DEEP LEARNING-BASED ECG CLASSIFICATION USING A TENSORFLOW LITE MODEL

by

Kushagra Sharma

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Graduate Program in Biomedical Engineering
The University of Manitoba
Winnipeg, Manitoba, Canada
December 2022

Thesis advisor
Dr. Rasit Eskicioglu

Author
Kushagra Sharma

DEEP LEARNING-BASED ECG CLASSIFICATION USING A TENSORFLOW LITE MODEL

Abstract

The number of IoT devices in healthcare is expected to rise sharply due to significantly increased demand since the COVID-19 pandemic. Deep learning and IoT devices are being employed to monitor body vitals and automate anomaly detection in clinical and non-clinical settings. Most of the current technology requires the transmission of raw data to a remote server, which is not efficient for resource-constrained IoT devices and embedded systems. In this work, we have developed machine learning models to be deployed on Raspberry Pi. We present an evaluation of our TensorFlow Model with various classification classes. We also present the evaluation of the corresponding TensorFlow Lite FlatBuffers to demonstrate their minimal run-time requirements while maintaining acceptable accuracy. Additionally, to address the problem of sensor and data integration when using multiple devices, we propose a unified server on our Edge Node.

Contents

Abstract ii
Table of Contents v
List of Figures vi
List of Tables vii
Acknowledgments ix
Dedication 1

1 Introduction **2**

2 Related Work **4**
2.1 Methods for Conserving Power in IoT devices 4
2.1.1 Offloading 4
2.1.2 Low Powered Transmission Protocols 6
2.1.3 Mobile-Edge/Edge Computing 7
2.1.4 Use of AI for power conservation 8
2.2 Applications of Artificial Intelligence in IoT 8
2.2.1 Improving performance and Efficiency 8
2.2.2 TensorFlow and Raspberry Pi Deployment 9
2.2.3 Signal Classification and R-peak detection 9

3 Background **12**
3.1 Human Heart and Electrocardiogram 12
3.1.1 Human Heart and the Blood Circulation System 12
3.1.2 Conduction System of the Heart 13
3.1.3 Cardiac Cycle 14
3.1.4 Einthoven's Triangle and ECG Leads 15
3.1.5 ECG signal waveform and Heart Anomalies 17
3.2 Internet of Things 24
3.3 Biomedical Applications of IoT 24
3.4 IoT and Artificial Intelligence 25
3.5 TensorFlow and Keras 26
3.6 TensorFlow Lite and Raspberry Pi 27

3.7 Convolutional Neural Network . . . 28
3.7.1 Long Short-Term Memory . . . 32
3.8 Public ECG Databases . . . 35
3.9 Raspberry Pi CM4 by Seeed - reTerminal . . . 35

4 Model and Prototype Design 37
4.1 Problem Overview . . . 37
4.2 Prototype Design . . . 38
4.3 Implementation Process . . . 39
4.3.1 Setting Up TensorFlow/Keras in Anaconda . . . 39
4.3.2 Data Pre-Processing . . . 40
Powerline Noise Removal . . . 41
Baseline Wander Correction . . . 41
Rolling Mean . . . 44
4.3.3 Determining Training Labels . . . 44
4.3.4 TensorFlow/Keras Model for 2-Class Classification . . . 46
4.3.5 TensorFlow/Keras Model for 5-Class Classification . . . 47
4.3.6 Determining Training Labels for Classification in 5 Classes . . . 49
4.3.7 Installing Raspbian OS . . . 49
4.3.8 Migrating TensorFlow model to Raspberry Pi . . . 49
4.3.9 Setting up Data Storage and Rendering . . . 50

