    print(f"Val Loss {val_loss:.4f}, Val Acc {val_acc:.4f}",f"Val Prec {val_prec:.4f}, Val Rec {val_rec:.4f}, F1 {val_F1:.4f}")
    print("\n")
    train_prediction.append(train_pred)
    train_labels.append(train_label)
    val_prediction.append(val_pred)
    val_labels.append(val_label)

    metrics_model["train_losses"].append(train_loss)
    metrics_model["train_accs"].append(train_acc)
    metrics_model["train_prec"].append(train_prec)
    metrics_model["train_rec"].append(train_rec)
    metrics_model["train_F1"].append(train_F1)
    metrics_model["val_losses"].append(val_loss)
    metrics_model["val_accs"].append(val_acc)
    metrics_model["val_prec"].append(val_prec)
    metrics_model["val_rec"].append(val_rec)
    metrics_model["val_F1"].append(val_F1)

    early_stopping(val_loss, model)

    if early_stopping.early_stop:
        print(f"Early stopping at epoch {epoch+1}")
        stopepoch = epoch+1
        break
```

Figure 4.31 Main code part 3

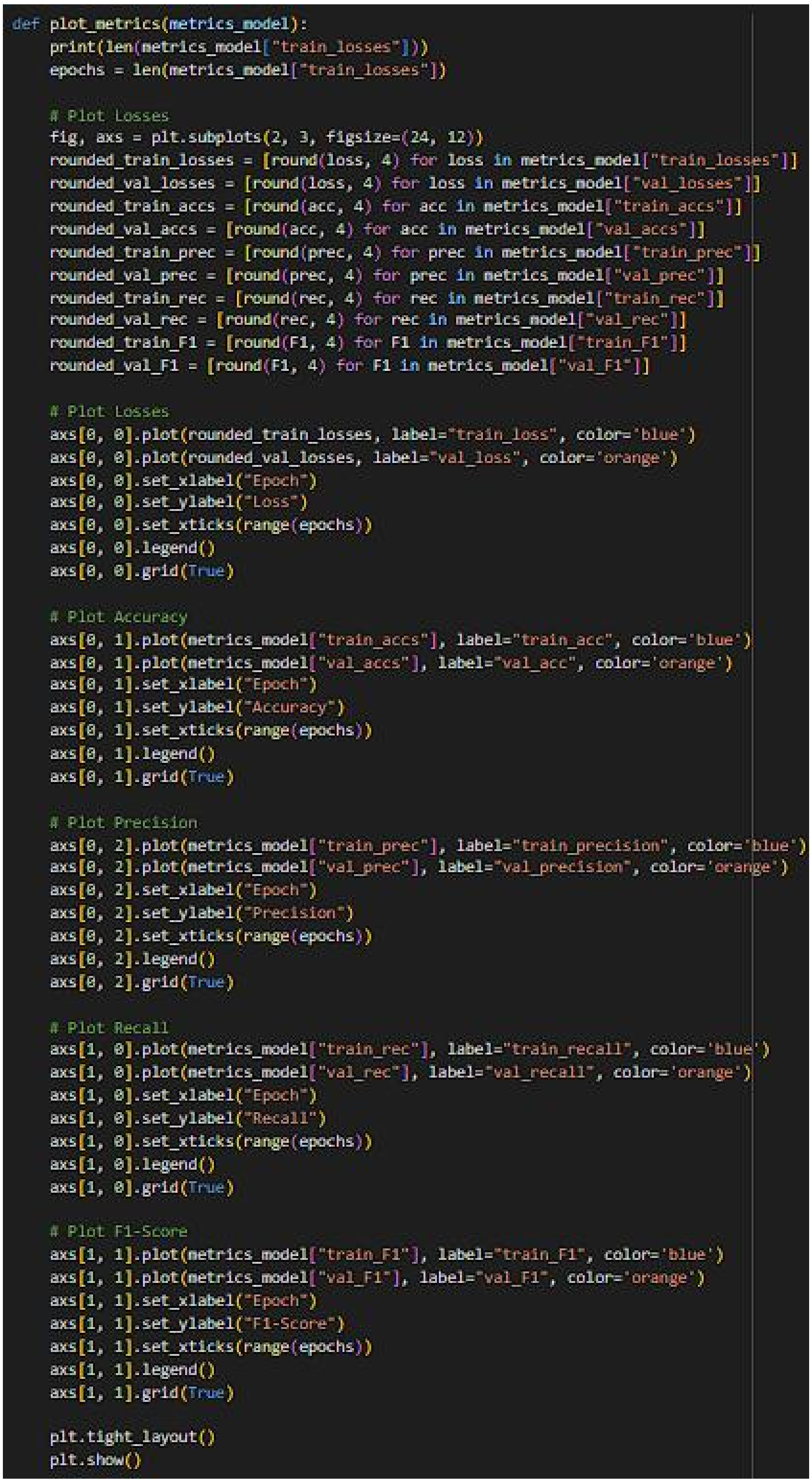


```
n+=1
# Load the best weights from early stopping
early_stopping.load_best_weights(model)

if val_loss < best_val_loss:
        best_val_loss = val_loss
        best_epochs = stopepoch
        best_model = copy.deepcopy(model)  # Save the best model
        best_model_metrics["train_losses"] = metrics_model["train_losses"]
        best_model_metrics["train_accs"] = metrics_model["train_accs"]
        best_model_metrics["train_prec"] = metrics_model["train_prec"]
        best_model_metrics["train_rec"] = metrics_model["train_rec"]
        best_model_metrics["train_F1"] = metrics_model["train_F1"]
        best_model_metrics["val_losses"] = metrics_model["val_losses"]
        best_model_metrics["val_accs"] = metrics_model["val_accs"]
        best_model_metrics["val_prec"] = metrics_model["val_prec"]
        best_model_metrics["val_rec"] = metrics_model["val_rec"]
        best_model_metrics["val_F1"] = metrics_model["val_F1"]

        # Store the predictions and labels for confusion matrix
        best_train_preds = train_prediction[-1]  # Get the predictions from the last epoch
        best_train_labels = train_labels[-1]
        best_val_preds = val_prediction[-1]
        best_val_labels = val_labels[-1]
```

Figure 4.32 Main code part 4

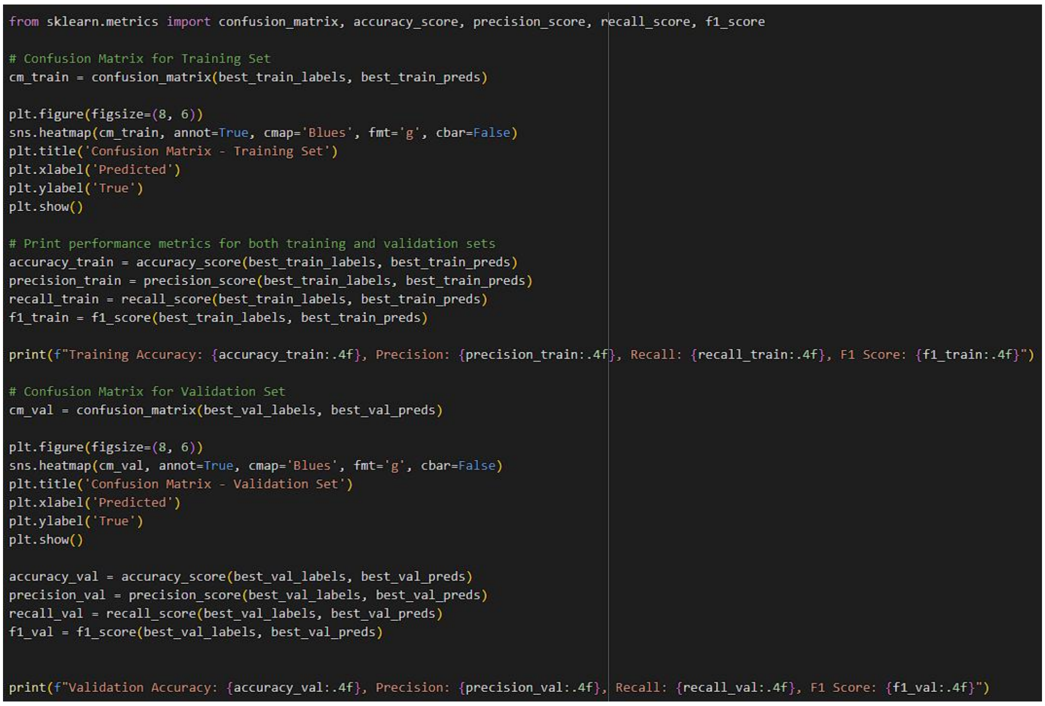
4.7 Visualization

Two methods were available in this project to visualize the performance metrics. Plotting a graph could show the changes in the performance metrics throughout the epochs while the confusion matrix could give an intuitive observation of the correct and wrong decisions made by the classifier. The implementation of graph plotting is available in **Figure 4.33** while **Figure 4.34** and **Figure 4.35** are responsible for the implementation of a confusion matrix.



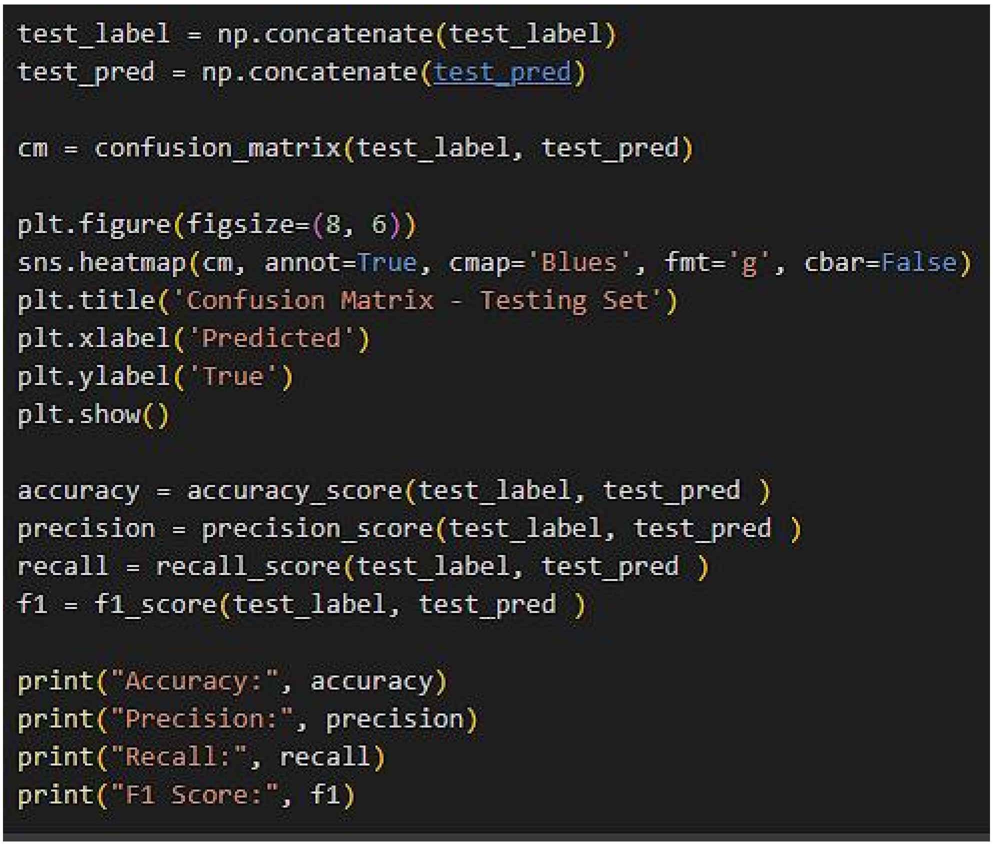
```python
def plot_metrics(metrics_model):
    print(len(metrics_model["train_losses"]))
    epochs = len(metrics_model["train_losses"])

    # Plot Losses
    fig, axs = plt.subplots(2, 3, figsize=(24, 12))
    rounded_train_losses = [round(loss, 4) for loss in metrics_model["train_losses"]]
    rounded_val_losses = [round(loss, 4) for loss in metrics_model["val_losses"]]
    rounded_train_accs = [round(acc, 4) for acc in metrics_model["train_accs"]]
    rounded_val_accs = [round(acc, 4) for acc in metrics_model["val_accs"]]
    rounded_train_prec = [round(prec, 4) for prec in metrics_model["train_prec"]]
    rounded_val_prec = [round(prec, 4) for prec in metrics_model["val_prec"]]
    rounded_train_rec = [round(rec, 4) for rec in metrics_model["train_rec"]]
    rounded_val_rec = [round(rec, 4) for rec in metrics_model["val_rec"]]
    rounded_train_F1 = [round(F1, 4) for F1 in metrics_model["train_F1"]]
    rounded_val_F1 = [round(F1, 4) for F1 in metrics_model["val_F1"]]

    # Plot Losses
    axs[0, 0].plot(rounded_train_losses, label="train_loss", color='blue')
    axs[0, 0].plot(rounded_val_losses, label="val_loss", color='orange')
    axs[0, 0].set_xlabel("Epoch")
    axs[0, 0].set_ylabel("Loss")
    axs[0, 0].set_xticks(range(epochs))
    axs[0, 0].legend()
    axs[0, 0].grid(True)

    # Plot Accuracy
    axs[0, 1].plot(metrics_model["train_accs"], label="train_acc", color='blue')
    axs[0, 1].plot(metrics_model["val_accs"], label="val_acc", color='orange')
    axs[0, 1].set_xlabel("Epoch")
    axs[0, 1].set_ylabel("Accuracy")
    axs[0, 1].set_xticks(range(epochs))
    axs[0, 1].legend()
    axs[0, 1].grid(True)

    # Plot Precision
    axs[0, 2].plot(metrics_model["train_prec"], label="train_precision", color='blue')
    axs[0, 2].plot(metrics_model["val_prec"], label="val_precision", color='orange')
    axs[0, 2].set_xlabel("Epoch")
    axs[0, 2].set_ylabel("Precision")
    axs[0, 2].set_xticks(range(epochs))
    axs[0, 2].legend()
    axs[0, 2].grid(True)

    # Plot Recall
    axs[1, 0].plot(metrics_model["train_rec"], label="train_recall", color='blue')
    axs[1, 0].plot(metrics_model["val_rec"], label="val_recall", color='orange')
    axs[1, 0].set_xlabel("Epoch")
    axs[1, 0].set_ylabel("Recall")
    axs[1, 0].set_xticks(range(epochs))
    axs[1, 0].legend()
    axs[1, 0].grid(True)

    # Plot F1-Score
    axs[1, 1].plot(metrics_model["train_F1"], label="train_F1", color='blue')
    axs[1, 1].plot(metrics_model["val_F1"], label="val_F1", color='orange')
    axs[1, 1].set_xlabel("Epoch")
    axs[1, 1].set_ylabel("F1-Score")
    axs[1, 1].set_xticks(range(epochs))
    axs[1, 1].legend()
    axs[1, 1].grid(True)

    plt.tight_layout()
    plt.show()
```

Figure 4.33 Graph plot for loss, accuracy, precision, recall and F1-score of best-performing model



```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Confusion Matrix for Training Set
cm_train = confusion_matrix(best_train_labels, best_train_preds)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_train, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix - Training Set')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Print performance metrics for both training and validation sets
accuracy_train = accuracy_score(best_train_labels, best_train_preds)
precision_train = precision_score(best_train_labels, best_train_preds)
recall_train = recall_score(best_train_labels, best_train_preds)
f1_train = f1_score(best_train_labels, best_train_preds)

print(f"Training Accuracy: {accuracy_train:.4f}, Precision: {precision_train:.4f}, Recall: {recall_train:.4f}, F1 Score: {f1_train:.4f}")

# Confusion Matrix for Validation Set
cm_val = confusion_matrix(best_val_labels, best_val_preds)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_val, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix - Validation Set')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

accuracy_val = accuracy_score(best_val_labels, best_val_preds)
precision_val = precision_score(best_val_labels, best_val_preds)
recall_val = recall_score(best_val_labels, best_val_preds)
f1_val = f1_score(best_val_labels, best_val_preds)


print(f"Validation Accuracy: {accuracy_val:.4f}, Precision: {precision_val:.4f}, Recall: {recall_val:.4f}, F1 Score: {f1_val:.4f}")
```

Figure 4.34 Confusion matrix of best-performing model's training and validation performance metrics



```
test_label = np.concatenate(test_label)
test_pred = np.concatenate(test_pred)

cm = confusion_matrix(test_label, test_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix - Testing Set')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

accuracy = accuracy_score(test_label, test_pred )
precision = precision_score(test_label, test_pred )
recall = recall_score(test_label, test_pred )
f1 = f1_score(test_label, test_pred )

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Figure 4.35 Confusion matrix of best-performing model's testing performance metrics

4.8 Training & Validation Performance Metrics

Table 4.2 is a summary recording all the performance metrics of the best-performing model in their 10-fold cross-validation. Alternatively, the visualization of the performance metrics of each classifier will be shown in the following sections. Overall, the 1D-CNN classifier had the best performance in both the intra-/inter-patient training and validation process. Other than that, GRU also had good and stable results in both types of analysis, followed by LSTM and Transformer. Throughout the training and validation process, there were no large differences were observed between the intra-patient classifier and the inter-patient classifier. Noticeably, the bi-directional version of LSTM and GRU does help the performance matrix in the intra-patient analysis but not the inter-patient analysis.

Almost all the classifiers are able to learn a good decision function from the training set to detect the presence of MI. Nevertheless, the transformer model was observed to suffer from underfitting which is an incapability to capture and learn features from the signals. This might be because of the reason of lack of complexity (the model is too simple) due to the application of a typical hyperparameter setting while other classifiers with designed architecture are able to converge to 90% above.

Table 4.2 Training and Validation performance metrics of trained classifier in both types of analysis

Performance metrics / Models	Accuracy	Precision	Sensitivity / Recall	F1 Score
Intra Uni - LSTM	Train: 96.87% Val: 96.74%	Train: 96.86% Val: 98.17%	Train: 96.89% Val: 95.25%	Train: 0.9687 Val: 0.9669
Inter Uni - LSTM	Train: 96.02% Val: 96.06%	Train: 96.27% Val: 96.68%	Train: 95.51% Val: 98.77%	Train: 0.9589 Val: 0.9771
Intra Bi - LSTM	Train: 97.21% Val: 97.43%	Train: 97.23% Val: 96.63%	Train: 97.20% Val: 98.30%	Train: 0.9721 Val: 0.9745

Inter Bi - LSTM	Train: 95.21% Val: 90.36%	Train: 96.14% Val: 99.80 %	Train: 93.94% Val: 88.86%	Train: 0.9502 Val: 0.9401
Intra Uni – GRU	Train: 98.11% Val: 98.70%	Train: 98.21% Val: 99.21%	Train: 98.00% Val: 98.18%	Train: 0.9811 Val: 0.9869
Inter Uni – GRU	Train: 97.74% Val: 96.22 %	Train: 97.68% Val: 98.23%	Train: 97.66% Val: 97.31%	Train: 0.9767 Val: 0.9777
Intra Bi – GRU	Train: 97.64% Val: 98.23%	Train: 97.64% Val: 97.40%	Train: 97.65% Val: 99.10%	Train: 0.9764 Val: 0.9824
Inter Bi – GRU	Train: 97.32% Val: 95.37%	Train: 97.42% Val: 94.99%	Train: 97.07% Val: 99.82%	Train: 0.9724 Val: 0.9734
Intra 1D – CNN	Train: 99.66% Val: 99.97%	Train: 99.64% Val: 99.98%	Train: 99.68% Val: 99.95%	Train: 0.9966 Val: 0.9997
Inter 1D - CNN	Train: 99.76% Val: 96.82%	Train: 99.76% Val: 97.02%	Train: 99.75% Val: 99.32%	Train: 0.9975 Val: 0.9815
Intra Transformer	Train: 86.79% Val: 82.01%	Train: 89.29% Val: 78.92%	Train: 83.61% Val: 87.36%	Train: 0.8636 Val: 0.8293
Inter Transformer	Train: 87.80% Val: 86.50%	Train: 88.72% Val: 85.83%	Train: 86.48% Val: 99.14%	Train: 0.8759 Val: 0.9200

4.8.1 Visualization

The performance metrics of each classifier were visualized through the graphs and confusion matrix produced after the completion of the training and validation process as shown in **Figure 4.36** to **Figure 4.67** type by type.

4.8.1.1 Intra-Patient Uni-Directional LSTM

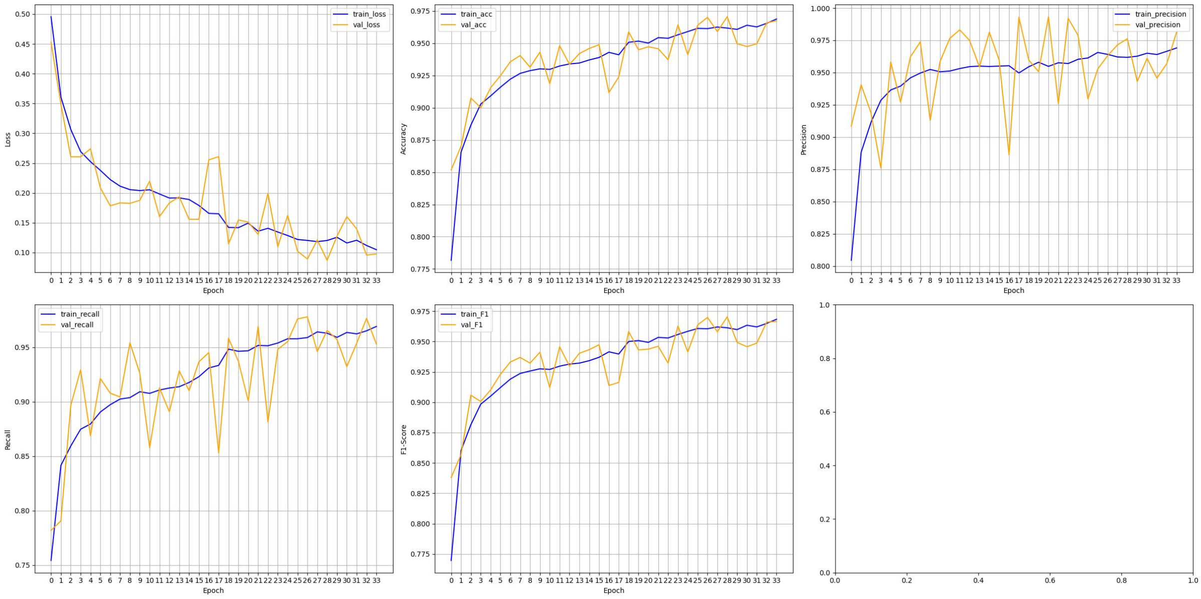
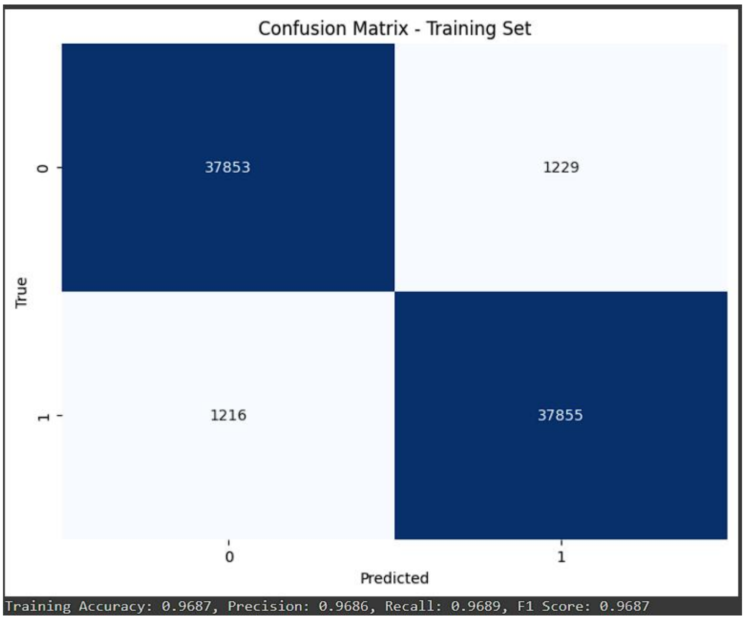
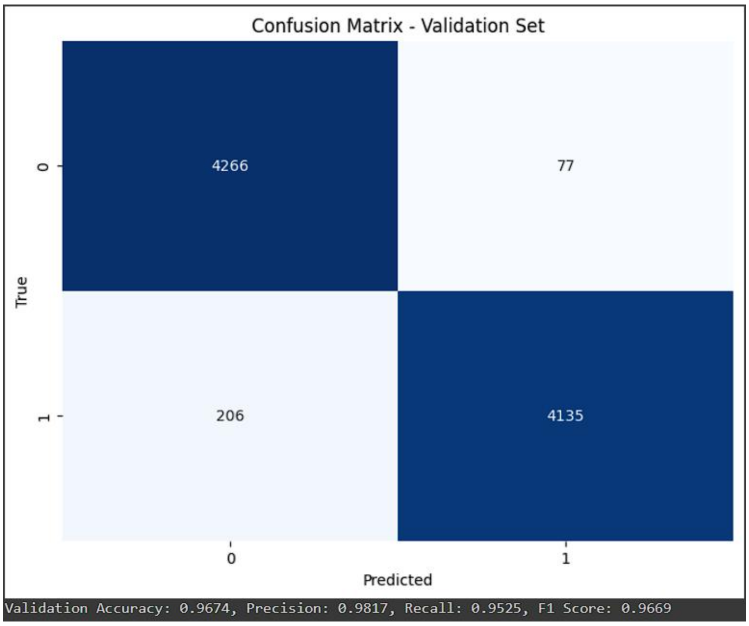


Figure 4.36 Graphs of performance metrics of intra-patient uni-directional LSTM classifier



```
Training Accuracy: 0.9687, Precision: 0.9686, Recall: 0.9689, F1 Score: 0.9687
```

Figure 4.37 Training confusion matrix of intra-patient uni-directional LSTM classifier



```
Validation Accuracy: 0.9674, Precision: 0.9817, Recall: 0.9525, F1 Score: 0.9669
```

Figure 4.38 Validation confusion matrix of intra-patient uni-directional LSTM classifier

4.8.1.2 Inter-Patient Uni-Directional LSTM

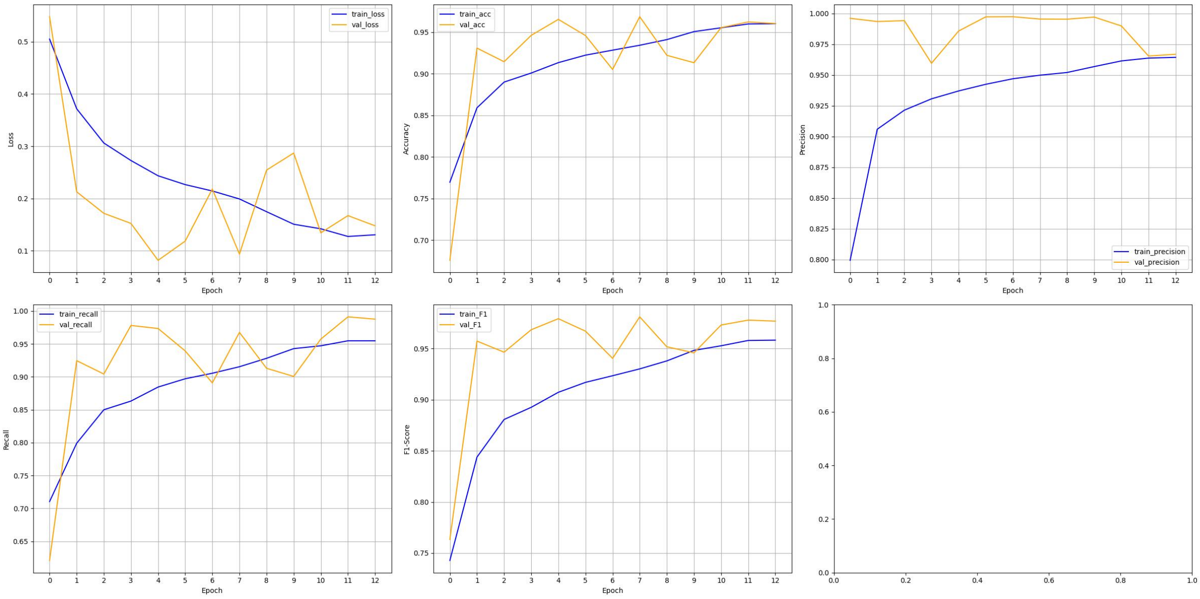


Figure 4.39 Graphs of performance metrics of inter-patient uni-directional LSTM classifier

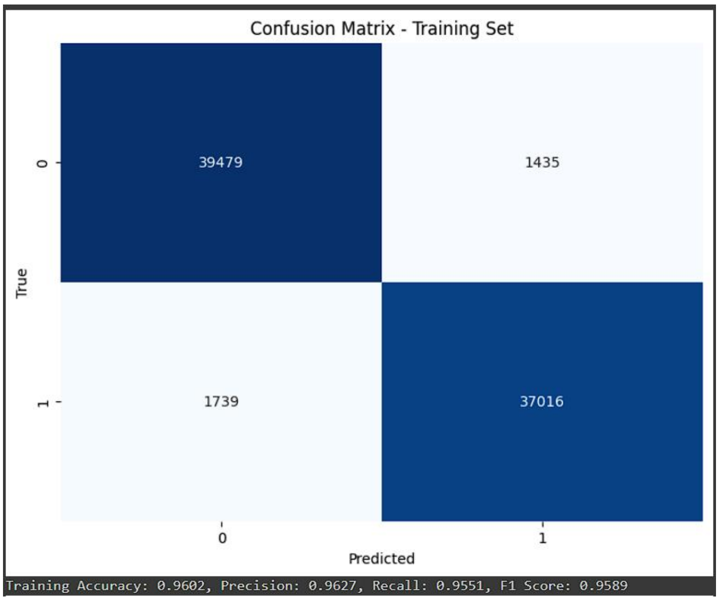


Figure 4.40 Training confusion matrix of inter-patient uni-directional LSTM classifier

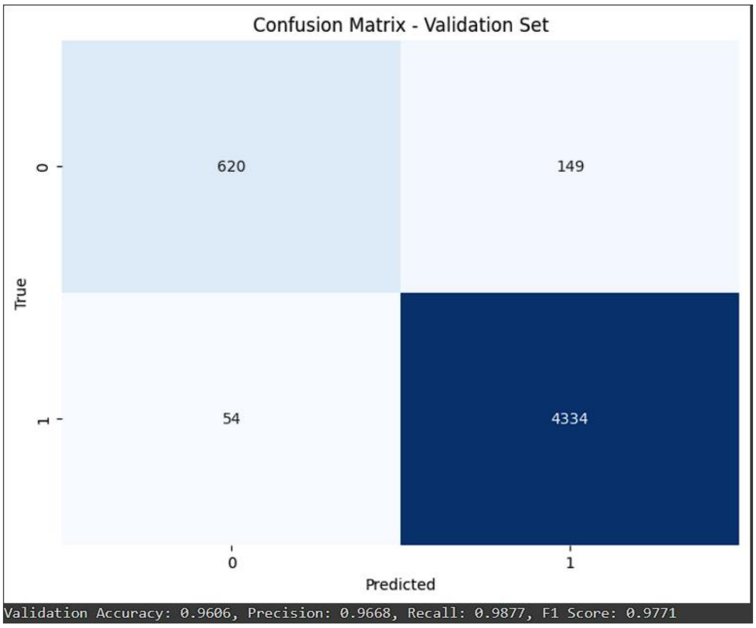


Figure 4.41 Validation confusion matrix of inter-patient uni-directional LSTM classifier

4.8.1.3 Intra-Patient Bi-Directional LSTM

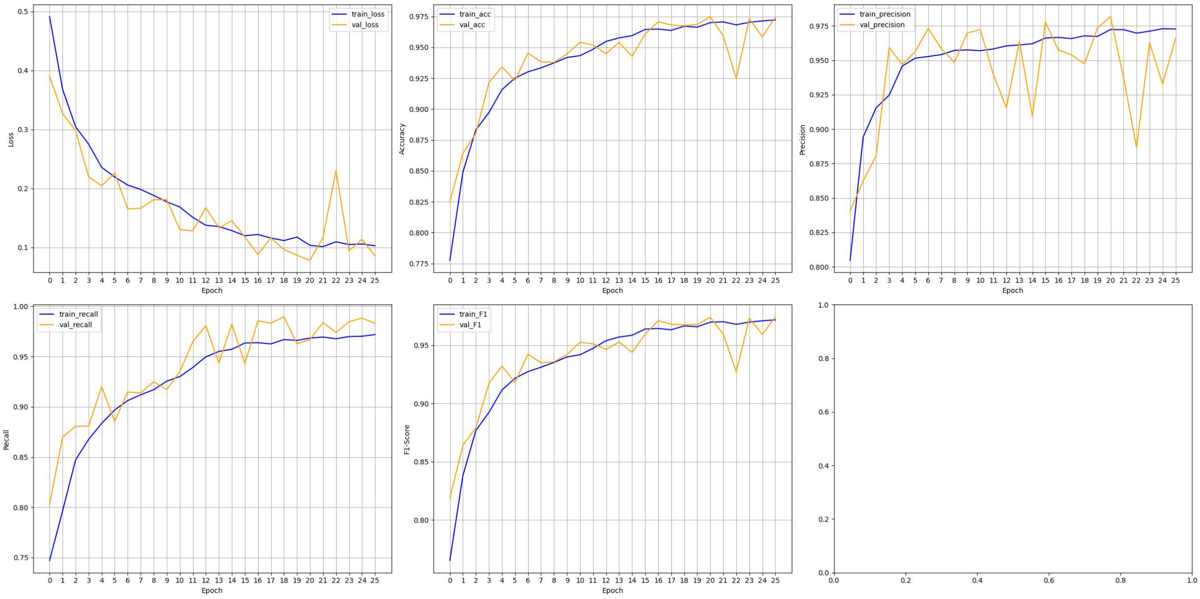
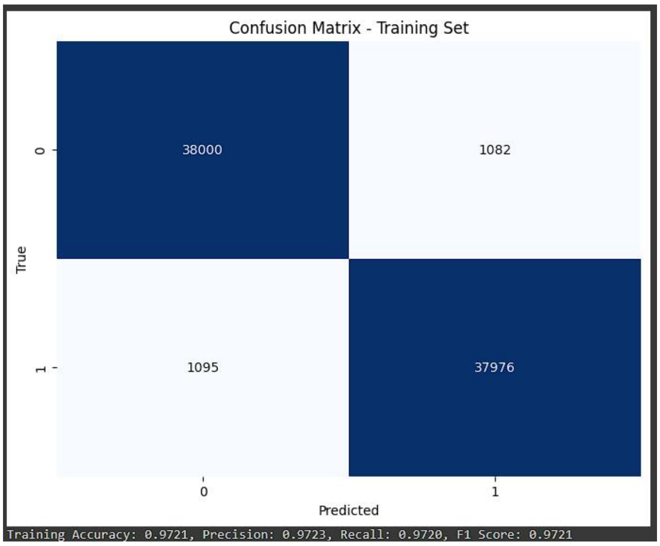
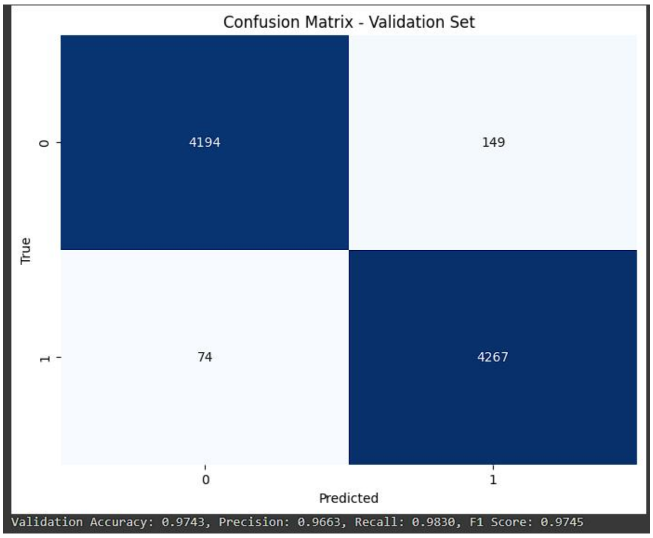


Figure 4.42 Graphs of performance metrics of intra-patient bi-directional LSTM classifier



```
Training Accuracy: 0.9721, Precision: 0.9723, Recall: 0.9720, F1 Score: 0.9721
```

Figure 4.43 Training confusion matrix of intra-patient bi-directional LSTM classifier



```
Validation Accuracy: 0.9743, Precision: 0.9663, Recall: 0.9830, F1 Score: 0.9745
```

Figure 4.44 Validation confusion matrix of intra-patient bi-directional LSTM classifier

4.8.1.4 Inter-Patient Bi-Directional LSTM

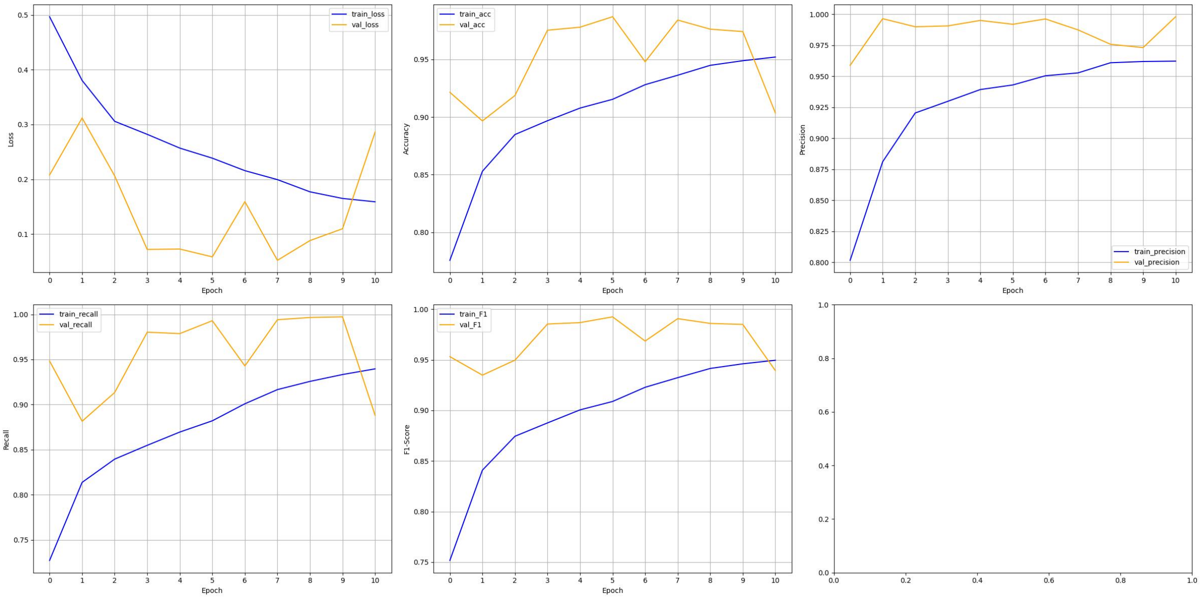
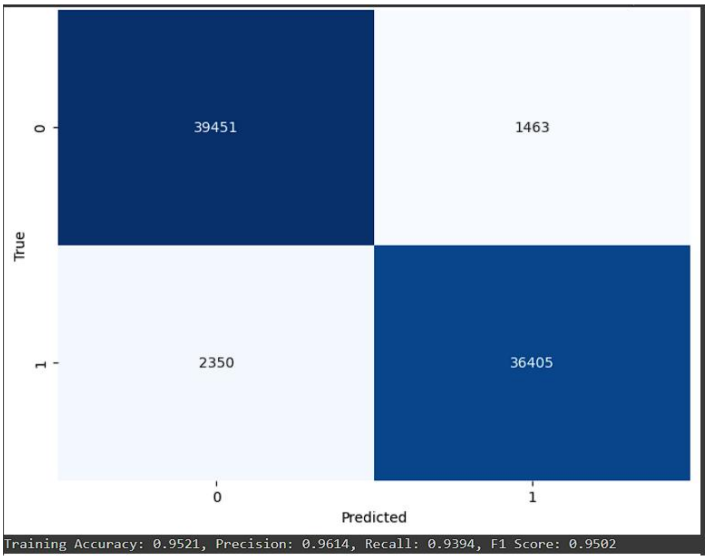
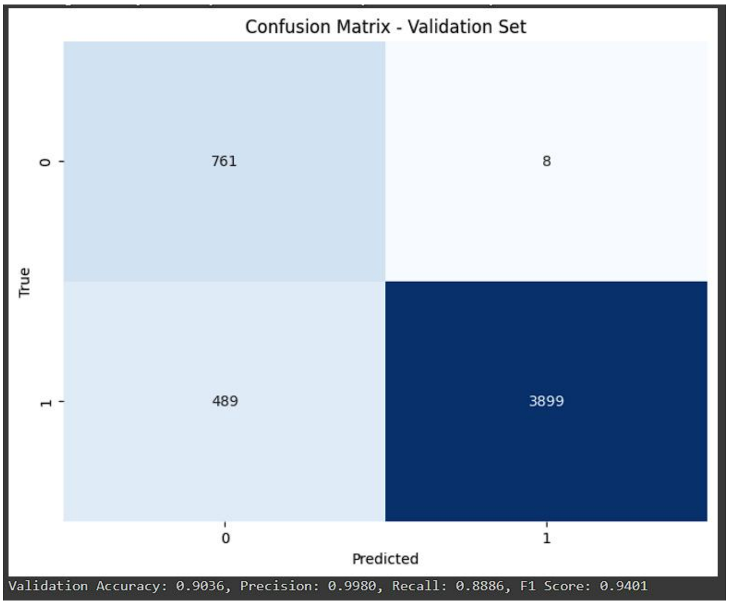


Figure 4.45 Graphs of performance metrics of inter-patient bi-directional LSTM classifier



```
Training Accuracy: 0.9521, Precision: 0.9614, Recall: 0.9394, F1 Score: 0.9502
```

Figure 4.46 Training confusion matrix of inter-patient bi-directional LSTM classifier



```
Validation Accuracy: 0.9036, Precision: 0.9980, Recall: 0.8886, F1 Score: 0.9401
```

Figure 4.47 Validation confusion matrix of inter-patient bi-directional LSTM classifier

4.8.1.5 Intra-Patient Uni-Directional GRU

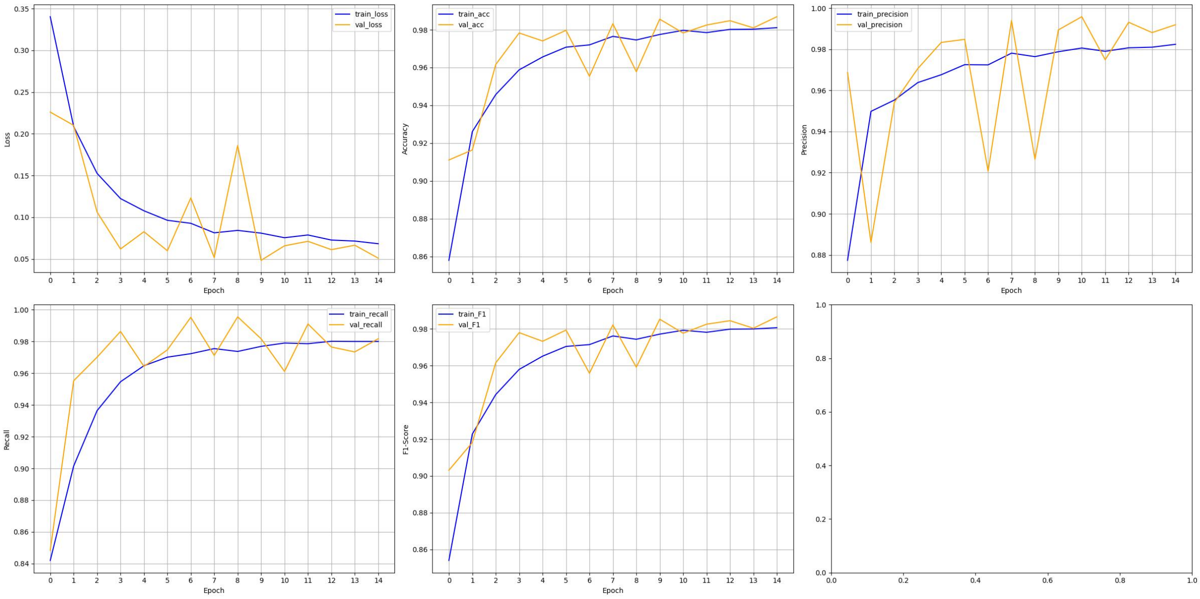
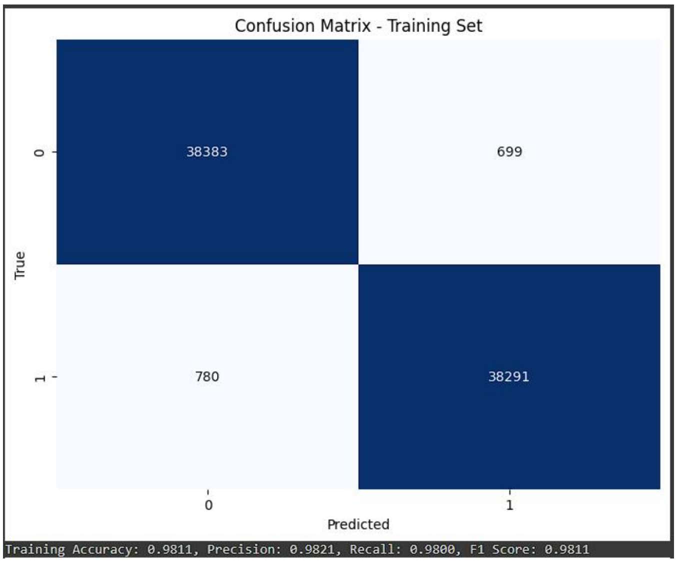
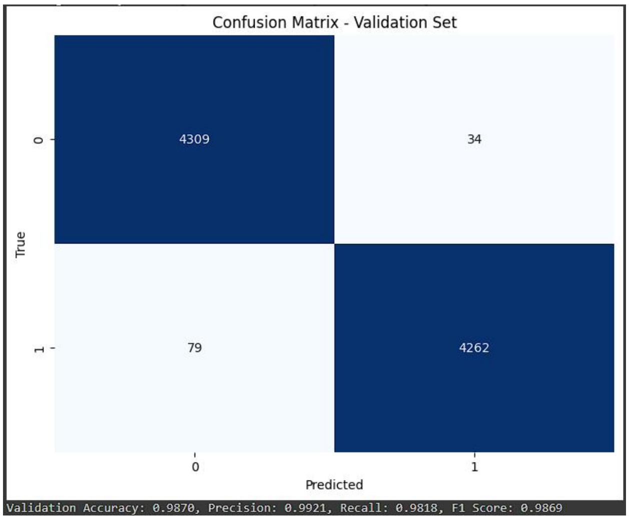