5 Evaluation and Discussion 51
5.1 Evaluation Method . . . 51
5.1.1 Evaluation Metrics . . . 51
5.1.2 TensorFlow Lite Evaluation . . . 52
5.2 Results from 2-Class Classification Model . . . 53
5.2.1 Confusion Matrix . . . 53
5.2.2 Model Evaluation . . . 53
5.2.3 TensorFlow Lite Evaluation . . . 54
5.3 Results from 5-Class Classification Model . . . 54
5.3.1 Confusion Matrix . . . 54
5.3.2 Model Evaluation . . . 56
5.3.3 TensorFlow Lite Evaluation . . . 57
5.3.4 Edge Node/Raspberry Pi Web Interface . . . 57

6 Conclusions and Future Work 59
6.1 Conclusions . . . 59
6.2 Future Work . . . 60

Bibliography 68

Publication **69**

List of Figures

3.1 Einthoven's Triangle . . . 15
3.2 A Normal ECG Beat . . . 17
3.3 Signals for All Leads - ECG ID 9, PTB-XL Dataset . . . 18
3.4 Atrial Fibrillation - ECG ID 351, Lead I, PTB-XL Dataset . . . 19
3.5 Atrial Flutter - ECG ID 18, Lead I, PTB-XL Dataset . . . 20
3.6 First Degree AV Block - ECG ID 2648, Lead II, PTB-XL Dataset . . . 21
3.7 Second Degree AV Block - ECG ID 14009, Lead V2, PTB-XL Dataset . . . 21
3.8 Third Degree AV Block - ECG ID 10505, Lead I, PTB-XL Dataset . . . 22
3.9 Paced Signal - ECG ID 498, Lead I, PTB-XL Dataset . . . 22
3.10 Complete Left Bundle Branch Block Lead V5 ECG ID 180, PTB-XL Dataset 23
3.11 Complete Right Bundle Branch Block Lead V1 ECG ID 621, PTB-XL Dataset 23
3.12 A Typical Convolutional Neural Network . . . 28
3.13 Popular Activation Functions . . . 31
3.14 A LSTM Cell . . . 34
3.15 General Architecture . . . 36

4.1 Raw Signal and Spectrogram for ECG ID 1, Lead I, PTB-XL . . . 40
4.2 Filtered Signal and Spectrogram for ECG ID 1, Lead I, PTB-XL . . . 41
4.3 Baseline Correction for ECG ID 1, Lead I, PTB-XL . . . 44
4.4 Distribution of Occurrences after Sorting and Merging Labels . . . 45
4.5 Training, Validation and Testing Distribution . . . 46
4.6 Binary Classification Model Architecture . . . 47
4.7 5-Class Classification Model Architecture . . . 48
4.8 Train, Test and Validation split - 5 Class . . . 49

5.1 2-Class Classification Confusion Matrix . . . 53
5.2 5-Class Classification Confusion Matrix . . . 56
5.3 Edge Node/Raspberry Pi Web Interface . . . 58

List of Tables

3.1 Open Access ECG Databases 35

5.1 2-Class TensorFlow Keras Model Evaluation 54
5.2 2-Class TensorFlow Lite Model Evaluation 54
5.3 Confusion Matrix for all Multi-Label Classification CNN-LSTM Models . . 55
5.4 5-Class TensorFlow/Keras Model Evaluation 56
5.5 5-Class TensorFlow Lite Model Evaluation 57

Acronyms

AI Artifical Intelligence. 38

AVN Atrioventricular Node. 13

BLE Bluetooth Low Energy. 6

CAGR Compounded Annual Growth Rate. 2

CM Compute Module. 35

CNN Convolutional Neural Network. 3, 28, 29

ECG Electrocardiogram. 2

eMMC embedded Multi-Media Card. 35

FFT Fast Fourier Transform. 5

FN False Negatives. 52

FP False Positive. 52

GPIO General Purpose Input Output . 35, 38

IoT Internet of Things. 2, 24

LSTM Long Short-Term Memory. 3, 32, 38

RNN Recurrent Neural Network. 32, 38

SAN Sino-Atrial Node. 13

TN True Negatives. 52

TP True Positive. 52

Acknowledgments

I want to express my cordial gratitude to Dr. Rasit Eskicioglu for being my mentor throughout this research. The amount of time he has given to guide me in improving my research skills is invaluable. I also want to thank Dr. Sherif Sherif and Dr. Cuneyt Akcora for being a part of my committee and for their valuable suggestions and feedback. Special thanks to my labmate Baha Rababah and Md Abdullah Al Mamun for their help and advice. I also want to thank all the faculty members of the Graduate Program in Biomedical Engineering, Rady Faculty of Health Sciences and Price Faculty of Engineering who helped me to increase my skills while completing their courses and through valuable seminars. Finally, I would like to acknowledge the support of the staff members at the University of Manitoba for providing me with the support to fulfill my dream of getting a Master's degree in Biomedical Engineering.

This work is dedicated to my mother and everyone who has supported me in all my endeavours.

Chapter 1

Introduction

Over the past few years, there has been a tremendous increase in IoT devices such as smartwatches, virtual assistants, smart plugs, smart switches, and healthcare devices like intelligent insulin pumps, pulse-oximeter, and heart rate monitors. There were more than 26 billion connected devices in 2019. This number may reach 75 billion by 2025 and over 500 billion by 2030 [1] [2]. The applications of Internet of Things (IoT) devices in healthcare have also risen sharply due to significantly increased demand since the COVID-19 pandemic. For many regions, the IoT-healthcare sector's Compounded Annual Growth Rate (CAGR) is above 20%. Rising awareness and AI-enabled IoT devices detecting real-time anomalies are significant growth drivers [3]. Rapidly increasing 5G network coverage is also contributing to the growth by enabling more connected devices as it offers a larger spectrum, better speed, and lower latency [4].

An increasingly enormous amount of data is being generated and transmitted with the addition of more devices in the house, and healthcare setting [5]. Since these devices are often battery operated, optimizing them for increasing operational life is necessary [6]. Higher power consumption in processing or transmitting the data leads to shorter battery life and increased maintenance costs. Additionally, implantable medical devices cannot be changed that often. Therefore having a longer battery life is more crucial for implantable devices [1].

Electrocardiogram (ECG) signals need a high sampling rate for clinical use and for higher precision applications [7]. Offloading such data to a server for computation re-

quires high transmission activity, negatively impacting energy efficiency. It leads to the battery draining faster as the energy consumed in data transmission would exceed the energy consumed in on-device processing [8]. Numerous models have been explored to mitigate these issues - IoT-fog-cloud architecture [9][10][11], Mobile Edge Computing [12], Distributed Mobile Edge Computing [10], and 'Reinforcement Q-learning Model' [13]. Therefore, computing ECG data on the IoT device using an efficient machine learning model appears ideal.

Machine learning is employed to capture variations in ECG signals. Numerous machine learning models were explored to classify ECG signals. However, they are often not suitable for applications on new data due to their limited information based on their smaller training dataset. PTB-XL [14] dataset provides 21,837 12-lead recording from 18885 patients. PTB-XL dataset covers a wide span of signals and diagnostic classes [15]. It exposes the machine learning model to various signals that could arise in real-life application scenarios. Since the database has 12 leads, models can be developed for clinical and non-clinical applications. Clinical applications often utilize all 12 leads, whereas home users utilize 1 to 3 leads.

Recent literature suggests a drift towards using LSTM and CNN to classify ECG data on wearable devices [16][17]. Since binary Long Short-Term Memory (LSTM) could run on the limited memory of wearable devices [18], their application is increasingly being explored. Numerous other models are being explored and adapted for the use case scenario. This research primarily focuses on leveraging One Dimensional-Convolutional Neural Network (CNN) for modeling using TensorFlow/Keras.

This research proposes developing an expandable framework using Raspberry Pi 4 as our edge node, which is deployed with a trained neural network for classifying ECG signals from a sensor using TensorFlow Lite. Along with the overall accuracy, the accuracy of our deployed model on the edge node will be compared against the TensorFlow model using the same data. Also, the results from our edge node will be transmitted to a server for displaying the output.