Figure 4.48 Graphs of performance metrics of intra-patient uni-directional GRU classifier



```
Training Accuracy: 0.9811, Precision: 0.9821, Recall: 0.9800, F1 Score: 0.9811
```

Figure 4.49 Training confusion matrix of intra-patient uni-directional GRU classifier



```
Validation Accuracy: 0.9870, Precision: 0.9921, Recall: 0.9818, F1 Score: 0.9869
```

Figure 4.50 Validation confusion matrix of intra-patient uni-directional GRU classifier

4.8.1.6 Inter-Patient Uni-Directional GRU

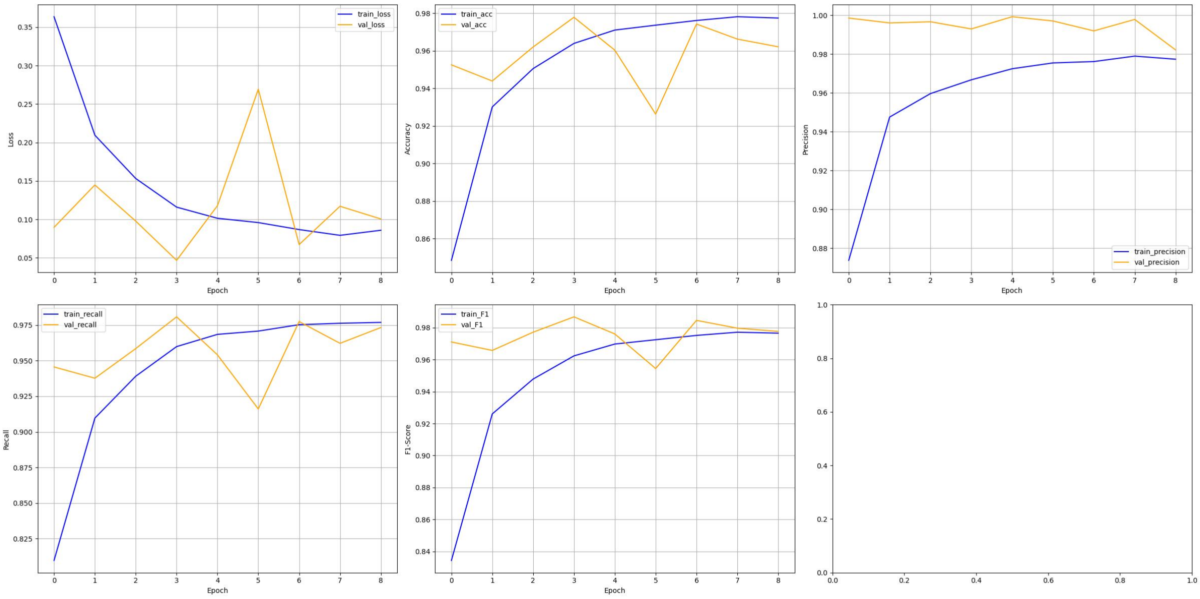
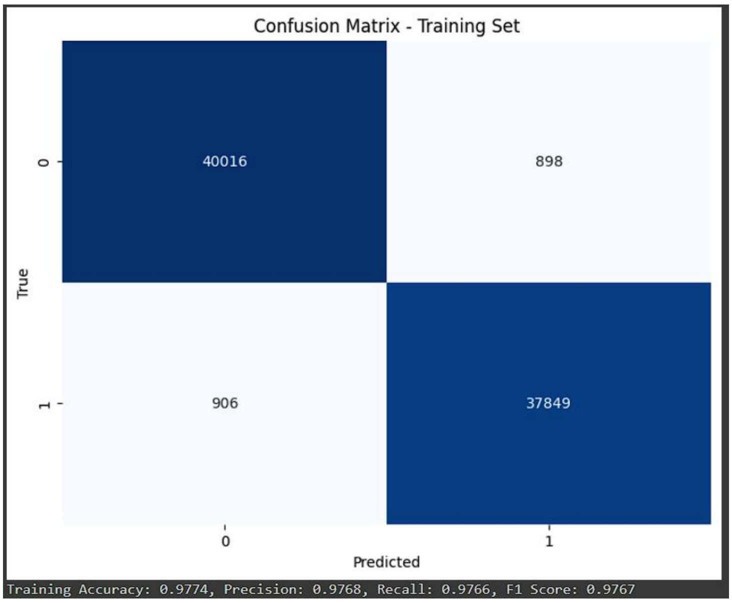
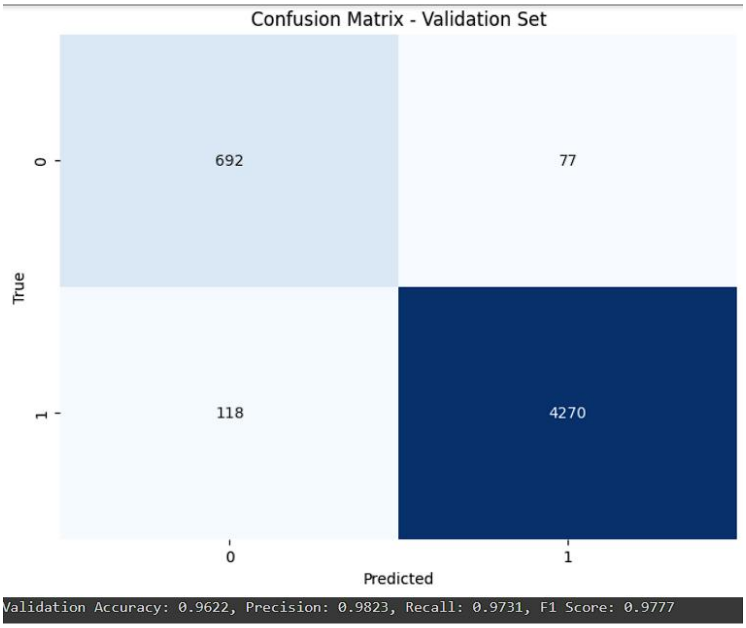


Figure 4.51 Graphs of performance metrics of inter-patient uni-directional GRU classifier



```
Training Accuracy: 0.9774, Precision: 0.9768, Recall: 0.9766, F1 Score: 0.9767
```

Figure 4.52 Training confusion matrix of inter-patient uni-directional GRU classifier



```
Validation Accuracy: 0.9622, Precision: 0.9823, Recall: 0.9731, F1 Score: 0.9777
```

Figure 4.53 Validation confusion matrix of inter-patient uni-directional GRU classifier

4.8.1.7 Intra-Patient Bi-Directional GRU

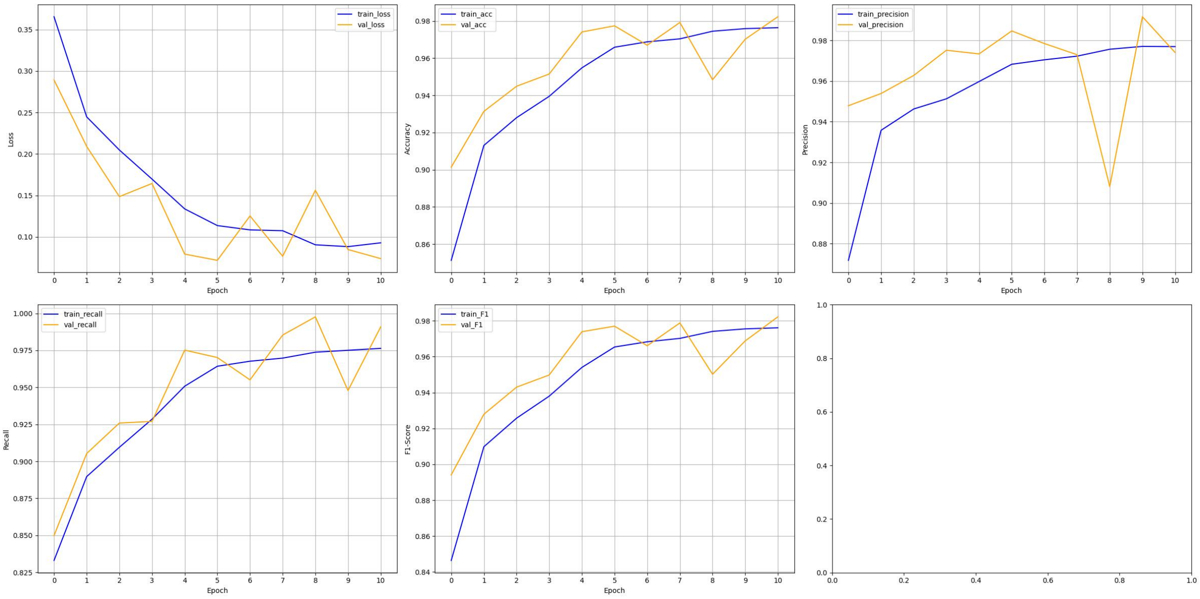
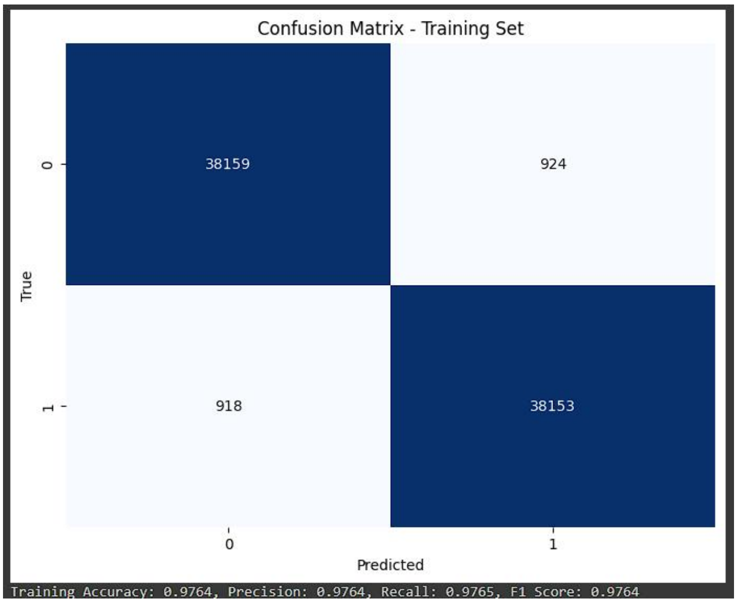
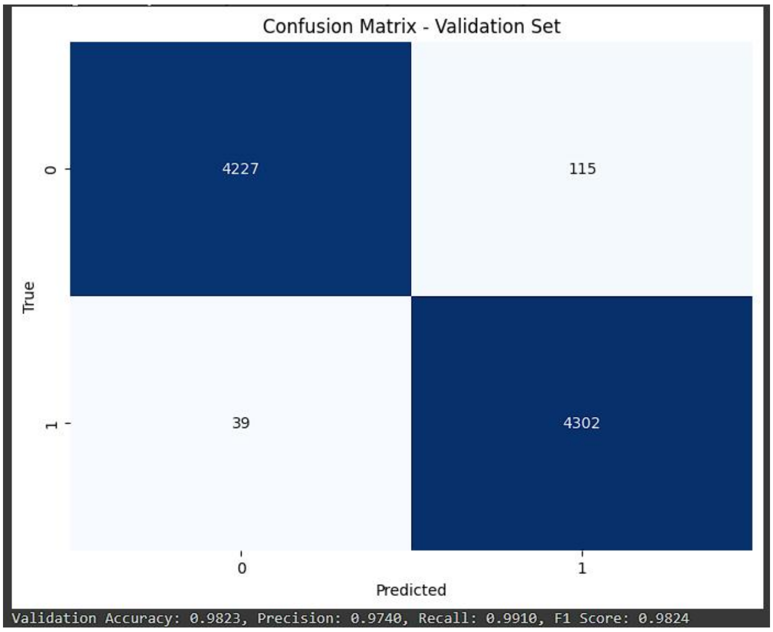


Figure 4.54 Graphs of performance metrics of intra-patient bi-directional GRU classifier



```
Training Accuracy: 0.9764, Precision: 0.9764, Recall: 0.9765, F1 Score: 0.9764
```

Figure 4.55 Validation confusion matrix of intra-patient bi-directional GRU classifier



```
Validation Accuracy: 0.9823, Precision: 0.9740, Recall: 0.9910, F1 Score: 0.9824
```

Figure 4.56 Validation confusion matrix of intra-patient bi-directional GRU classifier

4.8.1.8 Inter-Patient Bi-Directional GRU

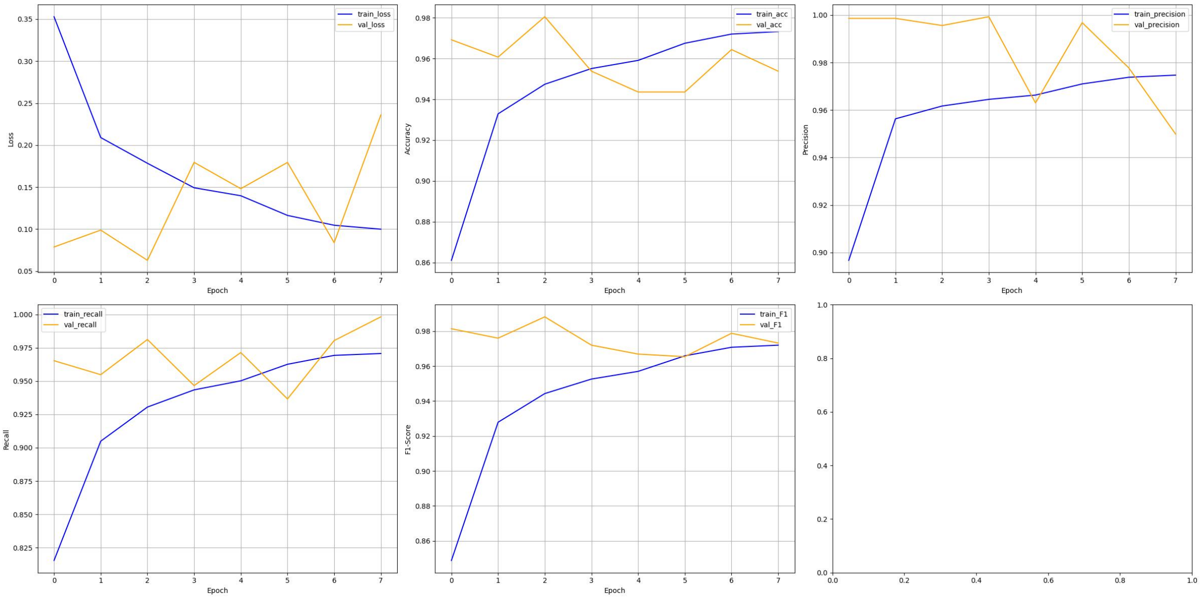
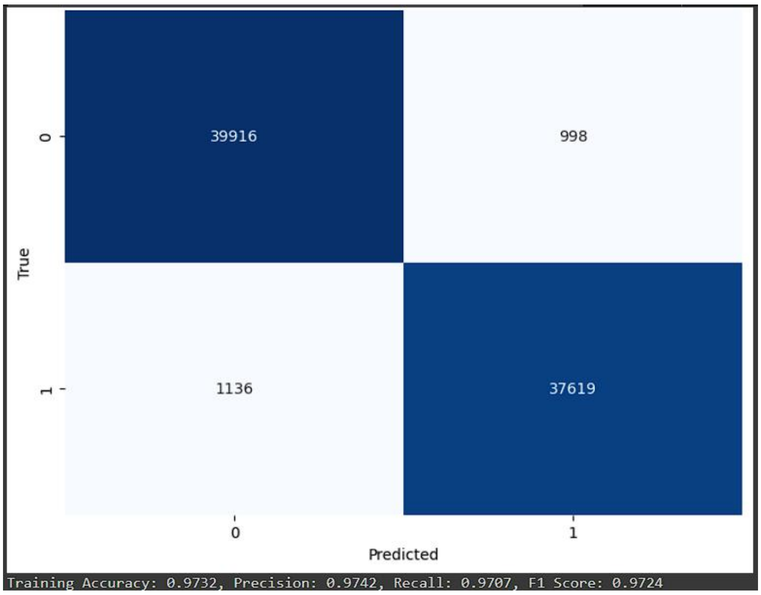
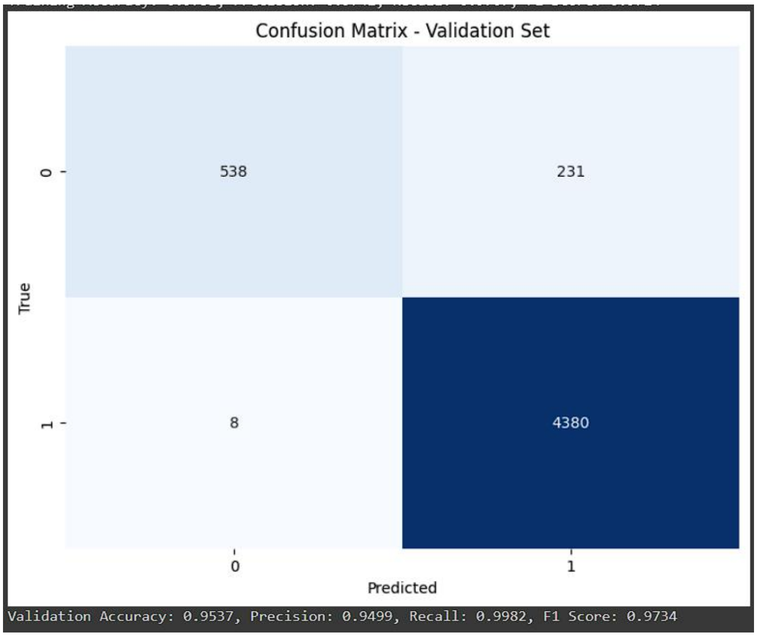


Figure 4.57 Graphs of performance metrics of inter-patient bi-directional GRU classifier



```
Training Accuracy: 0.9732, Precision: 0.9742, Recall: 0.9707, F1 Score: 0.9724
```

Figure 4.58 Validation confusion matrix of inter-patient bi-directional GRU classifier



```
Validation Accuracy: 0.9537, Precision: 0.9499, Recall: 0.9982, F1 Score: 0.9734
```

Figure 4.59 Validation confusion matrix of inter-patient bi-directional GRU classifier