Chapter 2

Related Work

New applications for IoT are being developed and adopted at an unprecedented pace. The strategy for handling efficiency issue vary vastly due to the application needs, scenarios, and hardware constraints. Some applications may need computational efficiency, while others may prefer lower latency. Some applications need prolonged battery life with acceptable compromise on efficiency or vice versa. In this section, we will look at some IoT applications and the strategies that are being implemented depending on the use case scenario.

2.1 Methods for Conserving Power in IoT devices

Most of the time, IoT devices are not connected by a constant power supply. Therefore, prolonging their battery life has always been one of the topmost priorities. Over the years, various methods have been developed, adapted, and proposed for IoT devices, but we will limit ourselves to today's more widely available and practical ones.

2.1.1 Offloading

Offloading is transferring of compute-intensive tasks or applications from a resource-constrained IoT device to a more capable computer. Offloading could take various forms such as Cloud Computing, Edge Computing, Fog Computing, Distributed Edge Computing, etc. Over the years, mixed models have also been developed and tested. However, the

choice of the model vastly depends upon the use case scenario.

Many applications are computationally demanding and are not capable or efficient enough to process the data themselves. Therefore they offload such data to the cloud, or nearby node for processing [12] in order to conserve battery. It is often the case with applications that need to transmit more extensive data frequently, such as real-time monitoring of traffic, body vitals such as heart rate, etc. However, offloading such data comes with some challenges as the transmission of information consumes energy and uses computational resources.

For making offloading energy efficient for IoT devices following issues need to be addressed – traffic fluctuation, collision listening, over-hearing, idle listening, and protocol overhead reduction [19]. These issues are more prevalent when data is being uploaded to the cloud directly or to a distant node. Therefore, to provide a stable connection and to avoid transmission losses, using edge nodes for local data gathering is preferred. This also helps in reducing the latency for real-time applications. This edge node is often connected to constant power and is capable enough to handle numerous IoT devices. This setup ensures minimal transmission losses but often constricts the portability of the IoT devices. For covering a wider area, multiple fog nodes may be used.

However, offloading everything is not the ideal choice as the power loss in transmission increases substantially [8]. Often A mixed approach is used for applications, where some computational tasks are offloaded, and some are performed locally. Borja Martinez *et al.* [20], in their research for analyzing the energy life cycle of an application, considered a system-level perspective for energy expenditures in communications, acquisitions, and processing. Their simulation of sub GHz reporting type of sensor using the LoRa approach suggests that the overall energy consumption is reduced, with lower radioactivity. In their experiment with the Time Slotted Channel Hopping scheme, no significant gain was observed for sensors performing Fast Fourier Transform (FFT) by increasing timeslots after a stage. It indicates that for achieving optimum efficiency, a balance needs to be achieved between power loss in computation and transmission.

Additionally, offloading induces a delay or latency issue, which is not suitable for applications like real-time positional monitoring in automated, traffic monitoring, etc. Om-Kolsoom Shahryari *et al.* [9], in their IoT-fog-cloud architecture, which involves op-

timizing transmit power and offloading probability in a multi-user multi-fog environment, demonstrates reduced delay and better power efficiency. It is due to their strategy of partially offloading the data based on the computational availability of nearby nodes. They also showed that the complete offloading strategy has the worst performance.

In terms of biomedical context, certain types of physiological signals, such as ECG, EEG, etc., have a high sampling rate [7]. Direct offloading of such data to servers, especially in a mobile environment, requires more significant transmission activity for both sending and receiving nodes, negatively impacting energy efficiency. It, in turn, leads to batteries draining faster as energy consumption in the transmission is much greater than in processing the data [8].

2.1.2 Low Powered Transmission Protocols

Sensor data is not usually massive in terms of size. However, transmission frequency may vary vastly and affect operational life due to transmission losses. Low-powered protocols save power consumed in transiting the data and provide an additional layer of security with over-the-air encryption. Various proprietary and non-proprietary/open protocols have been developed to handle this issue, such as Bluetooth Low Energy (BLE), z-wave, Zigbee, 6LoWPAN, etc. BLE and Zigbee are the most common protocols for commercial applications.

Bluetooth Low Energy [21], an open protocol, is more prevalent among modern-day intelligent devices such as virtual assistants, smart bulbs, and plugs. BLE is often used in medical devices and personal healthcare intelligent products. Bluetooth Low Energy provides a bandwidth of up to 2 Mb/s [21]. BLE uses a 2.4 GHz band with 40 channels, 3 for advertising and 37 for data.

Zigbee [22] is a proprietary low-power protocol that can deliver speeds up to 250 kbps. It operates between 868MHz to 2.4GHz and can deliver up to 100m. Although its data transmission capability is inferior to BLE, it consumes less power than BLE. Zigbee is used in automated homes, embedded sensors or industrial control systems.

2.1.3 Mobile-Edge/Edge Computing

More devices are being incorporated into making smart homes and intelligent buildings [5]. With the increase in the number of devices, more data is being generated and transmitted, which creates a power efficiency challenge for sensor and edge/fog nodes. IoT devices must be power-efficient as they maintain an extensive infrastructure of web-enabled devices such as sensors, wireless communication hardware or processor. Since nodes are usually battery-driven, conserving power is crucial to elongate their operational life [6]. Higher power consumption leads to a shorter life span and increases the cost of maintenance.

Offloading computation-intensive data to a conventional centralized cloud introduces transmission delay back and forth. Time spent in computation also adds to the delay. This latency in real-time data might be out of acceptable margins. Offloading data to edge or for node increases efficiency by reducing the latency of service computing [13]. Offloading methods involving fog nodes while maintaining acceptable delay are being explored in IoT-fog-cloud architecture for multi-user multi-fog node scenarios [9].

A survey by Pavel Mach *et al.* [12] discusses an emerging technology - Mobile Edge Computing (MEC). They studied various architectures such as distributed MEC or partial offloading to the cloud. It was observed that different architectures seem to work efficiently for specific use case scenarios. It suggests that applications need to be crafted according to their use case scenario to improve efficiency. Another Survey by Ju Ren *et al.* [10], reflects the rapid growth of interest in MEC among the research community, which seems to be promising. Md. Golam Rabiul Alam *et al.* [13] used autonomic offloading to ME/fog using the 'Reinforcement Q-learning Model'.

A majority of research discussed so far indicates that full offloading is both energy expenditure and increases delay, which might not suit many real-time applications.

2.1.4 Use of AI for power conservation

Different computational strategies are being developed and tested to increase the operational life of IoT devices. These strategies include AI solutions, fast data analysis, and prediction mechanisms that are rapidly being adopted for IoT devices according to the use [2]. These strategies help conserve energy by optimizing computational, transmission, or offloading efficiency. Energy-saving using deep learning autonomic management framework is also being explored in IoT enabled buildings [23] and for offloading demands of massive mobile networks in mobile edge/fog systems. Autonomic offloading [13] and Distributed intelligence models [24] to ME/fog using the 'Reinforcement Q-learning Model'. Transfer learning is also being employed to transfer pretrained models for on-device inference of data [25].

2.2 Applications of Artificial Intelligence in IoT

Artificial Intelligence is increasingly being employed in every aspect of IoT devices, from managing wireless transmission to performing on-device calculations more efficiently. It is also being used to make automated decisions without manual intervention. The combined effect is better operational costs and increased scalability. New applications are being developed with AI for devices incorporated into smart homes and intelligent buildings [5]. In this section, we'll have a look at some of the domains where AI is actively being incorporated.

2.2.1 Improving performance and Efficiency

Strategies for optimizing efficiencies in real-time data, such as processing per-Task and per-CPU operations on multi-core platforms, are being explored. C. Zhou *et al.* propose an energy-aware mapping technique for real-time data to explore energy savings in multi-core platforms. Research is done by Marius Laska *et al.* [11] provides an insight into the importance of using mobile edge/fog deep learning automation for better real-time vehicle localization. Far-end network solutions increase the network latency and impact real-time performance negatively [11].