4.7.9 Intra-Patient 1D-CNN

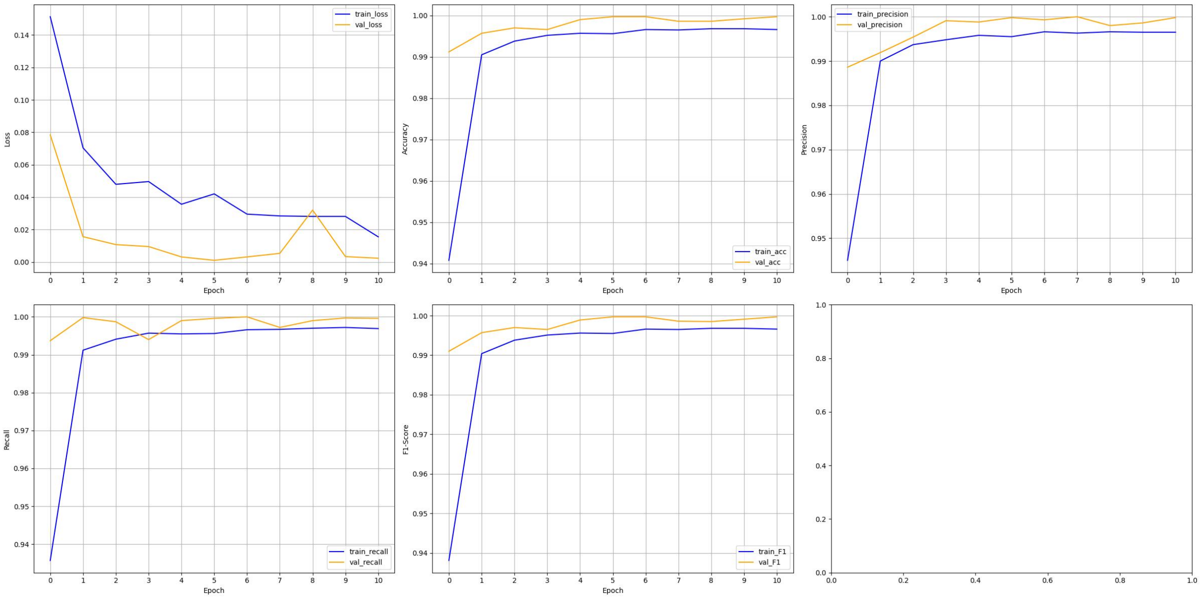
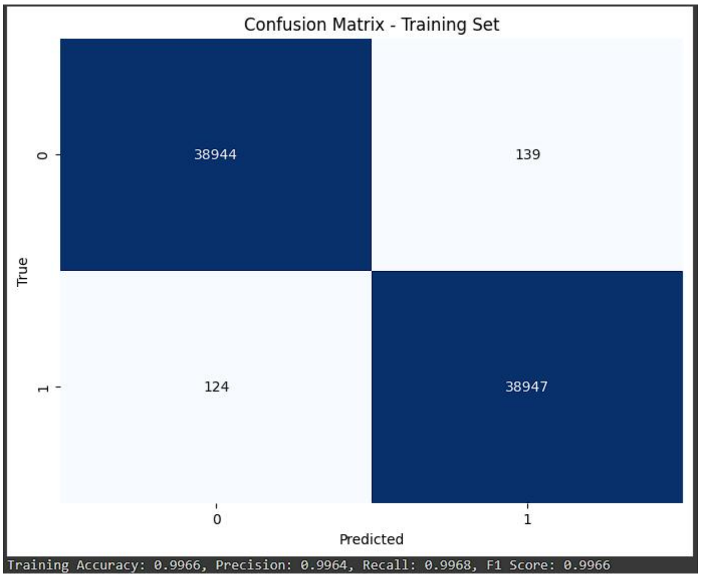
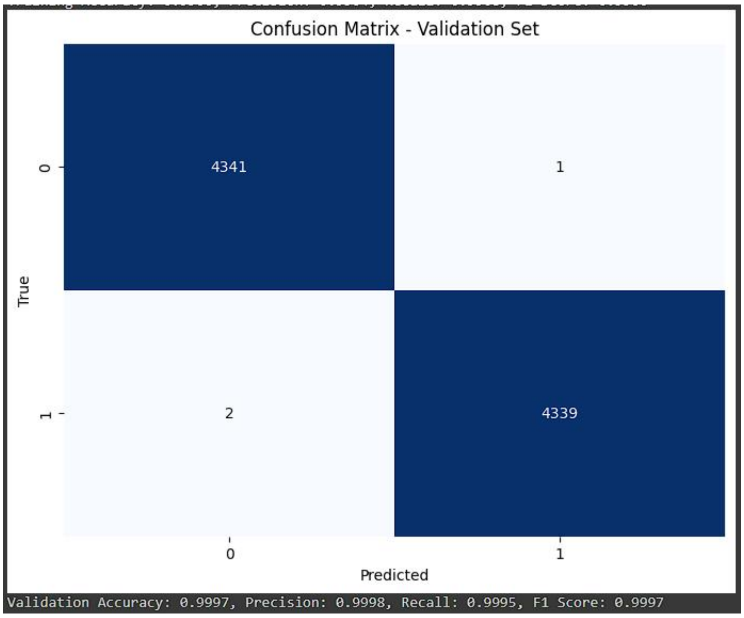


Figure 4.60 Graphs of performance metrics of intra-patient 1D-CNN classifier



```
Training Accuracy: 0.9966, Precision: 0.9964, Recall: 0.9968, F1 Score: 0.9966
```

Figure 4.61 Training confusion matrix of intra-patient 1D-CNN classifier



```
Validation Accuracy: 0.9997, Precision: 0.9998, Recall: 0.9995, F1 Score: 0.9997
```

Figure 4.62 Validation confusion matrix of intra-patient 1D-CNN classifier

4.7.10 Inter-Patient CNN Classifier

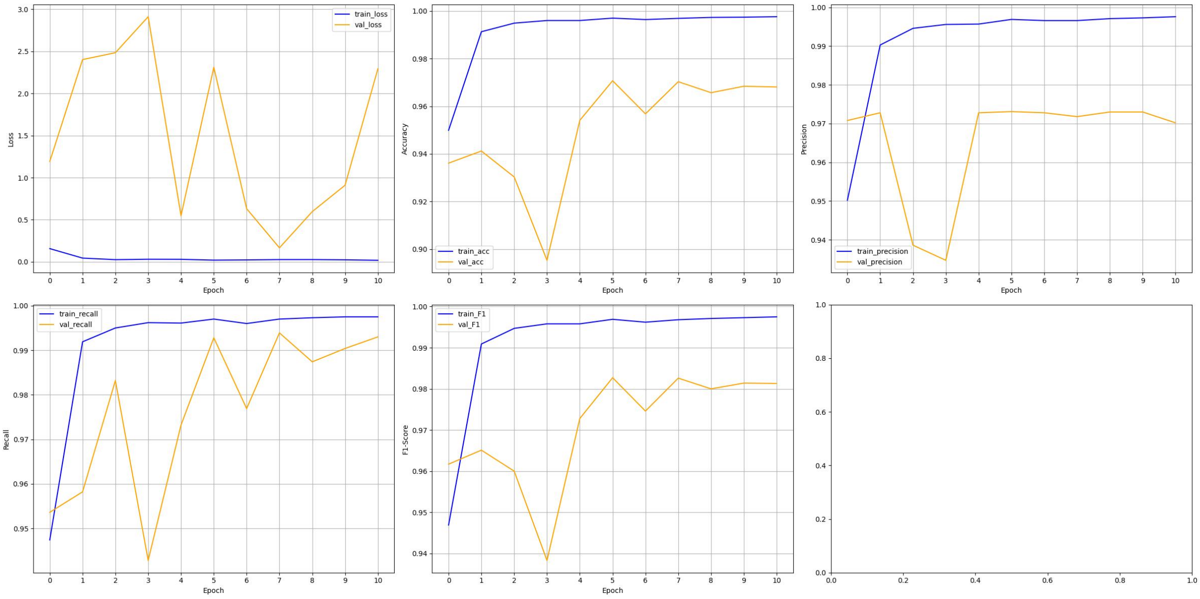
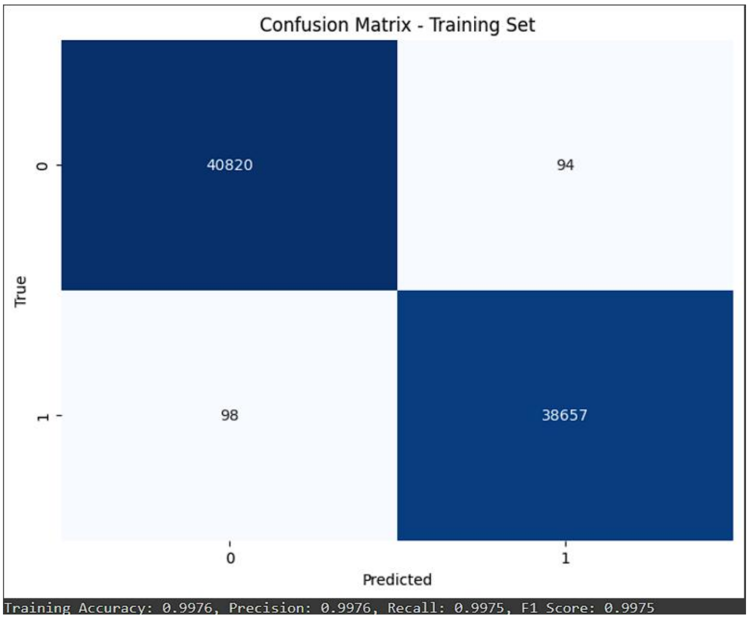
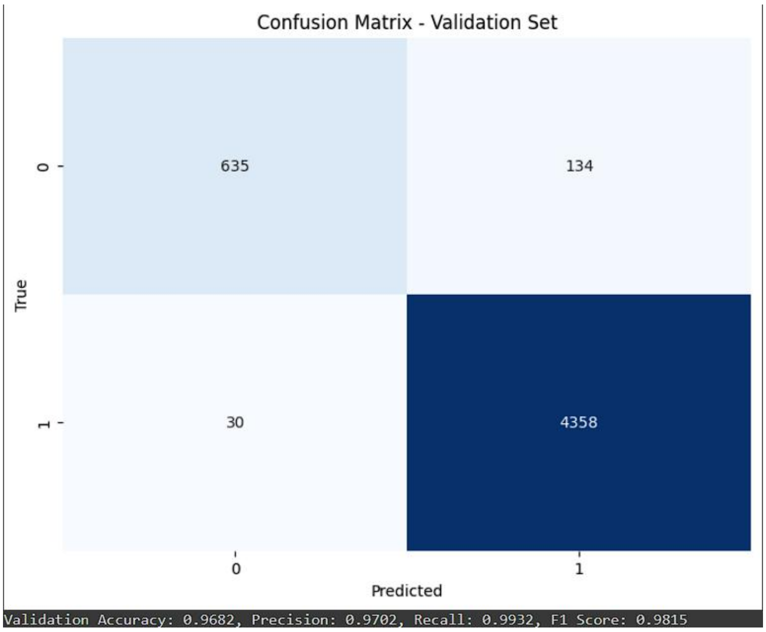


Figure 4.63 Graphs of performance metrics of inter-patient 1D-CNN classifier



```
Training Accuracy: 0.9976, Precision: 0.9976, Recall: 0.9975, F1 Score: 0.9975
```

Figure 4.64 Training confusion matrix of inter-patient 1D-CNN classifier



```
Validation Accuracy: 0.9682, Precision: 0.9702, Recall: 0.9932, F1 Score: 0.9815
```

Figure 4.65 Validation confusion matrix of inter-patient 1D-CNN classifier

4.7.11 Intra-Patient Transformer Classifier

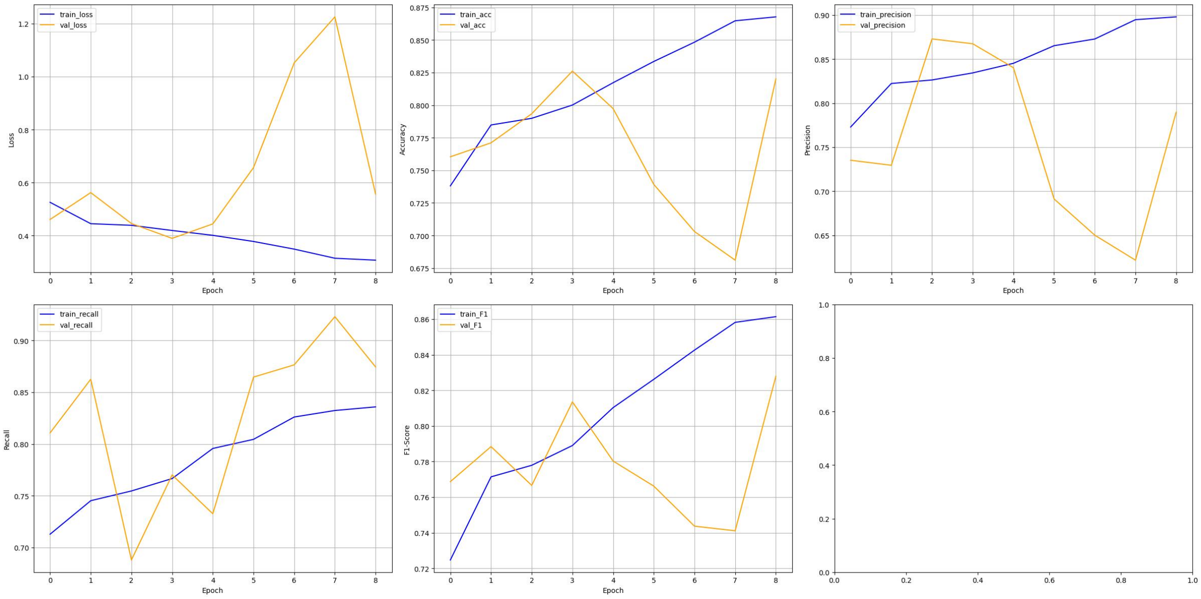
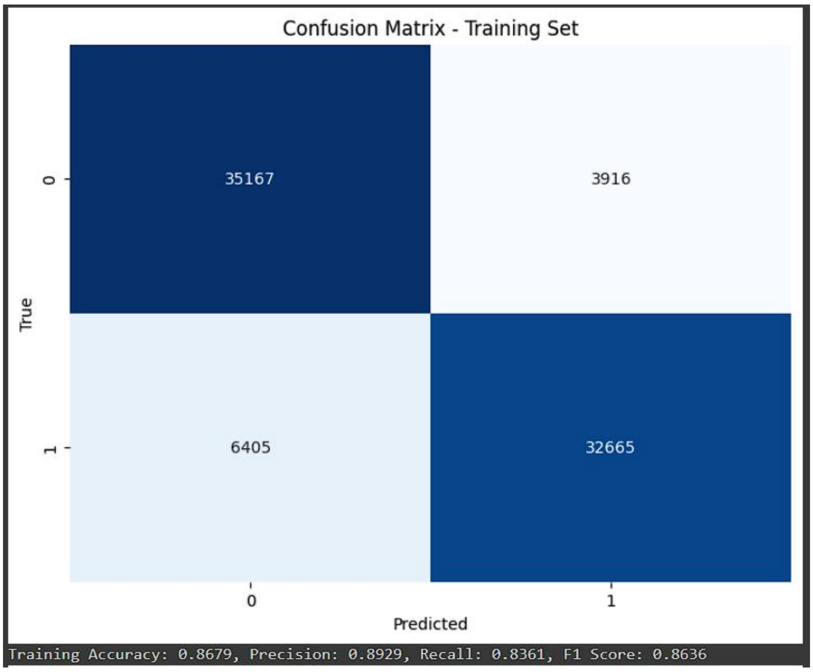
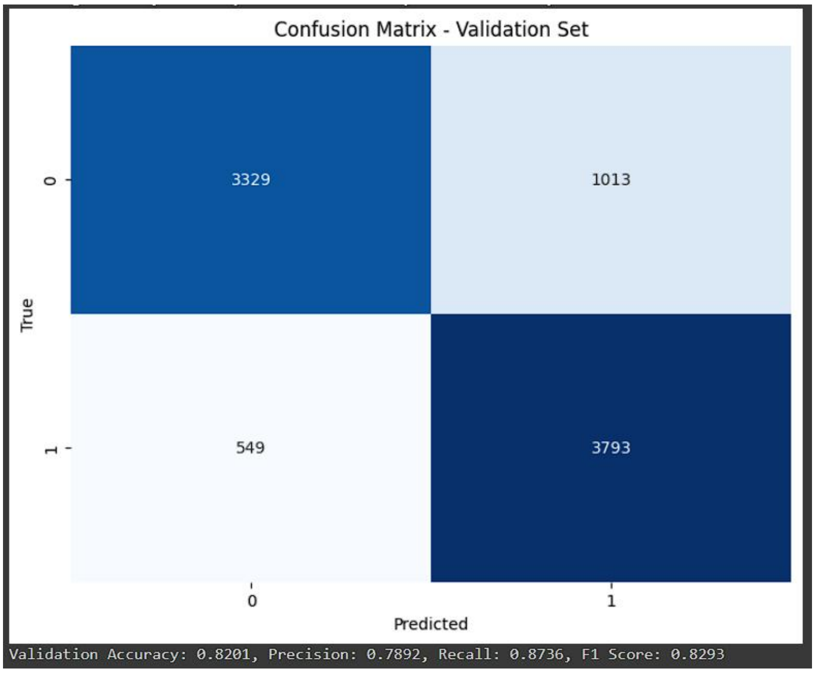


Figure 4.66 Graphs of performance metrics of intra-patient Transformer classifier



```
Training Accuracy: 0.8679, Precision: 0.8929, Recall: 0.8361, F1 Score: 0.8636
```

Figure 4.67 Training confusion matrix of intra-patient Transformer classifier



```
Validation Accuracy: 0.8201, Precision: 0.7892, Recall: 0.8736, F1 Score: 0.8293
```

Figure 4.68 Validation confusion matrix of intra-patient Transformer classifier

4.7.12 Inter-Patient Transformer Classifier

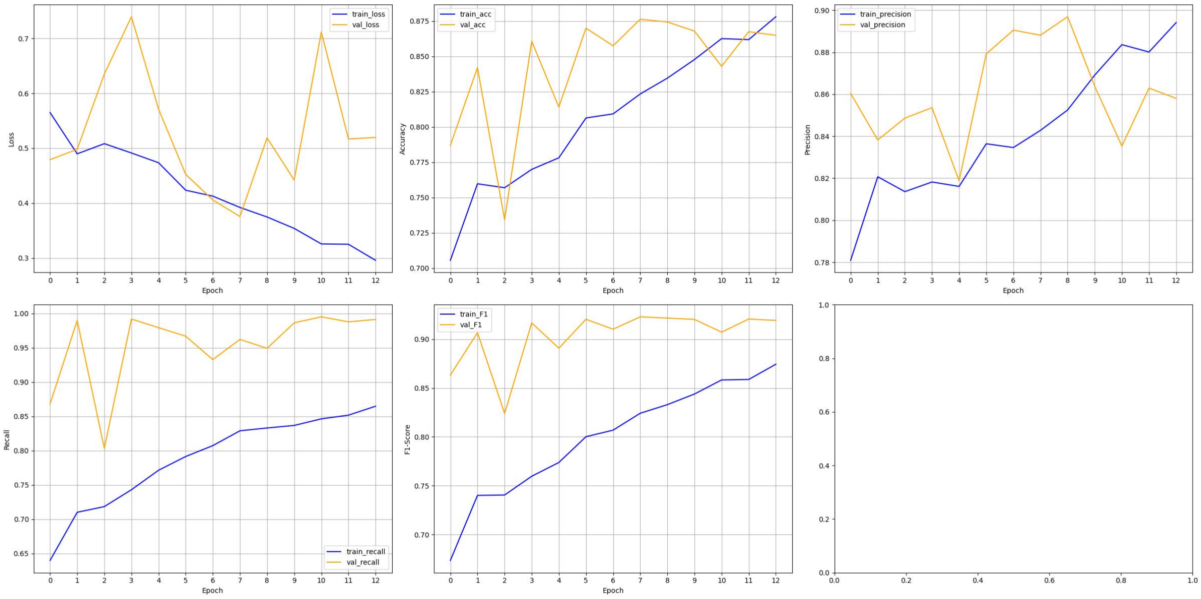
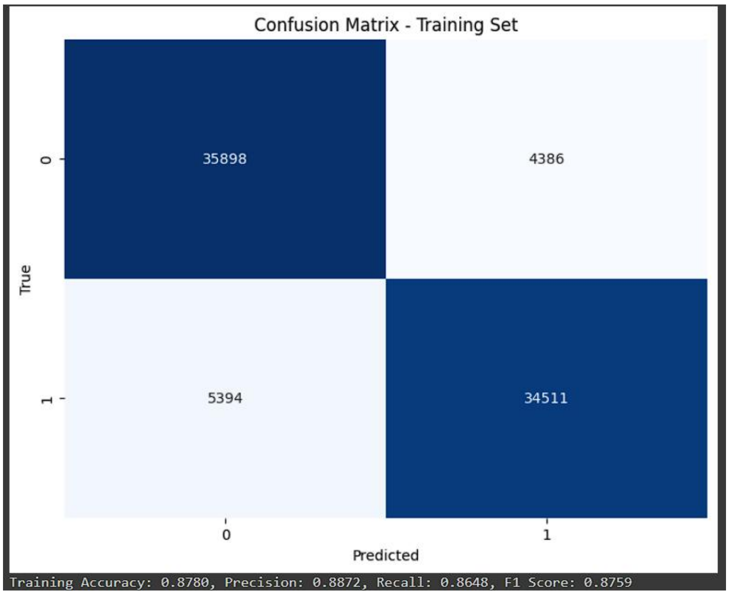
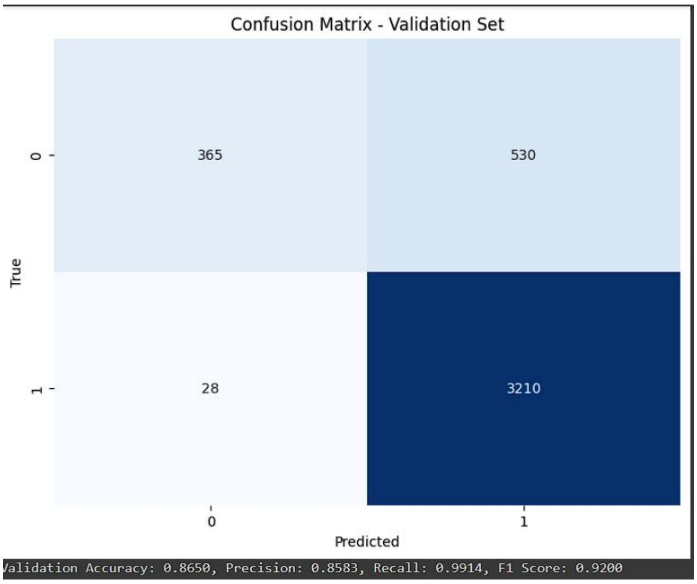


Figure 4.69 Graphs of performance metrics of inter-patient Transformer classifier



```
Training Accuracy: 0.8780, Precision: 0.8872, Recall: 0.8648, F1 Score: 0.8759
```

Figure 4.70 Training confusion matrix of inter-patient Transformer classifier



```
Validation Accuracy: 0.8650, Precision: 0.8583, Recall: 0.9914, F1 Score: 0.9200
```

Figure 4.71 Validation confusion matrix of inter-patient Transformer classifier

Chapter 5

SYSTEM EVALUATION AND DISCUSSION

5.1 Overview

Although all the classifiers trained by different data-split methods were able to obtain high performance in the training and validation phase, **Table 5.1**, the significant difference between the performance metrics of each pair of intra-patient classifier and inter-patient classifiers was observed. This is because of the data imbalance issue that exists in the dataset caused by the inter-patient splitting method. In inter-patient analysis, the patient folders were randomly assigned to only either the training set, validation set or testing set. The overfitting problem might appear if the data in the training set lack diversity or variability, therefore the classifier could not learn a more complex decision function to detect the presence of MI from an unusual or poor-quality MI ECG.

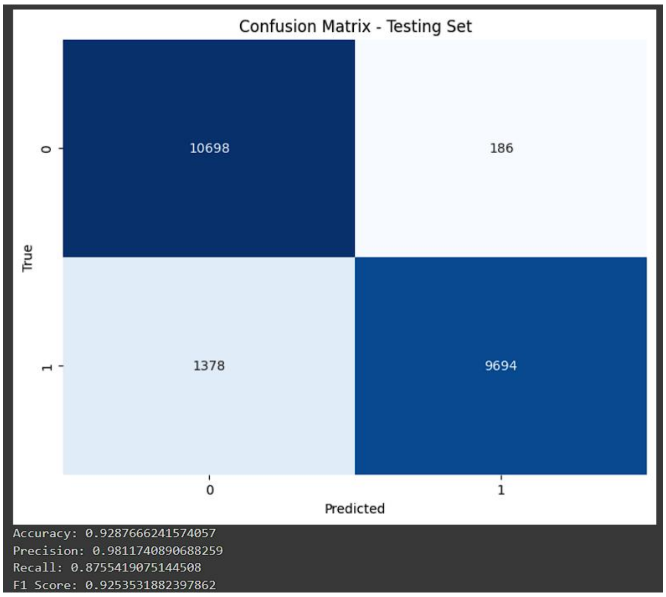
Table 5.1 Testing performance metrics of trained classifier in both types of analysis

Performance metrics / **Models**	**Accuracy**	**Precision**	**Sensitivity / Recall**	**F1 Score**
Intra Uni - LSTM	Test: 92.90%	Test: 97.81%	Test: 87.26%	Test: 0.9213
Inter Uni - LSTM	Test: 90.53%	Test: 94.11%	Test: 94.88%	Test: 0.9449
Intra Bi - LSTM	Test: 95.46%	Test: 96.48%	Test: 94.44%	Test: 0.9545
Inter Bi - LSTM	Test: 73. 40%	Test: 93.59%	Test: 73.99%	Test: 0.8264
Intra Uni – GRU	Test: 94.60%	Test: 98.71%	Test: 90.48%	Test: 0.9442

Inter Uni – GRU	Test: 86.65 %	Test: 94.88%	Test: 89.21%	Test: 0.9196
Intra Bi – GRU	Test: 95.43%	Test: 98.15%	Test: 92.68 %	Test: 0.9534
Inter Bi – GRU	Test: 85.85%	Test: 92.33%	Test: 91.02%	Test: 0.9167
Intra 1D – CNN	Test: 97.68%	Test: 99.92%	Test: 95.47%	Test: 0.9764
Inter 1D - CNN	Test: 85.82%	Test: 90.46%	Test: 93.27%	Test: 0.9184
Intra Transformer	Test: 82.28%	Test: 77.63%	Test: 91.12%	Test: 0.8384
Inter Transformer	Test: 91.15%	Test: 91.75%	Test: 98.52%	Test: 0.9501

5.2 LSTM

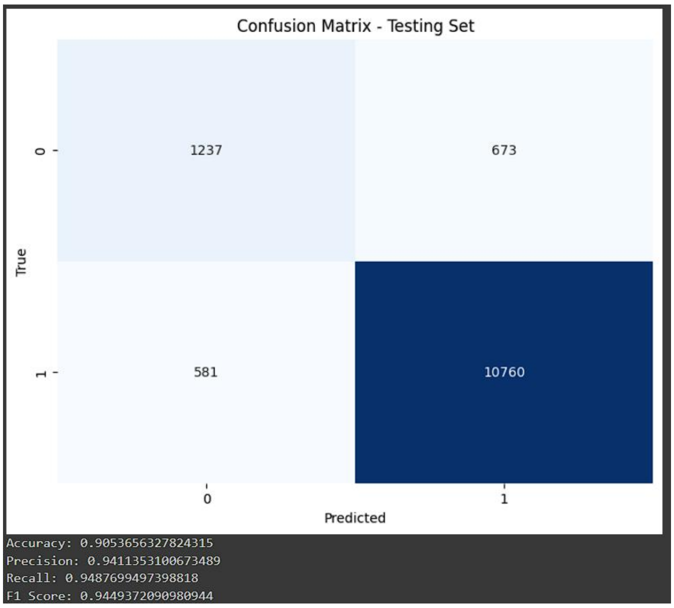
5.2.1 Intra-Patient Uni-Directional LSTM



```
Accuracy: 0.9287666241574057
Precision: 0.9811740890688259
Recall: 0.8755419075144508
F1 Score: 0.9253531882397862
```

Figure 5.1 Testing confusion matrix of intra-patient uni-directional LSTM classifier

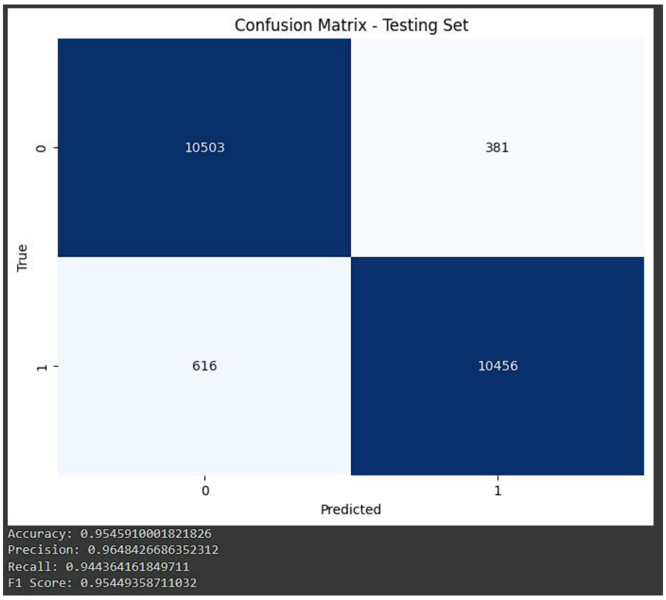
5.2.2 Inter-Patient Uni-Directional LSTM



```
Accuracy: 0.9053656327824315
Precision: 0.9411353100673489
Recall: 0.9487699497398818
F1 Score: 0.9449372090980944
```

Figure 5.2 Testing confusion matrix of inter-patient uni-directional LSTM classifier

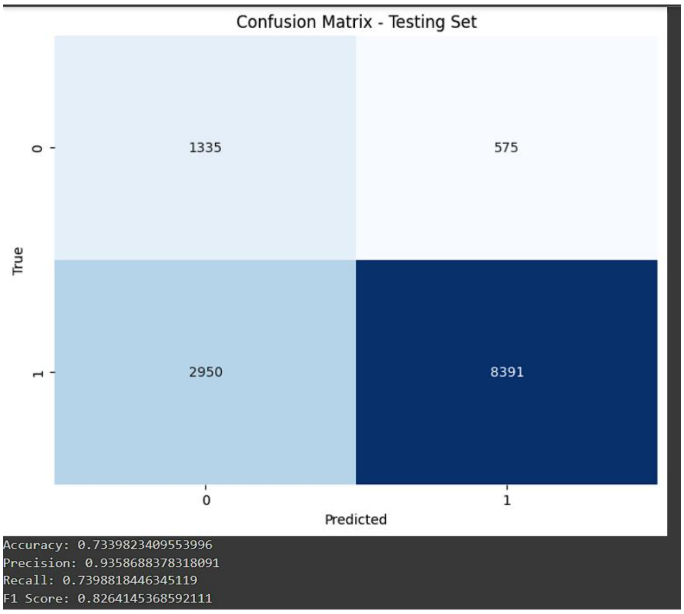
5.2.3 Intra-Patient Bi-Directional LSTM



```
Accuracy: 0.9545910001821826
Precision: 0.9648426686352312
Recall: 0.944364161849711
F1 Score: 0.95449358711032
```

Figure 5.3 Testing confusion matrix of intra-patient bi-directional LSTM classifier

5.2.4 Inter-Patient Bi-Directional LSTM

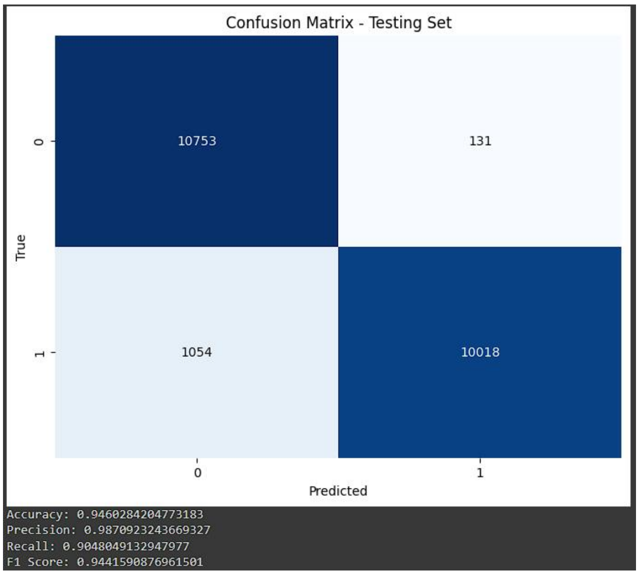


```
Accuracy: 0.7339823409553996
Precision: 0.9358688378318091
Recall: 0.7398818446345119
F1 Score: 0.8264145368592111
```

Figure 5.4 Testing confusion matrix of inter-patient bi-directional LSTM classifier

5.3 GRU

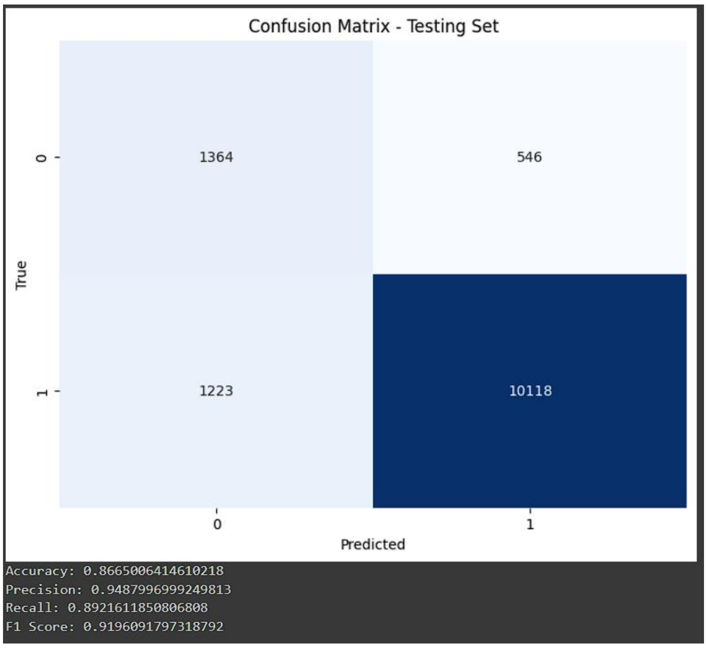
5.3.1 Intra-Patient Uni-Directional GRU



```
Accuracy: 0.9460284204773183
Precision: 0.9870923243669327
Recall: 0.9048049132947977
F1 Score: 0.9441590876961501
```

Figure 5.5 Testing confusion matrix of intra-patient uni-directional GRU classifier

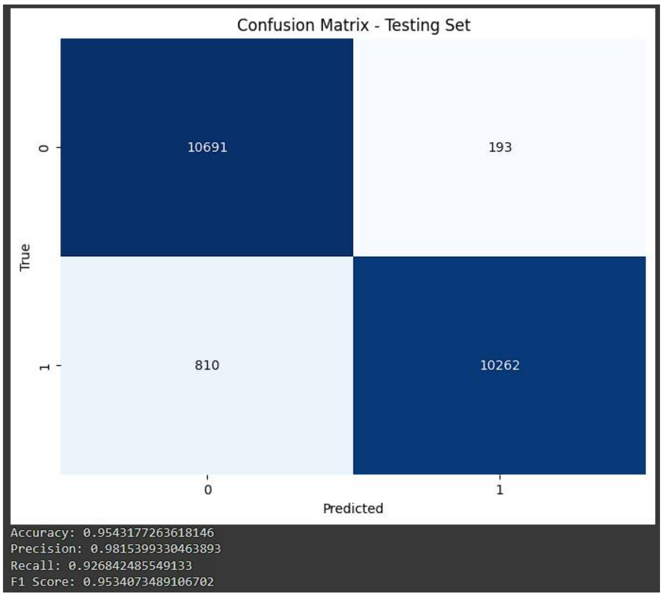
5.3.2 Inter-Patient Uni-Directional GRU



```
Accuracy: 0.8665006414610218
Precision: 0.9487996999249813
Recall: 0.8921611850806808
F1 Score: 0.9196091797318792
```

Figure 5.6 Testing confusion matrix of inter-patient uni-directional GRU classifier

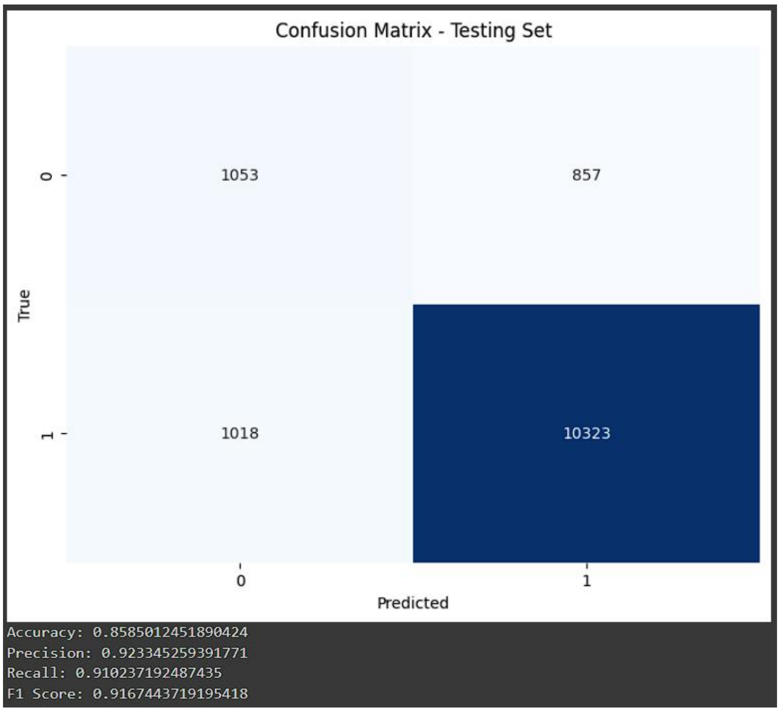
5.3.3 Intra-Patient Bi-Directional GRU



```
Accuracy: 0.9543177263618146
Precision: 0.9815399330463893
Recall: 0.926842485549133
F1 Score: 0.9534073489106702
```

Figure 5.7 Testing confusion matrix of intra-patient bi-directional GRU classifier

5.3.4 Inter-Patient Bi-Directional GRU

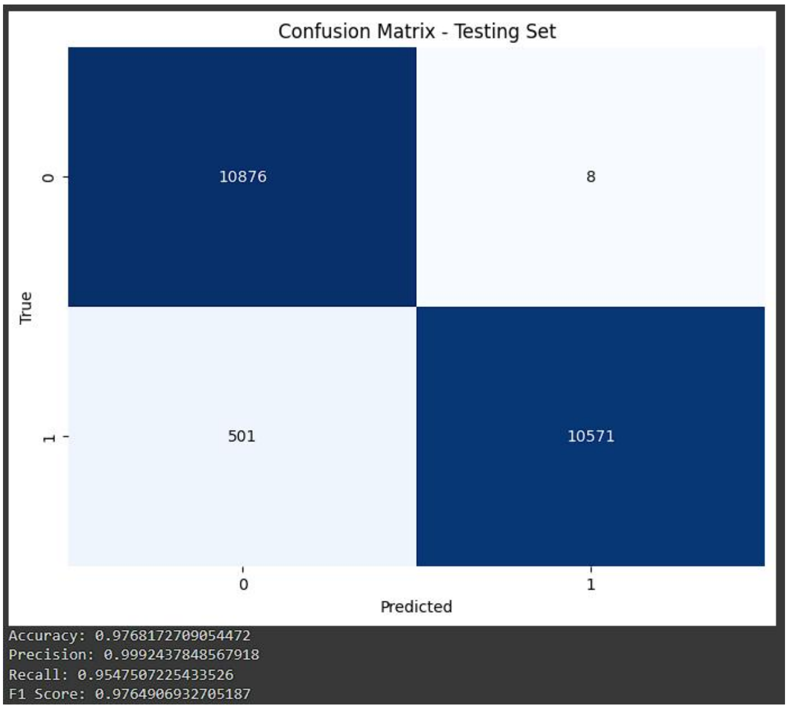


```
Accuracy: 0.8585012451890424
Precision: 0.923345259391771
Recall: 0.910237192487435
F1 Score: 0.9167443719195418
```

Figure 5.8 Testing confusion matrix of inter-patient bi-directional GRU classifier

5.4 1D-CNN

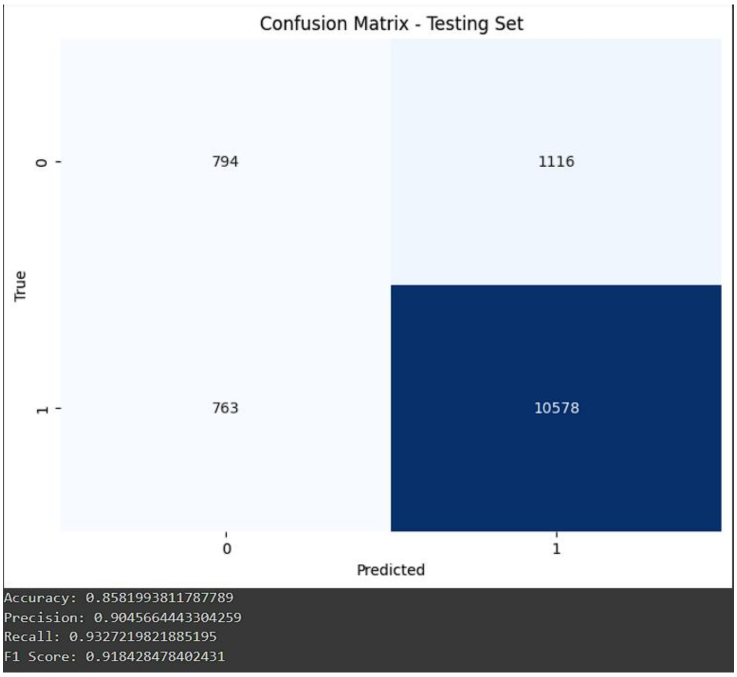
5.4.1 Intra-Patient 1D-CNN



```
Accuracy: 0.9768172709054472
Precision: 0.9992437848567918
Recall: 0.9547507225433526
F1 Score: 0.9764906932705187
```

Figure 5.9 Testing confusion matrix of intra-patient 1D-CNN classifier

5.4.2 Inter-Patient 1D-CNN

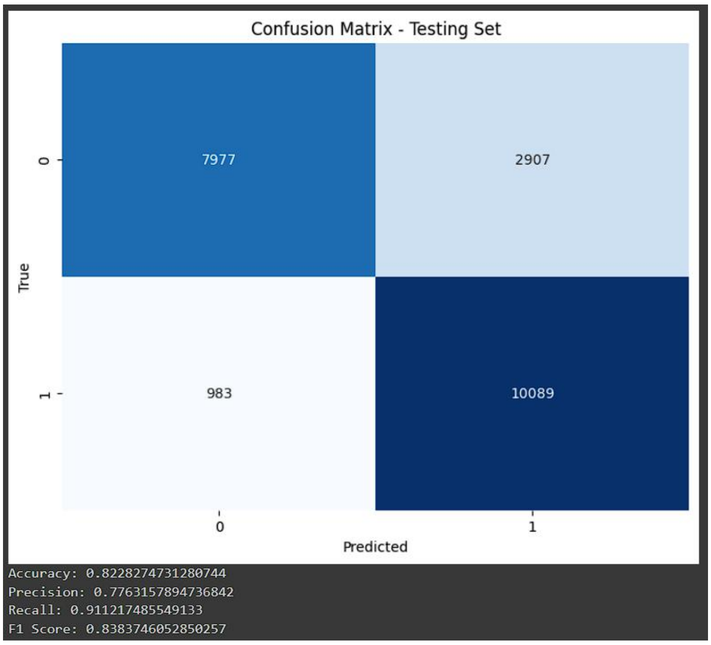


```
Accuracy: 0.8581993811787789
Precision: 0.9045664443304259
Recall: 0.9327219821885195
F1 Score: 0.918428478402431
```

Figure 5.10 Testing confusion matrix of inter-patient 1D-CNN classifier

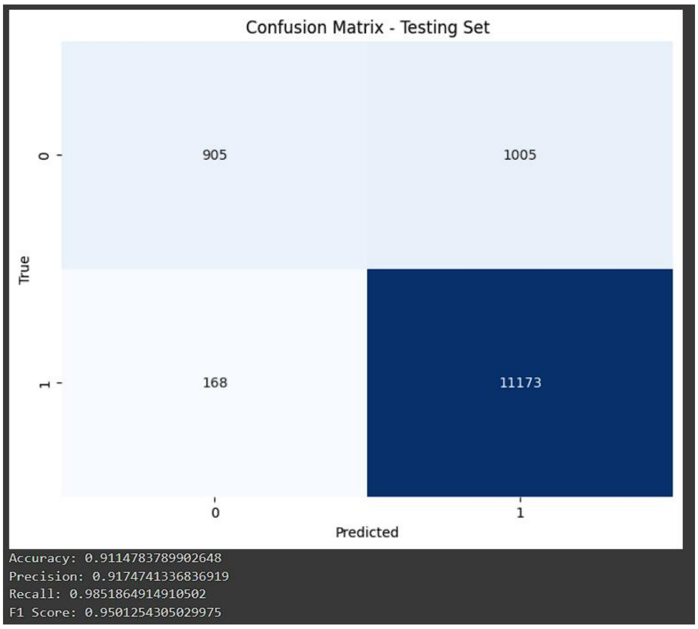
5.5 Transformer

5.5.1 Intra-Patient Transformer Classifier



```
Accuracy: 0.8228274731280744
Precision: 0.7763157894736842
Recall: 0.911217485549133
F1 Score: 0.8383746052850257
```

5.5.2 Inter-Patient Transformer Classifier



```
Accuracy: 0.9114783789902648
Precision: 0.9174741336836919
Recall: 0.9851864914910502
F1 Score: 0.9501254305029975
```

Figure 5.12 Testing confusion matrix of inter-patient Transformer classifier

5.3 Challenges

Although all the classifiers trained by different data-split methods were able to obtain high performance in the training and validation phase, **Table 5.1**, the significant difference between the performance metrics of each pair of intra-patient classifier and inter-patient classifiers was observed. This is because of the data imbalance issue that exists in the dataset caused by the inter-patient splitting method. In inter-patient analysis, the patient folders were randomly assigned to only either the training set, validation set or testing set. The overfitting problem might appear if the data in the training set lack diversity or variability, therefore the classifier could not learn a more complex decision function to detect the presence of MI from an unseen, unusual or poor-quality MI ECG. Plus, the number of patients in the dataset was only 200 and only 48 of them are healthy subjects. Therefore, one of the challenges in this project is to get a balanced and large dataset with high signal quality.

Secondly, the options for data augmentation that can be applied to the signal are limited as the voltage value on each time step carries information.

Thirdly, to prove the complexity of the classifier network is enough to capture necessary features from the ECGs, a lot of fine-tuning/experiments are required to observe the performance metrics.