Borja Martinez *et al.* [20], in their research for analyzing the energy life cycle of an application, considered a system-level perspective for energy expenditures in communications, acquisitions, and processing. Their simulation of sub GHz reporting type of sensor using the LoRa approach suggests that the overall energy consumption is reduced with lower radioactivity. In their experiment with the Time Slotted Channel Hopping scheme, no significant gain was observed increasing timeslots after a stage for sensors performing FFT. This indicates that for achieving optimum efficiency, a balance needs to be achieved between power loss in computation and transmission. The model they developed helps estimate energy consumption requirements by an IoT application.

2.2.2 TensorFlow and Raspberry Pi Deployment

In the past few years platforms such as TensorFlow Lite, for deploying trained neural networks on low-powered devices such as Raspberry Pi 4 have emerged for real-time applications. A similar framework was also developed by MatLab. This indicates that more applications in the future will be centered around increasing the overall efficiency and productivity of IoT devices using machine learning.

TensorFlow Lite [26] is an open-source deep learning framework for on-device inference [27]. With the increase in wireless technology, more applications are being explored for IoT devices requiring sophisticated deep learning models on edge and embedded devices [28][29][30]. TensorFlow Lite converts the TensorFlow/Keras saved model to a FlatBuffer. A Flatbuffer is an serialization cross-platform library for C, C++, C#, Java or Python. The Flatbuffer can be executed with much fewer resources. During the conversion process, quantization optimizations are used to reduce the model's size. Flatbuffers are memory efficient, cross-platform compatible, and smaller in size. They do not need to be parsed or unpacked; they are ideal for resource-constrained IoT devices [31].

2.2.3 Signal Classification and R-peak detection

There is plenty of literature regarding peak detection and ECG signal classification using different machine learning algorithms. Survey papers by Hongzu Li *et al.* [32] and David Menotti *et al.* [33] discuss numerous methods for heart anomaly detection. Some

methods rely on hardware enhancements, while others incorporate advanced software processing. We will confine ourselves to the software segment only.

Guy J. J. Warmerdam *et al.*[34] developed a multi-channel hierarchical probabilistic framework with predictive modeling for fetal ECG R-peak detection. Their method presented an overall efficiency of over 99%. H. B. Seidel *et al.* [35] proposed using Haar-DWT hardware architecture for energy-efficient processing of ECG signals while maintaining an R-peak detection accuracy of 99.68%. Jeong-Seon Park *et al.* [36] have used wavelet transform and modified Shannon energy envelope for detection of R-peak. Their proposed method reports an average accuracy of over 99% for detecting R peaks. Agostino Giorgio *et al.* [37] discuss the detection of late ventricular potentials using wavelet denoising and support vector machine classification. His experiment yields an accuracy of over 90%.

The literature suggests numerous ways for classifying ECG signals - CNN, RNN, SVN, K-NN, and MLP [38][39] [40]. Using MATLAB simulation, Abdelhamid Daamouche *et al.* [41] and Milad Nazarahari *et al.* [42] classified ECG signals using wavelets. The average accuracy of both methods is over 80%, along with other similar methods[43][44][45][46]. Serkan Kiranyaz *et al.* [47] propose a fast and accurate method to classify ECG signals using a trained, dedicated CNN for a patient. It helps in quick real-time classification of long ECG data stream. Their model demonstrates great performance in classifying ventricular ectopic beats and supraventricular ectopic beats.

Jia Li *et al.* [40] proposed a model that converts 1-D information vector to 2-D image via one-hot encoding. ADADELTA optimizer increased their learning rate substantially. Their method yields an accuracy of over 99%. Their method could be suitable for the classification of arrhythmia in portable devices.

Seyed Ahmad Mirsalaria *et al.* [18] proposed using lightweight binary LSTMs to cope with the limited memory and processing capabilities of wearable devices. They used 5-level binarized input with 1 level of binarization for weights. The proposed method was computationally inexpensive and provided accuracy almost equal (reduced by 0.004%) to conventional LSTM methods.