5.4 Objectives Evaluation

The first objective was achieved by training the proposed designed networks, LSTM, GRU, 1D-CNN and transformer with single-lead ECG data through an inter-patient analysis. The second objective was achieved by visualizing the performance metrics including accuracy, precision, sensitivity and F1-score of each type of classifier through the graphs and the confusion matrix. The difference in performance metrics was also explored by developing an intra-patient classifier for each type of classifier with the same hyperparameter setting to make a comparison. The effect of different data-split methods is serious, especially in the dataset used in this project. Without learning leaked heartbeats from the same patients, the classifier should not have such high performance.

Chapter 6

CONCLUSION AND LIMITATIONS

6.1 Conclusion

By implementing the patient-split technique, the decrement in performance metrics obtained by the models could be expected when compared to the previous studies that implemented intra-patient analysis or did not mention how they split the data. To increase the reliability of the performance shown by the models, it must be trained in a way that can represent or simulate the real-world environment otherwise its performance can be questioned. On top of this, it is dangerous to society when a classifier with misleadingly high testing performance especially in terms of sensitivity is applied by the public as misclassification can happen. In conclusion, four types of classifiers are proposed in this project to observe their ability to detect the presence of MI in an inter-patient analysis, by taking the ECGs as the input. Uni-directional LSTM and Transformer classifiers are able to get an accuracy above 90% and yet also keep other performance metrics above 90%. GRU classifiers were the weakest classifiers explored throughout this project. While in both intra-patient and inter-patient analysis, 1D-CNN has the highest performance metrics in training and validation phases and it can converge in through a small number of epochs. Thus, it has the highest potential to be developed as the most robust and accurate classifier to detect MI.

To counter the growing population size and unhealthy diet habits spread across the global society, the growing demand for manual human interpretation of ECG caused too much workload to the medical field, especially the cardiologists, therefore a deep-learning-based classifier takes place to monitor the occurrence of MI among the public simultaneously.

6.2 Future Works

A few future works can be expected to explore the potential of each classifier, which are more fine-tuning and re-train the networks using a large balanced dataset to allow them to learn more features from both MI patients and normal subjects.

Secondly, another dataset with the same sampling rate is required to test the ability of generalization of the classifier to increase its reliability. Unfortunately, another open-source database, PTB-XL [45] is applying a different sampling rate (500Hz frequency) while the trained classifiers in this project were trained by ECGs with a 1000Hz sampling rate. That means half of the information the classifier requires to make the decision is lost.

REFERENCES

[1]
World Health Organization, “Cardiovascular Diseases,” *World Health Organization*, 2023. https://www.who.int/health-topics/cardiovascular-diseases#tab=tab_1
[2]
Ministry of Health Malaysia, “Institute for Public Health - NHMS 2019,” *iku.gov.my*, May 30, 2020. https://iku.gov.my/nhms-2019
[3]
“Cardiovascular Disease (CVD) Position Paper,” *Cardiovascular Disease (CVD) Position Paper*. https://www.malaysianheart.org/publication/clinical-practice-guidelines/p/cardiovascular-disease-cvd-position-paper (accessed Apr. 15, 2024).
[4]
DOSM, “Department of Statistics Malaysia,” *Dosm.gov.my*, 2023. https://www.dosm.gov.my/portal-main/release-content/statistics-on-causes-of-death-malaysia-2023
[5]
Cleveland Clinic, “Cardiovascular Disease: Symptoms, Types, Causes, Management & Prevention,” *Cleveland Clinic*, May 10, 2021. https://my.clevelandclinic.org/health/diseases/21493-cardiovascular-disease
[6]
World Health Organization, “Cardiovascular diseases (CVDs),” *World Health Organization*, Jun. 11, 2021. https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)
[7]
Centers for Disease Control and Prevention, “Coronary Artery disease: Causes, Diagonosis & Prevention,” *Centers for Disease Control and Prevention*, Jul. 19, 2021. https://www.cdc.gov/heartdisease/coronary_ad.htm
[8]
N. Ojha and A. S. Dhamoon, “Myocardial Infarction,” *PubMed*, 2023. https://www.ncbi.nlm.nih.gov/books/NBK537076/#:~:text=Myocardial%20infarction%20(MI)%2C%20colloquially
[9]
Ranya N. Sweis and Arif Jivan, “Overview of Coronary Artery Disease - Cardiovascular Disorders,” *MSD Manual Professional Edition*. https://www.msdmanuals.com/professional/cardiovascular-disorders/coronary-artery-disease/overview-of-coronary-artery-disease#Treatment_v934055 (accessed Apr. 18, 2024).
[10]
Ranya N. Sweis and Arif Jivan, “Acute Myocardial Infarction (MI),” *MSD Manual Professional Edition*, 2018. https://www.msdmanuals.com/professional/cardiovascular-disorders/coronary-artery-disease/acute-myocardial-infarction-mi (accessed Feb. 2024).
[11]
M. N. Berman and A. Bhardwaj, “Physiology, Left Ventricular Function,” *PubMed*, Sep. 19, 2022. https://www.ncbi.nlm.nih.gov/books/NBK541098/
[12]
Cleveland Clinic, “What is a STEMI Heart Attack?,” *Cleveland Clinic*, Nov. 15, 2021. https://my.clevelandclinic.org/health/diseases/22068-stemi-heart-attack
[13]

"Portal Rasmi Kementerian Kesihatan Malaysia," *www.moh.gov.my*. https://www.moh.gov.my/index.php/pages/view/133

[14]
O. J. Mechanic, S. A. Grossman, and M. Gavin, "Acute Myocardial Infarction," *National Library of Medicine*, Sep. 03, 2023. https://www.ncbi.nlm.nih.gov/books/NBK459269/

[15]
H. Shiomi *et al.*, "Association of onset to balloon and door to balloon time with long term clinical outcome in patients with ST elevation acute myocardial infarction having primary percutaneous coronary intervention: observational study," *BMJ*, vol. 344, no. may23 1, pp. e3257–e3257, May 2012, doi: https://doi.org/10.1136/bmj.e3257.

[16]
L. Wu *et al.*, "Deep Learning Networks Accurately Detect ST-Segment Elevation Myocardial Infarction and Culprit Vessel," *Frontiers in Cardiovascular Medicine*, vol. 9, Mar. 2022, doi: https://doi.org/10.3389/fcvm.2022.797207.

[17]
John Hopkins Medicine, "Electrocardiogram," *www.hopkinsmedicine.org*, Aug. 08, 2021. https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electrocardiogram#:~:text=What%20is%20an%20electrocardiogram%3F

[18]
Y. Sattar and L. Chhabra, "Electrocardiogram," *National Library of Medicine*, Jun. 05, 2023. https://www.ncbi.nlm.nih.gov/books/NBK549803/

[19]
S. Mehta *et al.*, "Global Challenges and Solutions," *Interventional Cardiology Clinics*, vol. 5, no. 4, pp. 569–581, Oct. 2016, doi: https://doi.org/10.1016/j.iccl.2016.06.013.

[20]
P. Xiong, S. M.-Y. Lee, and G. Chan, "Deep Learning for Detecting and Locating Myocardial Infarction by Electrocardiogram: A Literature Review," *Frontiers in Cardiovascular Medicine*, vol. 9, Mar. 2022, doi: https://doi.org/10.3389/fcvm.2022.860032.

[21]
X. Liu, H. Wang, Z. Li, and L. Qin, "Deep learning in ECG diagnosis: A review," *Knowledge-Based Systems*, vol. 227, no. 227, p. 107187, Sep. 2021, doi: https://doi.org/10.1016/j.knosys.2021.107187.

[22]
S. Hong, Y. Zhou, J. Shang, C. Xiao, and J. Sun, "Opportunities and challenges of deep learning methods for electrocardiogram data: A systematic review," *Computers in Biology and Medicine*, vol. 122, p. 103801, Jul. 2020, doi: https://doi.org/10.1016/j.compbiomed.2020.103801.

[23]
H. Martin, W. Izquierdo, M. Cabrerizo, A. Cabrera, and M. Adjouadi, "Near real-time single-beat myocardial infarction detection from single-lead electrocardiogram using Long Short-Term Memory Neural Network," *Biomedical Signal Processing and Control*, vol. 68, p. 102683, Jul. 2021, doi: https://doi.org/10.1016/j.bspc.2021.102683.

[24]
A. L. Goldberger *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet," *Circulation*, vol. 101, no. 23, Jun. 2000, doi: https://doi.org/10.1161/01.cir.101.23.e215.

[25]
C. M. Gibson *et al.*, "Evolution of single-lead ECG for STEMI detection using a deep learning approach," *International Journal of Cardiology*, Nov. 2021, doi: https://doi.org/10.1016/j.ijcard.2021.11.039.

[26]
Y. Zhao *et al.*, “Early detection of ST-segment elevated myocardial infarction by artificial intelligence with 12-lead electrocardiogram,” *International Journal of Cardiology*, vol. 317, pp. 223–230, Oct. 2020, doi: https://doi.org/10.1016/j.ijcard.2020.04.089.
[27]
S. Kavak, X.-D. Chiu, S.-J. Yen, and M. Y.-C. Chen, “Application of CNN for Detection and Localization of STEMI using 12-Lead ECG Images,” *IEEE Access*, pp. 1–1, 2022, doi: https://doi.org/10.1109/access.2022.3165966.
[28]
H. W. Lui and K. L. Chow, “Multiclass classification of myocardial infarction with convolutional and recurrent neural networks for portable ECG devices,” *Informatics in Medicine Unlocked*, vol. 13, pp. 26–33, 2018, doi: https://doi.org/10.1016/j.imu.2018.08.002.
[29]
X. Zhang, R. Li, Q. Hu, B. Zhou, and Z. Wang, “A New Automatic Approach to Distinguish Myocardial Infarction Based on LSTM,” *2019 8th International Symposium on Next Generation Electronics (ISNE)*, Oct. 2019, doi: https://doi.org/10.1109/isne.2019.8896550.
[30]
A. Darmawahyuni, S. Nurmaini, and Sukemi, “(PDF) Deep Learning with Long Short-Term Memory for Enhancement Myocardial Infarction Classification,” *www.researchgate.net*, 2019. https://www.researchgate.net/publication/337643008_Deep_Learning_with_Long_Short-Term_Memory_for_Enhancement_Myocardial_Infarction_Classification
[31]
U. B. Baloglu, M. Talo, O. Yildirim, R. S. Tan, and U. R. Acharya, “Classification of myocardial infarction with multi-lead ECG signals and deep CNN,” *Pattern Recognition Letters*, vol. 122, pp. 23–30, May 2019, doi: https://doi.org/10.1016/j.patrec.2019.02.016.
[32]
U. R. Acharya, H. Fujita, S. L. Oh, Y. Hagiwara, J. H. Tan, and M. Adam, “Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals,” *Information Sciences*, vol. 415–416, pp. 190–198, Nov. 2017, doi: https://doi.org/10.1016/j.ins.2017.06.027.
[33]
A. C. V. Maggio, M. P. Bonomini, E. L. Leber, and P. D. Arini, “Quantification of Ventricular Repolarization Dispersion Using Digital Processing of the Surface ECG,” *www.intechopen.com*, Jan. 25, 2012. https://www.intechopen.com/chapters/27012 (accessed Apr. 18, 2024).
[34]
A. Bajaj, “Performance Metrics in Machine Learning [Complete Guide],” *neptune.ai*, May 02, 2021. https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide
[35]
R. Bousseljot, D. Kreiseler, and A. Schnabel, “Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet,” *Biomedizinische Technik/Biomedical Engineering*, vol. 40, no. 1, pp. 317–318, 1995, doi: https://doi.org/10.1515/bmte.1995.40.s1.317.
[36]
Abraham. Savitzky and M. J. E. Golay, “Smoothing and Differentiation of Data by Simplified Least Squares Procedures.,” *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, Jul. 1964, doi: https://doi.org/10.1021/ac60214a047.
[37]

J. Pan and W. J. Tompkins, "A Real-Time QRS Detection Algorithm," *IEEE Transactions on Biomedical Engineering*, vol. BME-32, no. 3, pp. 230–236, Mar. 1985, doi: https://doi.org/10.1109/tbme.1985.325532.
[38]
M. Ahmad, P. Mehta, A. K. R. Reddivari, and S. Mungee, "Percutaneous coronary intervention," *PubMed*, 2020. https://www.ncbi.nlm.nih.gov/books/NBK556123/
[39]
"LSTM — PyTorch 1.8.1 documentation," *pytorch.org*. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html
[40]
"GRU — PyTorch 2.1 documentation," *pytorch.org*. https://pytorch.org/docs/stable/generated/torch.nn.GRU.html
[41]
"Conv1d — PyTorch 2.0 documentation," *pytorch.org*. https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html
[42]
A. Vaswani *et al.*, "Attention Is All You Need," *arXiv.org*, Jun. 12, 2017. https://arxiv.org/abs/1706.03762
[43]
"sklearn.model_selection.StratifiedKFold — scikit-learn 0.21.3 documentation," *Scikit-learn.org*, 2019. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html
[44]
"3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.21.3 documentation," *Scikit-learn.org*, 2009. https://scikit-learn.org/stable/modules/cross_validation.html
[45]
P. Wagner *et al.*, "PTB-XL, a large publicly available electrocardiography dataset," *Scientific Data*, vol. 7, no. 1, May 2020, doi: https://doi.org/10.1038/s41597-020-0495-6.

APPENDIX

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: First Semester Third Year	**Study week no.: 2**
Student Name & ID: Koh Zi Kang 22ACB00550	
Supervisor: Dr Mogana a/p Vadiveloo	
Project Title: Automatic Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence	

1. WORK DONE

1. Realized the intra-patient splitting
2. Trained intra-patient LSTM and GRU classifier

2. WORK TO BE DONE

1. Design architecture of 1D-CNN
2. Train intra-patient and inter-patient1D-CNN classifier

3. PROBLEMS ENCOUNTERED

1. Architecture design is a subjective

4. SELF EVALUATION OF THE PROGRESS

1. I should review and take reference from other research paper to look for an appropriate network design.

Supervisor's signature

KOH

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: First Semester Third Year	**Study week no.: 4**
Student Name & ID: Koh Zi Kang 22ACB00550	
Supervisor: Dr Mogana a/p Vadiveloo	
Project Title: Automatic Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence	

1. WORK DONE

1. Designed an architecture for 1D-CNN classifier
2. Successfully upload the dataset to Google Drive
3. Utilize the function from the wfdb library to read the data file to obtain the reading.
4. Assign the label (0/1) to each subject to differentiate them
5. Continued the literature review

2. WORK TO BE DONE

1. Split the data into a training set and a testing set at the patient level to ensure inter-patient analysis
2. Check the type, data type and shape of data
3. Calculate the number of MI patients and normal subjects
4. Visualize the ECGs

3. PROBLEMS ENCOUNTERED

1. 1D-CNN requires a input with different shape

4. SELF EVALUATION OF THE PROGRESS

1. The progress is smooth.

____
Supervisor's signature

___________KOH___________
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: First Semester Third Year	**Study week no.: 6**
Student Name & ID: Koh Zi Kang 22ACB00550	
Supervisor: Dr Mogana a/p Vadiveloo	
Project Title: Automatic Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence	

1. WORK DONE

1. Used train_test_split function from Sklearn to do the inter-patient data splitting.
2. The types, data types and shape of data were revealed through Numpy and Python built-in function
3. The total number of MI patients and normal subjects were revealed: 148 MI patients and 52 normal subjects
4. The ECGs were visualized using matplotlib

2. WORK TO BE DONE

1. Preprocessing steps such as denoise and baseline wandering
2. Search for implementation of Pan Tompkins algorithm to carry out the QRS and R peak detection
3. Segmentation of ECGs to produce distinct heartbeats

3. PROBLEMS ENCOUNTERED

1. It is memory-consuming to visualize all the ECGs

4. SELF EVALUATION OF THE PROGRESS

1. I should be more careful in memory management or subscribe to Google Colab Pro to have more memory

____
Supervisor's signature

____________KOH__________
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: First Semester Third Year	**Study week no.: 8**
Student Name & ID: Koh Zi Kang 22ACB00550	
Supervisor: Dr Mogana a/p Vadiveloo	
Project Title: Automatic Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence	

1. WORK DONE

1. A Savitzky-Golay Filter was used to denoise the ECGs and remove their baseline wandering
2. The Pan-Tompkins algorithm was found on GitHub
3. Segmentation was carried out using the R-peak for each heartbeat cycle

2. WORK TO BE DONE

1. Build LSTM network architecture
2. Build GRU network architecture
3. Build 1D-CNN network architecture
4. Training and evaluation of different network
5. Make the data into batches using the data loader function

3. PROBLEMS ENCOUNTERED

1. None

4. SELF EVALUATION OF THE PROGRESS

1. The progress is good

_____________________________ ____________KOH__________

Supervisor's signature Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: First Semester Third Year	**Study week no.: 10**
Student Name & ID: Koh Zi Kang 22ACB00550	
Supervisor: Dr Mogana a/p Vadiveloo	
Project Title: Automatic Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence	

1. WORK DONE

1. Network architecture is built for LSTM, GRU and 1D-CNN
2. Each model is trained using batches of training data
3. Each model is evaluated using batches of testing data
4. Visualization of the performance of each network

2. WORK TO BE DONE

1. The fine-tuning to search for better parameters for each network
2. Produce the FYP1 report
3. Poster design

3. PROBLEMS ENCOUNTERED

1. Time-consuming to train the network if the epoch number is too large.

4. SELF EVALUATION OF THE PROGRESS

1. I should properly manage my time

____
Supervisor's signature

___________KOH___________
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: First Semester Third Year	**Study week no.: 12**
Student Name & ID: Koh Zi Kang 22ACB00550	
Supervisor: Dr Mogana a/p Vadiveloo	
Project Title: Automatic Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

1. The fine-tuning process for each network
2. Poster was designed

2. WORK TO BE DONE

1. FYP 1 report
2. Presentation slide for moderation

3. PROBLEMS ENCOUNTERED

1. None

4. SELF EVALUATION OF THE PROGRESS

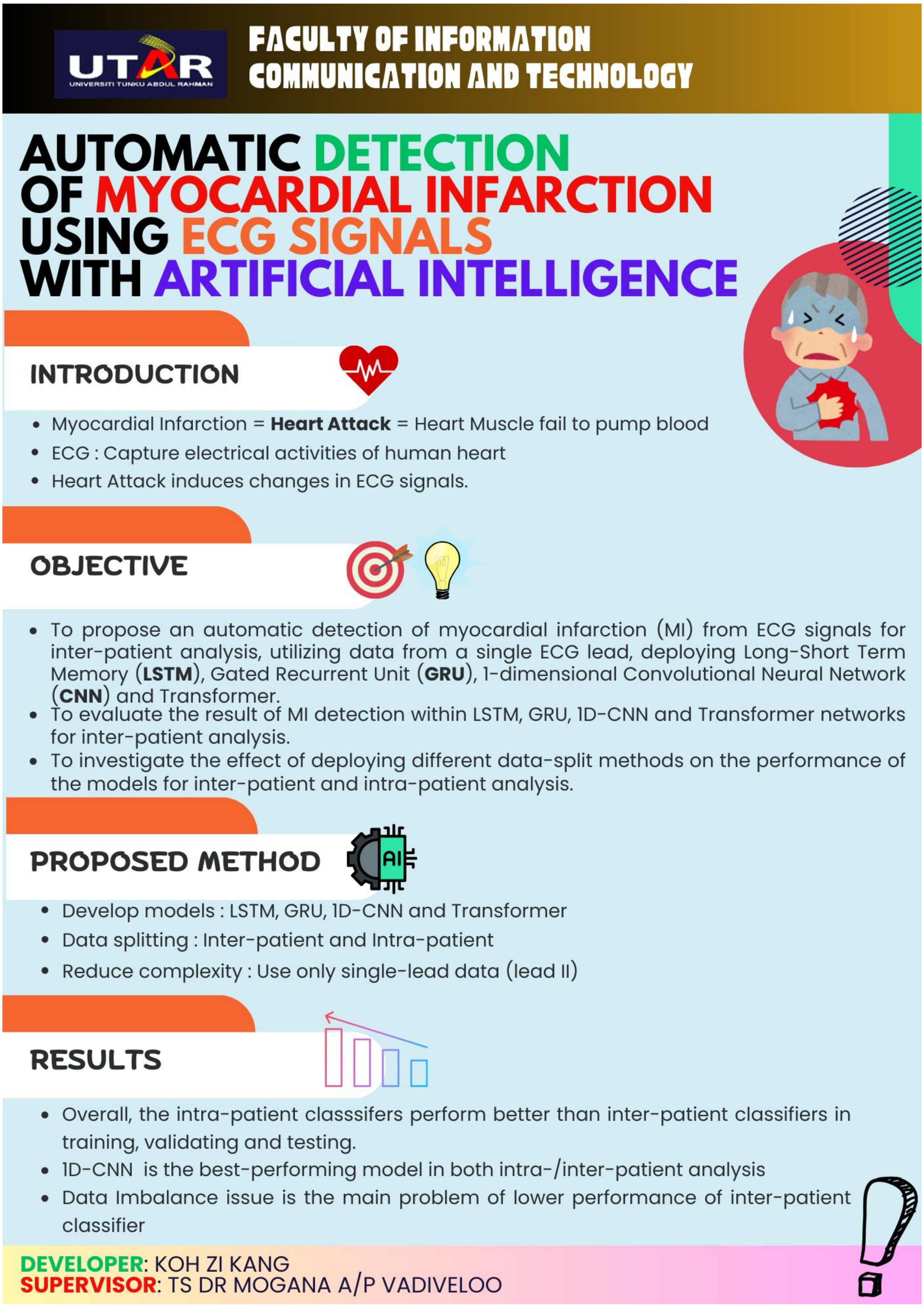
1. The progress is lagging behind

Supervisor's signature

KOH

Student's signature

POSTER



UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION COMMUNICATION AND TECHNOLOGY

AUTOMATIC DETECTION OF MYOCARDIAL INFARCTION USING ECG SIGNALS WITH ARTIFICIAL INTELLIGENCE

INTRODUCTION

- Myocardial Infarction = **Heart Attack** = Heart Muscle fail to pump blood
- ECG : Capture electrical activities of human heart
- Heart Attack induces changes in ECG signals.

OBJECTIVE

- To propose an automatic detection of myocardial infarction (MI) from ECG signals for inter-patient analysis, utilizing data from a single ECG lead, deploying Long-Short Term Memory (**LSTM**), Gated Recurrent Unit (**GRU**), 1-dimensional Convolutional Neural Network (**CNN**) and Transformer.
- To evaluate the result of MI detection within LSTM, GRU, 1D-CNN and Transformer networks for inter-patient analysis.
- To investigate the effect of deploying different data-split methods on the performance of the models for inter-patient and intra-patient analysis.