Transfer learning is also being used to deploy pertained CNN models for on-device inference of data [25]. Kuba Weimann *et al.* [25] used transfer learning using CNN to classify heart rhythm from a short ECG recording. They explored unsupervised pre-training of

ECG data which yielded acceptable results. This approach is suited for applications where large ECG annotation is not available.

Saeed Saadatnejad *et al.* [48] used wavelet transforms for feature extraction and RNN-LSTM for the classification of heart-beats. It uses results from two different classification model, which is then blended to form the final classification model.

Recent literature suggests a drift towards using LSTM and CNN to classify ECG data on wearable devices. Although they present good accuracy, these models cannot be used for real-time applications due to their limited information about anomaly classes.

Chapter 3

Background

3.1 Human Heart and Electrocardiogram

3.1.1 Human Heart and the Blood Circulation System

Section 3.1.1 to 3.1.6 has been adapted from [49] [32].

The human heart is the primary organ of the blood circulation system. It is made up of strong cardiac muscles powered and coordinated by electrical impulses. The human heart consists of four chambers - two atria and two ventricles. The atria and the ventricle are separated by the atrial and ventricular septum, respectively. The atria and ventricle on the left and the right side are separated by the Mitral and tricuspid valve, respectively. The heart's primary function is to move the blood through the body. The circulation system consists of two circuits:

- **Pulmonary Loop**

  The pulmonary loop transports the blood between the heart and the lungs. The de-oxygenated blood is transported to the heart though vena-cava via a network of veins and cavaliers. The blood enters the right atrium of the heart and is pumped from right ventricle to the lungs via pulmonary artery. The lungs replenish the blood with oxygen by respiration. The oxygenated blood from the lungs enters the left atrium via the pulmonary vein. The aorta pumps blood from the left ventricle to various body organs. This cycle is repeated 60-100 times per minute for a normal

human being in the resting state.

- **Systemic Loop**

  The oxygenated blood is pumped from the left ventricle to different body parts via various branches of aorta. The descending thoracic aorta takes the blood to the lower parts of the body, while the carotid artery provides blood to the head and the brain. The left and the right subclavian artery provides blood to the pectoral limbs.

3.1.2 Conduction System of the Heart

A heartbeat is generated by electrical signals which travel through conduction pathways. The heart has a complex conduction system with varying velocities in different fibers. The difference in conduction velocities helps achieve a prolonged action potential for a strong and well-timed contraction of heart muscles, forming a heartbeat as we know. Unlike nerve cell action potential, which peaks for about 1-2ms, cardiac action potential has a prolonged plateau lasting about 250ms. The absolute refractory period lasts about 200ms and the relative refractory period lasts about 50ms. The extended refractory periods prevent the heart from undergoing premature contractions.

The electrical signals start in the Sino-Atrial Node (SAN), which is located at the upper wall of the right atrium, close to the opening of the superior vena cava. The Sino-Atrial node is also known as the heart's pacemaker, as it primarily governs the heart rate. The action potential from the SA node causes atrial depolarization leading to atrial contraction. The action potential from the SA node travels to the right atrium via Bachmann's Bundle. Simultaneously, the Action potential from the SA node reaches Atrioventricular Node (AVN) via the Internodal Pathway. The AV node is situated at the base of the right atrium near the interventricular septum. The Action potential continues from the AV node to the Bundle of His, which is located within the interventricular septum. The Left and Right Bundle Branches follow the Bundle of His. The action potential from the Left and Right Bundle branches reaches Purkinje fibers, which causes ventricular depolarization, leading to ventricular muscle contraction. Purkinje fibers are in the subendocardial surface of the ventricle wall.

Each node is capable of generating independent action potential and functions independently. SA Node provides a resting heart rate of 60-100 bpm. In the absence of an SA node, the AV node may provide a heart rate of 40-60bpm. In the absence of both SA and AV nodes, a Bundle of His and Purkinje Fibre may provide a heart rate of 20-40 bpm. This step ladder fashion prevents the heart from failing altogether if a node fails.