PROPOSED METHOD

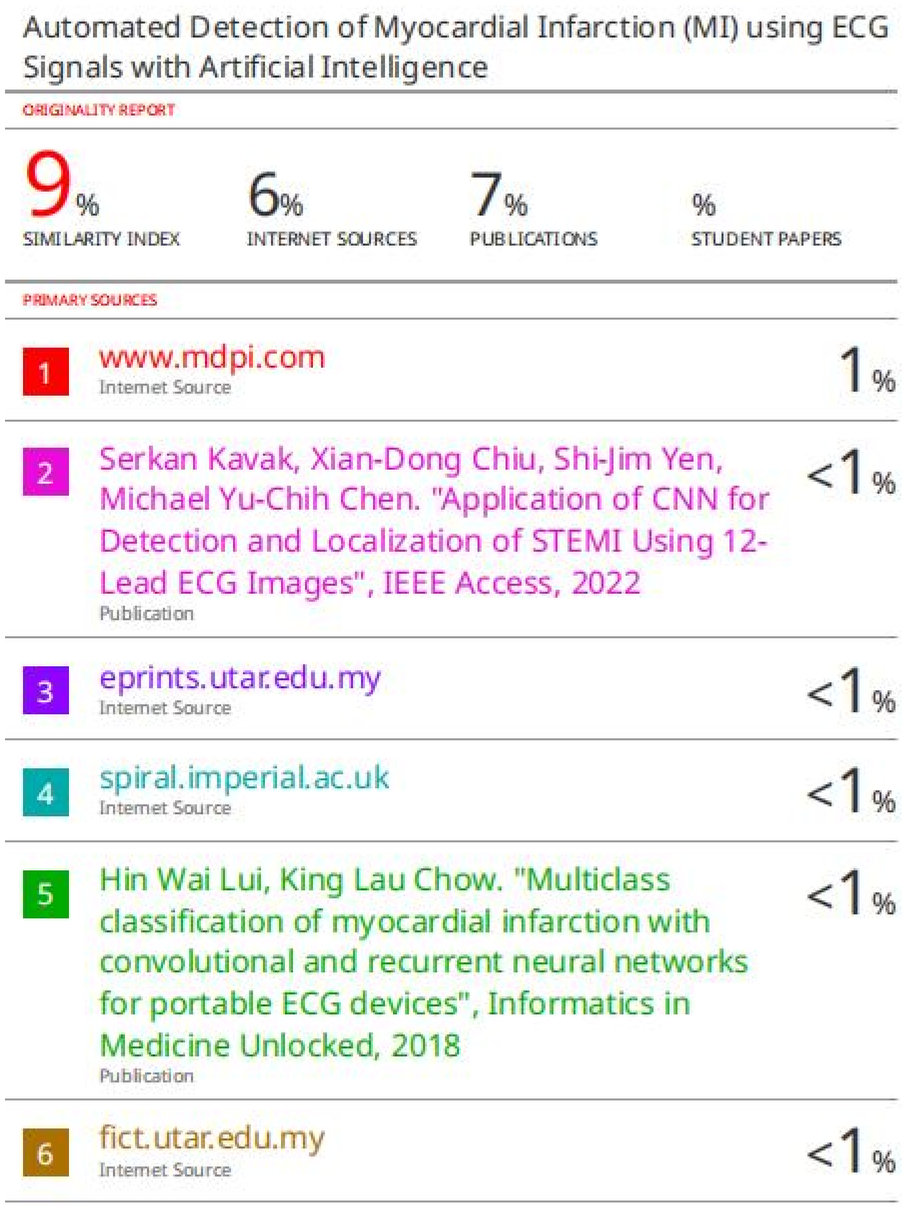
- Develop models : LSTM, GRU, 1D-CNN and Transformer
- Data splitting : Inter-patient and Intra-patient
- Reduce complexity : Use only single-lead data (lead II)

RESULTS

- Overall, the intra-patient classsifers perform better than inter-patient classifiers in training, validating and testing.
- 1D-CNN is the best-performing model in both intra-/inter-patient analysis
- Data Imbalance issue is the main problem of lower performance of inter-patient classifier

DEVELOPER: KOH ZI KANG
SUPERVISOR: TS DR MOGANA A/P VADIVELOO

PLAGIARISM CHECK RESULT



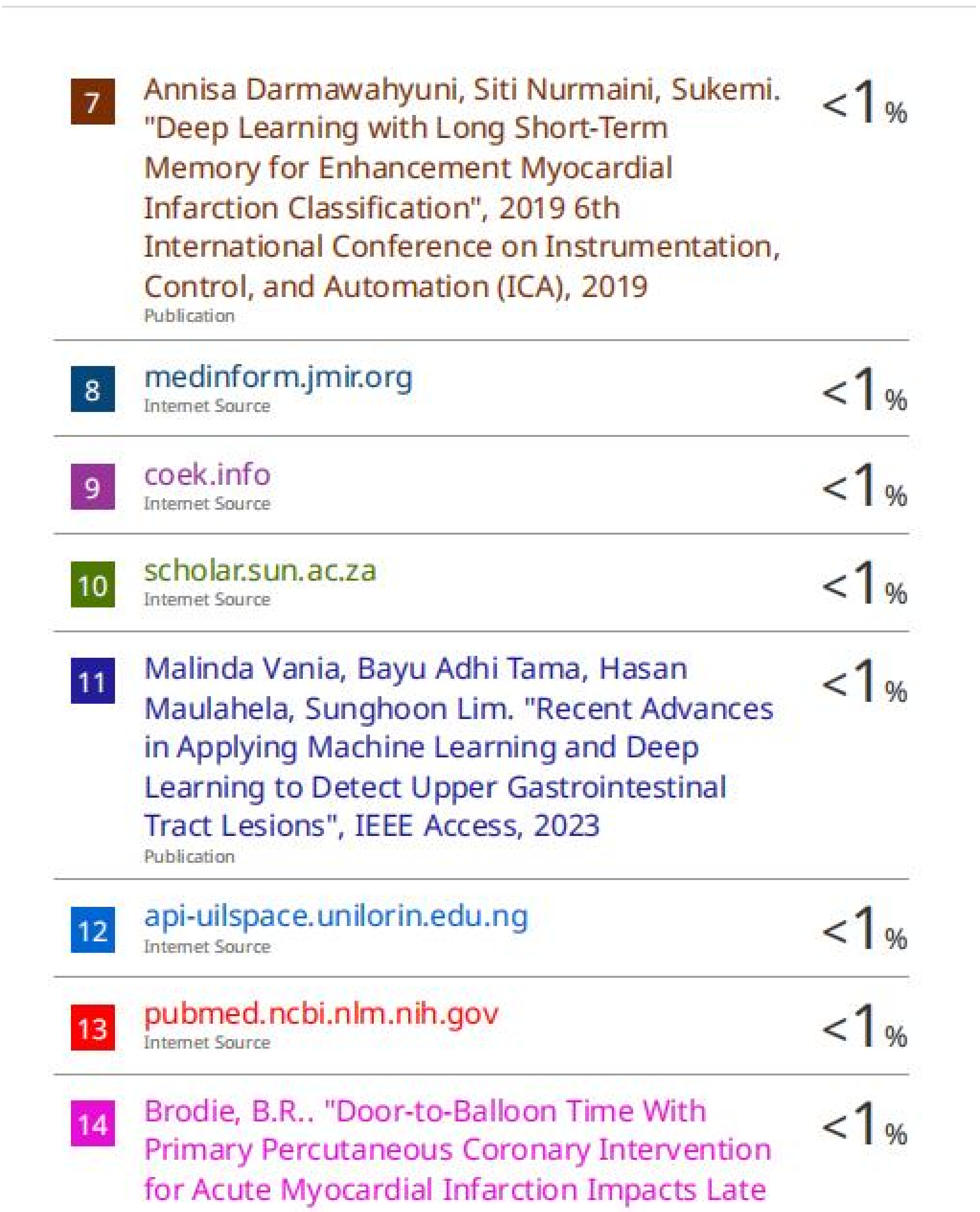
Automated Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence

ORIGINALITY REPORT

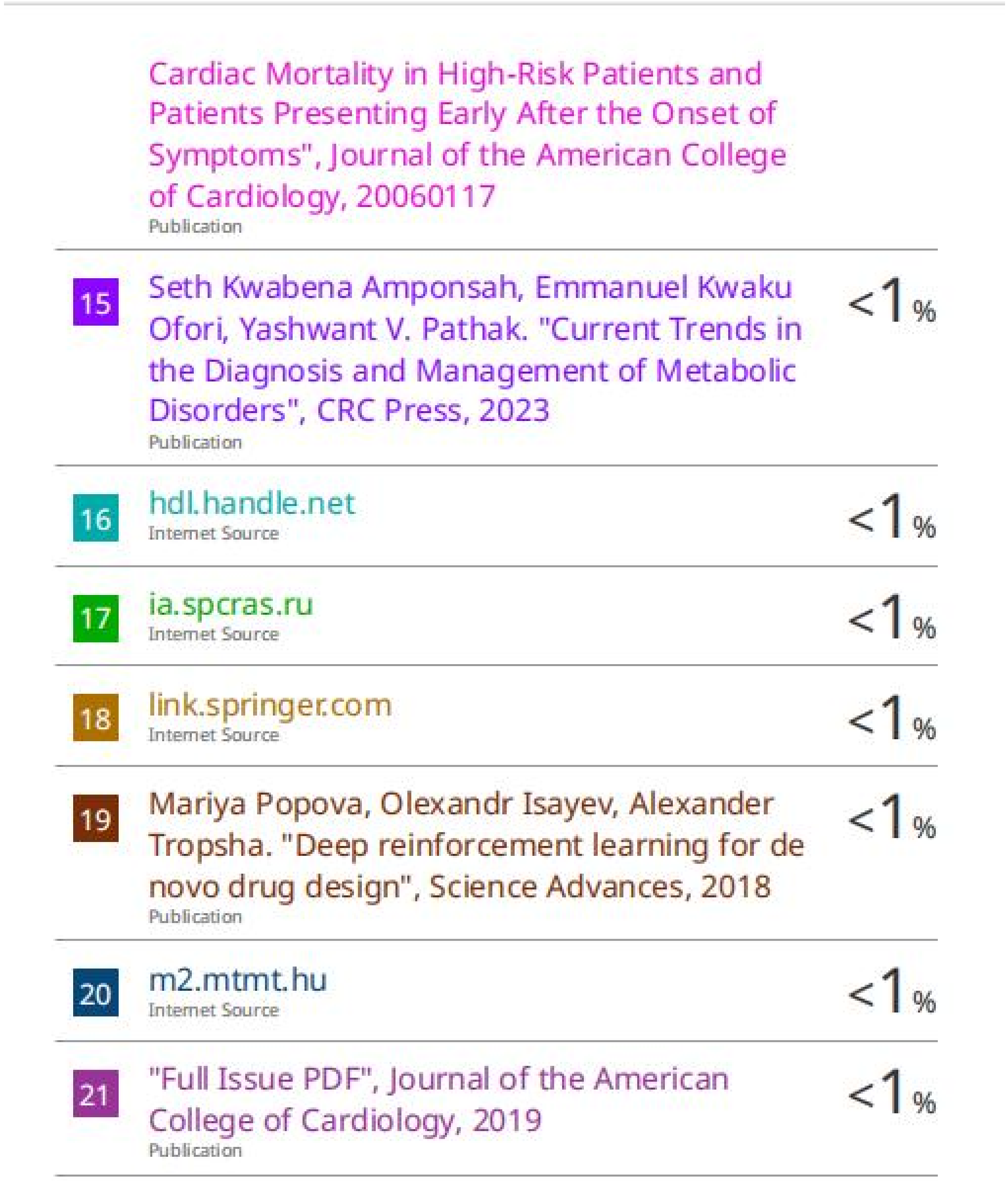
9%	6%	7%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

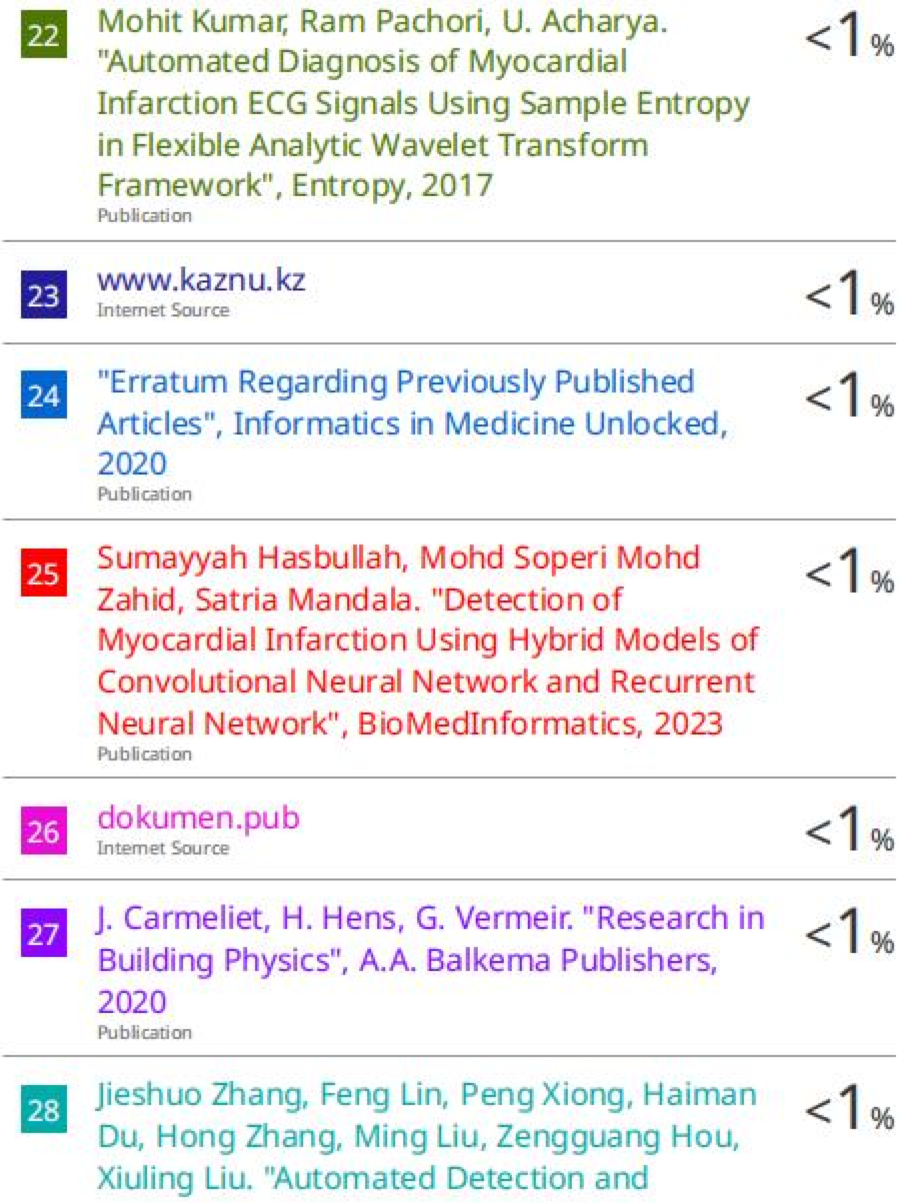
1	www.mdpi.com Internet Source	1%
2	Serkan Kavak, Xian-Dong Chiu, Shi-Jim Yen, Michael Yu-Chih Chen. "Application of CNN for Detection and Localization of STEMI Using 12-Lead ECG Images", IEEE Access, 2022 Publication	<1%
3	eprints.utar.edu.my Internet Source	<1%
4	spiral.imperial.ac.uk Internet Source	<1%
5	Hin Wai Lui, King Lau Chow. "Multiclass classification of myocardial infarction with convolutional and recurrent neural networks for portable ECG devices", Informatics in Medicine Unlocked, 2018 Publication	<1%
6	fict.utar.edu.my Internet Source	<1%



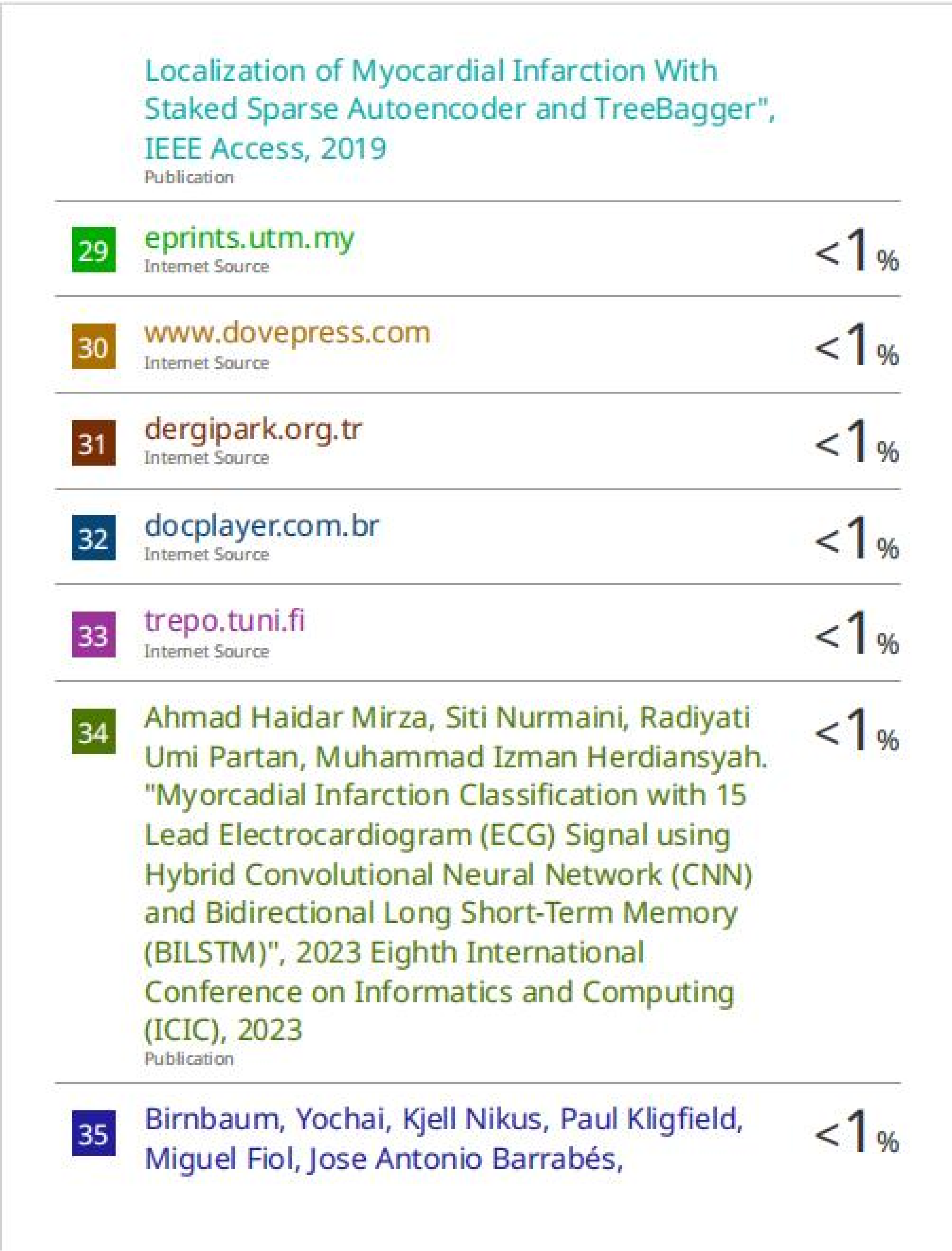
7	Annisa Darmawahyuni, Siti Nurmaini, Sukemi. "Deep Learning with Long Short-Term Memory for Enhancement Myocardial Infarction Classification", 2019 6th International Conference on Instrumentation, Control, and Automation (ICA), 2019 Publication	<1 %
8	medinform.jmir.org Internet Source	<1 %
9	coek.info Internet Source	<1 %
10	scholar.sun.ac.za Internet Source	<1 %
11	Malinda Vania, Bayu Adhi Tama, Hasan Maulahela, Sunghoon Lim. "Recent Advances in Applying Machine Learning and Deep Learning to Detect Upper Gastrointestinal Tract Lesions", IEEE Access, 2023 Publication	<1 %
12	api-uilspace.unilorin.edu.ng Internet Source	<1 %
13	pubmed.ncbi.nlm.nih.gov Internet Source	<1 %
14	Brodie, B.R.. "Door-to-Balloon Time With Primary Percutaneous Coronary Intervention for Acute Myocardial Infarction Impacts Late	<1 %



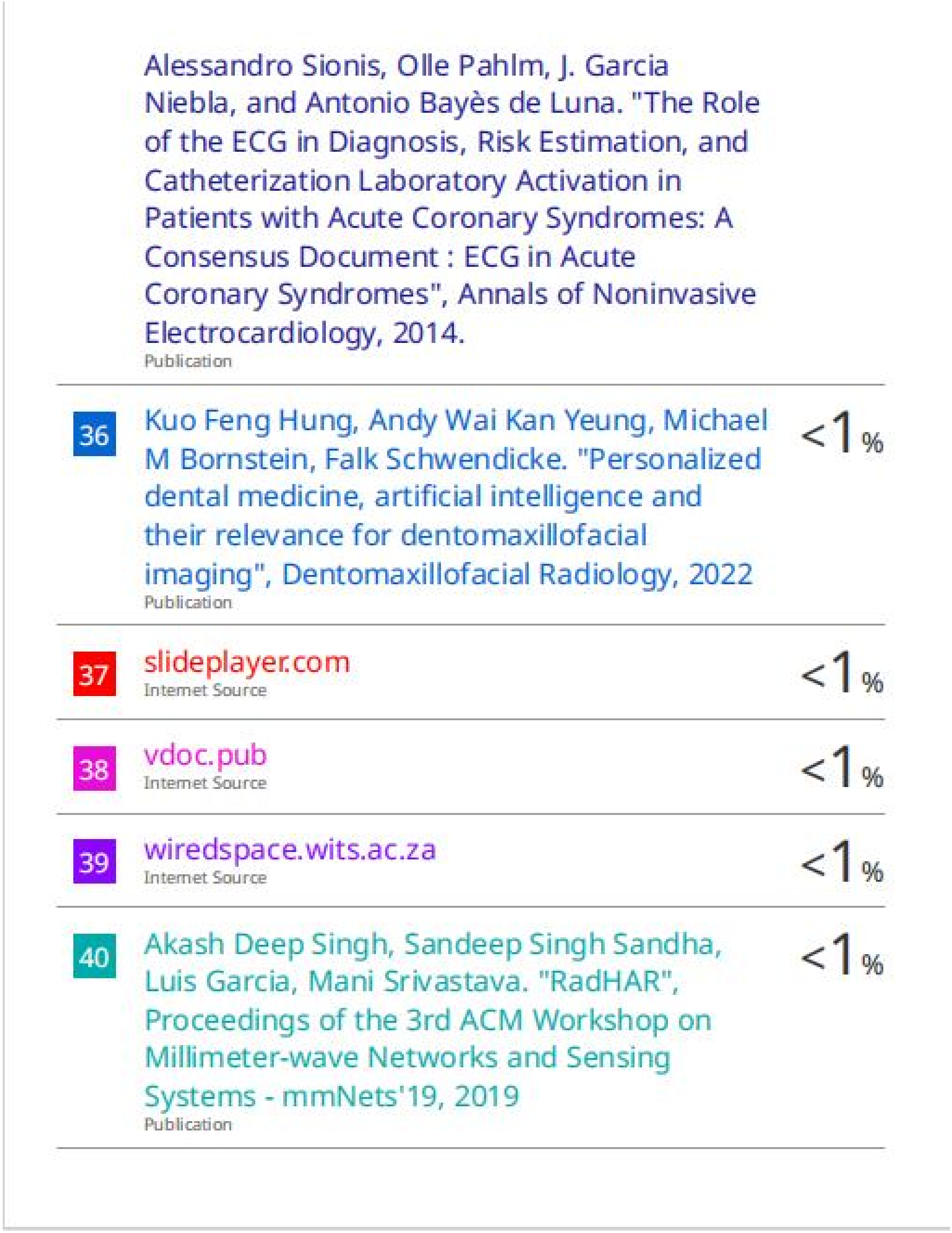
	Cardiac Mortality in High-Risk Patients and Patients Presenting Early After the Onset of Symptoms", Journal of the American College of Cardiology, 20060117 Publication	
15	Seth Kwabena Amponsah, Emmanuel Kwaku Ofori, Yashwant V. Pathak. "Current Trends in the Diagnosis and Management of Metabolic Disorders", CRC Press, 2023 Publication	<1 %
16	hdl.handle.net Internet Source	<1 %
17	ia.spcras.ru Internet Source	<1 %
18	link.springer.com Internet Source	<1 %
19	Mariya Popova, Olexandr Isayev, Alexander Tropsha. "Deep reinforcement learning for de novo drug design", Science Advances, 2018 Publication	<1 %
20	m2.mtmt.hu Internet Source	<1 %
21	"Full Issue PDF", Journal of the American College of Cardiology, 2019 Publication	<1 %



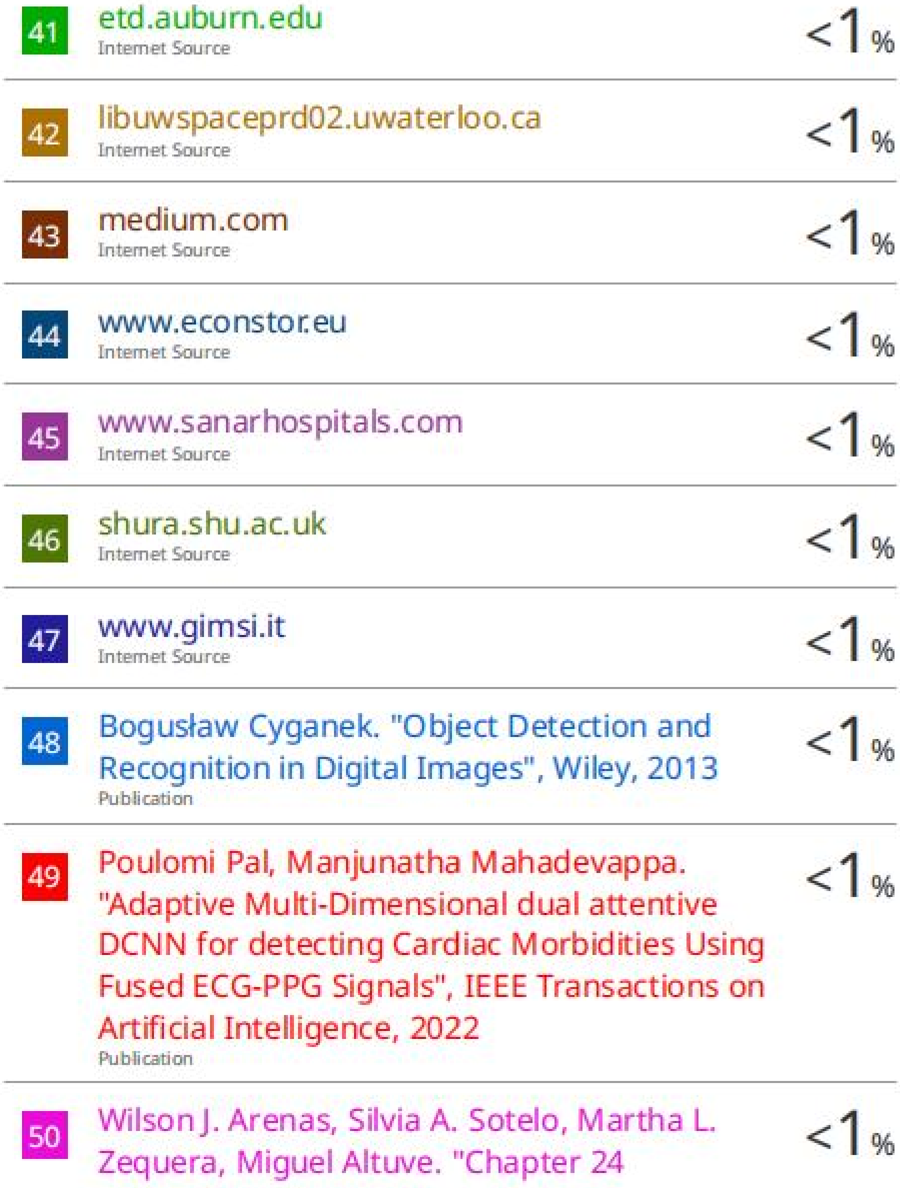
22	Mohit Kumar, Ram Pachori, U. Acharya. "Automated Diagnosis of Myocardial Infarction ECG Signals Using Sample Entropy in Flexible Analytic Wavelet Transform Framework", Entropy, 2017 Publication	<1%
23	www.kaznu.kz Internet Source	<1%
24	"Erratum Regarding Previously Published Articles", Informatics in Medicine Unlocked, 2020 Publication	<1%
25	Sumayyah Hasbullah, Mohd Soperi Mohd Zahid, Satria Mandala. "Detection of Myocardial Infarction Using Hybrid Models of Convolutional Neural Network and Recurrent Neural Network", BioMedInformatics, 2023 Publication	<1%
26	dokumen.pub Internet Source	<1%
27	J. Carmeliet, H. Hens, G. Vermeir. "Research in Building Physics", A.A. Balkema Publishers, 2020 Publication	<1%
28	Jieshuo Zhang, Feng Lin, Peng Xiong, Haiman Du, Hong Zhang, Ming Liu, Zengguang Hou, Xiuling Liu. "Automated Detection and	<1%



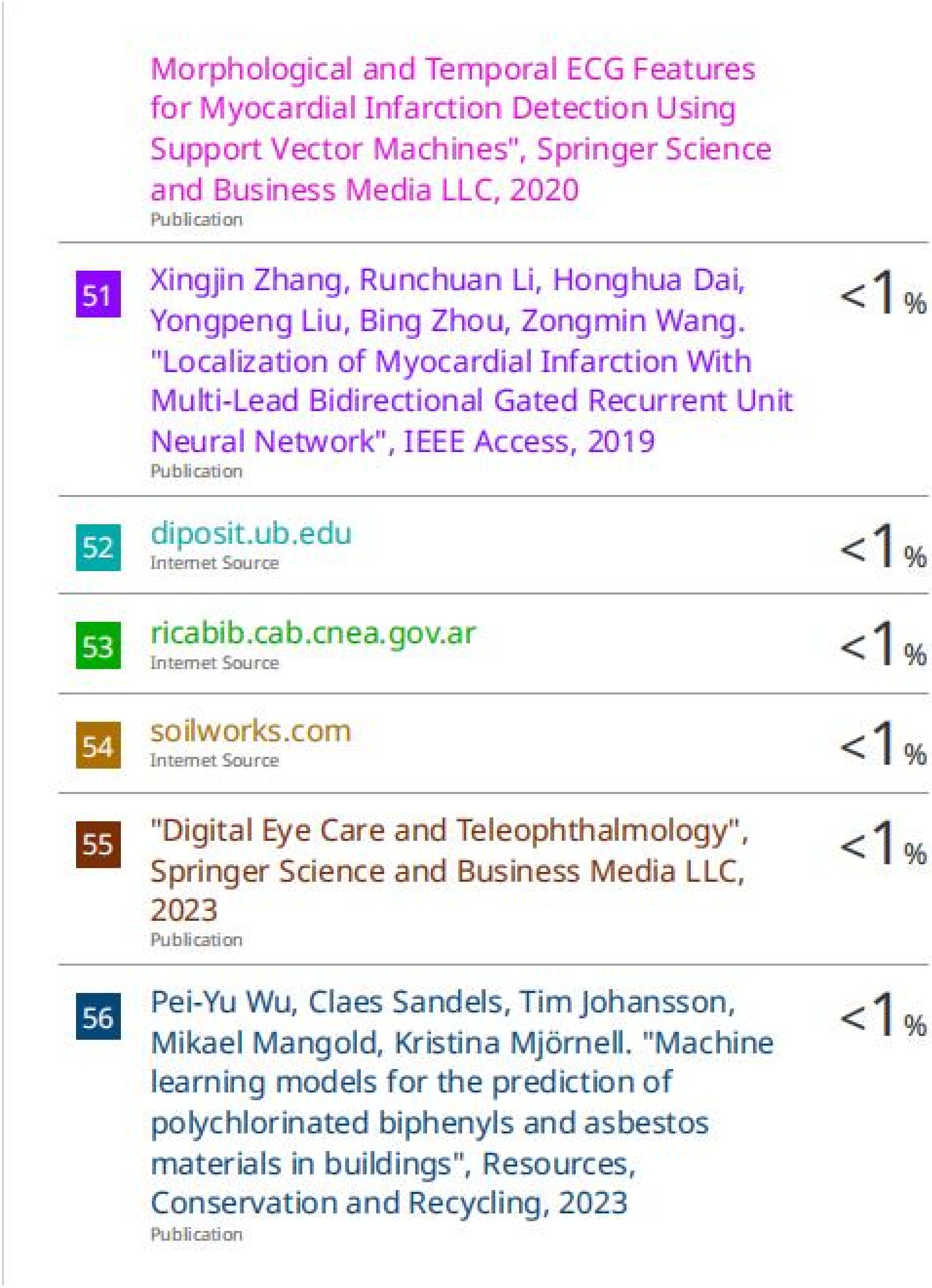
	Localization of Myocardial Infarction With Staked Sparse Autoencoder and TreeBagger", IEEE Access, 2019 Publication	
29	eprints.utm.my Internet Source	<1 %
30	www.dovepress.com Internet Source	<1 %
31	dergipark.org.tr Internet Source	<1 %
32	docplayer.com.br Internet Source	<1 %
33	trepo.tuni.fi Internet Source	<1 %
34	Ahmad Haidar Mirza, Siti Nurmaini, Radiyati Umi Partan, Muhammad Izman Herdiansyah. "Myorcadial Infarction Classification with 15 Lead Electrocardiogram (ECG) Signal using Hybrid Convolutional Neural Network (CNN) and Bidirectional Long Short-Term Memory (BILSTM)", 2023 Eighth International Conference on Informatics and Computing (ICIC), 2023 Publication	<1 %
35	Birnbaum, Yochai, Kjell Nikus, Paul Kligfield, Miguel Fiol, Jose Antonio Barrabés,	<1 %



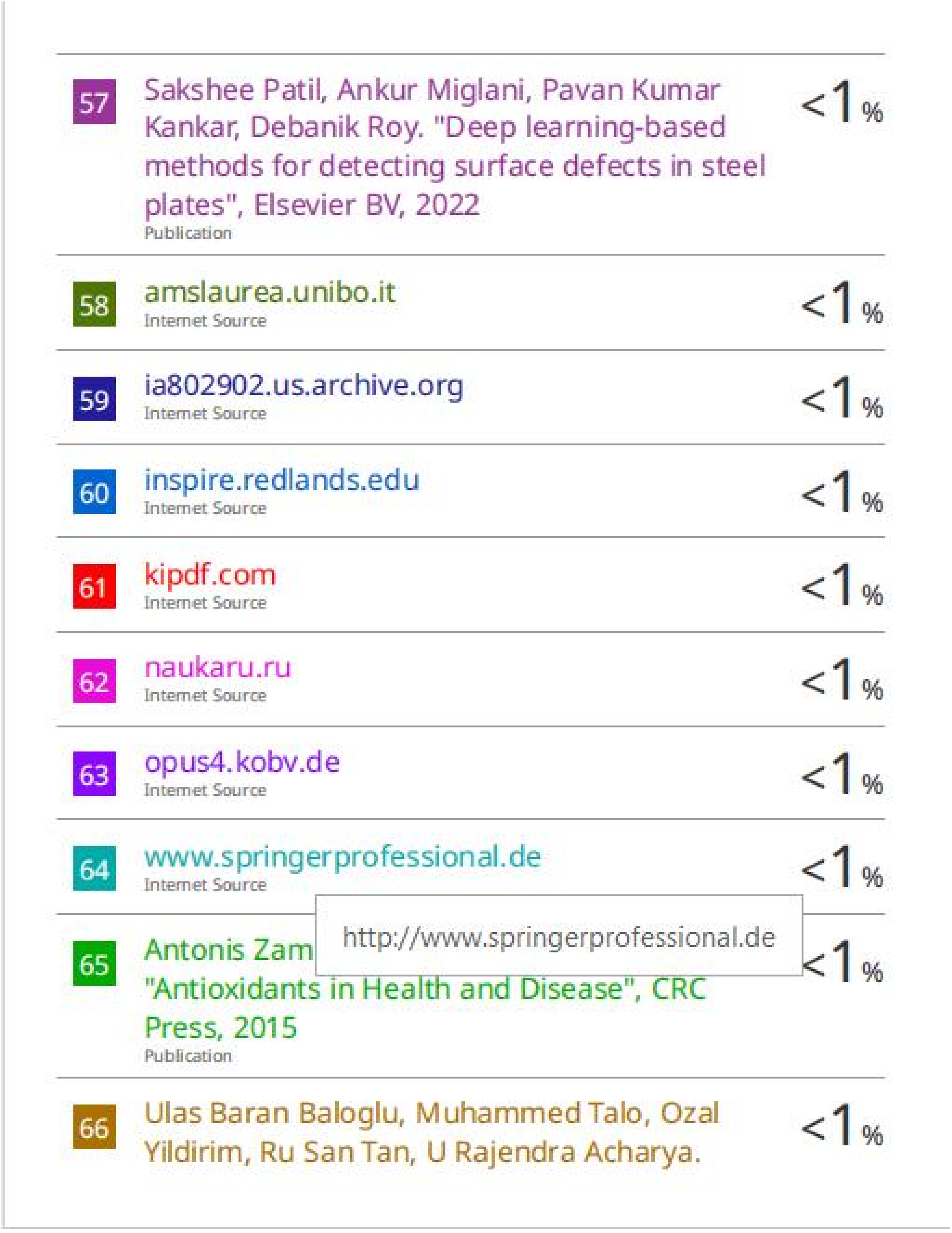
	Alessandro Sionis, Olle Pahlm, J. Garcia Niebla, and Antonio Bayès de Luna. "The Role of the ECG in Diagnosis, Risk Estimation, and Catheterization Laboratory Activation in Patients with Acute Coronary Syndromes: A Consensus Document : ECG in Acute Coronary Syndromes", Annals of Noninvasive Electrocardiology, 2014. Publication	
36	Kuo Feng Hung, Andy Wai Kan Yeung, Michael M Bornstein, Falk Schwendicke. "Personalized dental medicine, artificial intelligence and their relevance for dentomaxillofacial imaging", Dentomaxillofacial Radiology, 2022 Publication	<1 %
37	slideplayer.com Internet Source	<1 %
38	vdoc.pub Internet Source	<1 %
39	wiredspace.wits.ac.za Internet Source	<1 %
40	Akash Deep Singh, Sandeep Singh Sandha, Luis Garcia, Mani Srivastava. "RadHAR", Proceedings of the 3rd ACM Workshop on Millimeter-wave Networks and Sensing Systems - mmNets'19, 2019 Publication	<1 %



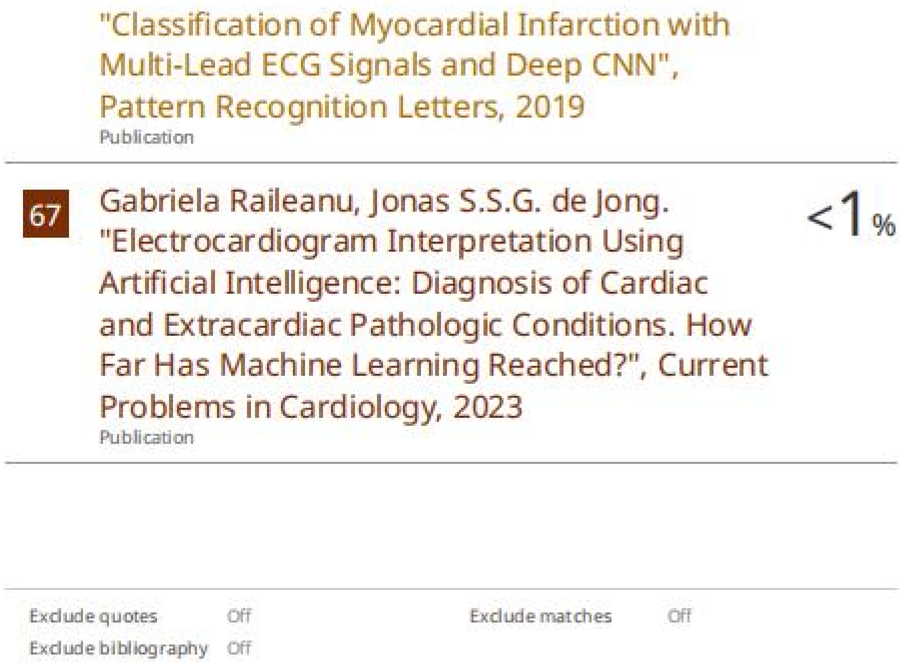
41	etd.auburn.edu Internet Source	<1%
42	libuwspaceprd02.uwaterloo.ca Internet Source	<1%
43	medium.com Internet Source	<1%
44	www.econstor.eu Internet Source	<1%
45	www.sanarhospitals.com Internet Source	<1%
46	shura.shu.ac.uk Internet Source	<1%
47	www.gimsi.it Internet Source	<1%
48	Bogusław Cyganek. "Object Detection and Recognition in Digital Images", Wiley, 2013 Publication	<1%
49	Poulomi Pal, Manjunatha Mahadevappa. "Adaptive Multi-Dimensional dual attentive DCNN for detecting Cardiac Morbidities Using Fused ECG-PPG Signals", IEEE Transactions on Artificial Intelligence, 2022 Publication	<1%
50	Wilson J. Arenas, Silvia A. Sotelo, Martha L. Zequera, Miguel Altuve. "Chapter 24	<1%



	Morphological and Temporal ECG Features for Myocardial Infarction Detection Using Support Vector Machines", Springer Science and Business Media LLC, 2020 Publication	
51	Xingjin Zhang, Runchuan Li, Honghua Dai, Yongpeng Liu, Bing Zhou, Zongmin Wang. "Localization of Myocardial Infarction With Multi-Lead Bidirectional Gated Recurrent Unit Neural Network", IEEE Access, 2019 Publication	<1 %
52	diposit.ub.edu Internet Source	<1 %
53	ricabib.cab.cnea.gov.ar Internet Source	<1 %
54	soilworks.com Internet Source	<1 %
55	"Digital Eye Care and Teleophthalmology", Springer Science and Business Media LLC, 2023 Publication	<1 %
56	Pei-Yu Wu, Claes Sandels, Tim Johansson, Mikael Mangold, Kristina Mjörnell. "Machine learning models for the prediction of polychlorinated biphenyls and asbestos materials in buildings", Resources, Conservation and Recycling, 2023 Publication	<1 %



57	Sakshee Patil, Ankur Miglani, Pavan Kumar Kankar, Debanik Roy. "Deep learning-based methods for detecting surface defects in steel plates", Elsevier BV, 2022 Publication	<1 %
58	amslaurea.unibo.it Internet Source	<1 %
59	ia802902.us.archive.org Internet Source	<1 %
60	inspire.redlands.edu Internet Source	<1 %
61	kipdf.com Internet Source	<1 %
62	naukaru.ru Internet Source	<1 %
63	opus4.kobv.de Internet Source	<1 %
64	www.springerprofessional.de Internet Source	<1 %
65	Antonis Zam "Antioxidants in Health and Disease", CRC Press, 2015 Publication	<1 %
66	Ulas Baran Baloglu, Muhammed Talo, Ozal Yildirim, Ru San Tan, U Rajendra Acharya.	<1 %



"Classification of Myocardial Infarction with Multi-Lead ECG Signals and Deep CNN", Pattern Recognition Letters, 2019

Publication

67	Gabriela Raileanu, Jonas S.S.G. de Jong. "Electrocardiogram Interpretation Using Artificial Intelligence: Diagnosis of Cardiac and Extracardiac Pathologic Conditions. How Far Has Machine Learning Reached?", Current Problems in Cardiology, 2023 Publication	<1%

Exclude quotes Off
Exclude bibliography Off
Exclude matches Off

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	KOH ZI KANG
ID Number(s)	22ACB00550
Programme / Course	Bachelor of Computer Science (Honours)
Title of Final Year Project	Automatic Detection of Myocardial Infarction (MI) using ECG Signals with Artificial Intelligence

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 9 % **Similarity by source** Internet Sources: 6 % Publications: 7 % Student Papers: %	No comments.
Number of individual sources listed of more than 3% similarity: -	No comments.

Parameters of originality required and limits approved by UTAR are as Follows:

(i) Overall similarity index is 20% and below, and

(ii) Matching of individual sources listed must be less than 3% each, and

(iii) Matching texts in continuous block must not exceed 8 words

Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor

Name: Mogana Vadiveloo

Date: 13/09/2024

Signature of Co-Supervisor

Name: ______________________

Date: ______________________

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	KOH ZI KANG	
Student Name	22ACB00550	
Supervisor Name	Dr Mogana a/p Vadiveloo	

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
x	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

__________KOH____________
(Signature of Student)
Date: 13/9/2024