3.1.3 Cardiac Cycle

The cardiac cycle refers to all the events that occur from the beginning of one heartbeat to the beginning of the next. It can be divided into two parts:
Systole – a period of contraction due to rapid depolarization.
Diastole – a period of repolarization and relaxation.

The cardiac cycle can be divided into four stages:

1. **Atrial systole** – Atrial contraction forces an additional small amount of blood into relaxed ventricles. Atrial systole lasts about 0.1 seconds. Heart sound atrial gallop or S4 is heard during this phase.

2. **Atrial Diastole** - overlaps with Ventricular systole and lasts about 0.7 seconds.

3. **Ventricular systole** – Both the ventricles contract. The entire phase lasts about 0.3 seconds.

    - First phase – As the contraction beings, the pressure inside the heart rises, which leads to the closing of AV valves, which produces heart sound S1, also known as 'lub.'
    - Second phase – The pressure further increases, which leads to the opening of semilunar valves, and the blood is ejected out.

4. **Ventricular diastole** - The entire phase lasts about 0.5 seconds. It can further be classified into early and late phases.

- Early - As the pressure in the ventricle drops, the backward flow of blood forces the closure of semilunar valves (aortic and pulmonary valves), which produces heart sound S2, also known as 'dub.'
- Late - all four heart chambers are in diastole, and blood flows passively into the ventricles.

3.1.4 Einthoven's Triangle and ECG Leads

The heart acts as an electrical dipole. The strength and orientation of this dipole change with each beat. This change is measured and recorded by an ECG. Einthoven's triangle is an imaginary triangle composed of leads I, II, and III. These leads are formed by three bi-polar electrodes on the Left Arm (LA), Right Arm (RA), and Left Leg (LL).

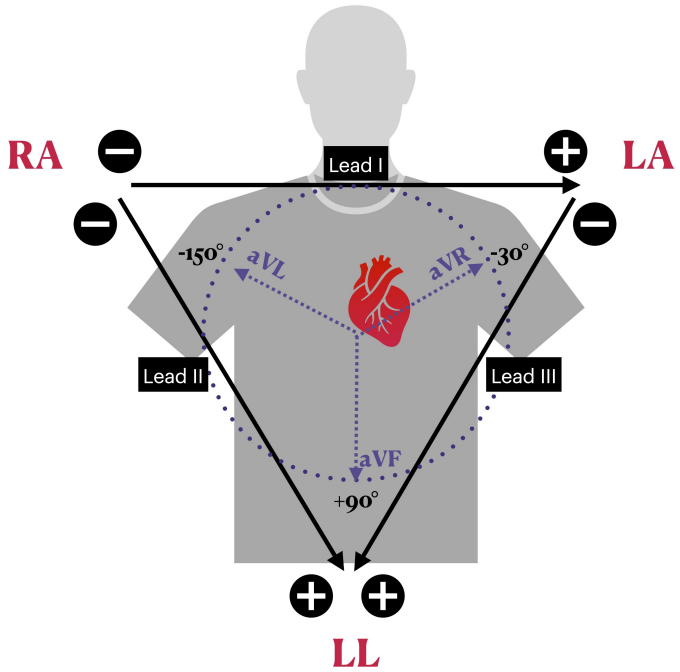

Figure 3.1: Einthoven's Triangle

1. **Standard Limb Leads**

   The voltage difference between the three electrodes provides us with lead I, III, and III.

Lead I = RA - LA

Lead II = RA − LL

Lead III = LA -LL

2. **Augmented Limb Leads**

   These are unipolar and are referenced against a combination of other limb electrodes.

   aVL = LA − (RA + LL)/2

   aVF = LL − (LA + RA)/2

   aVR = RA − (LA + LL)/2

3. **Precordial Leads**

   There are six precordial leads. These are composed of 6 electrodes on the surface of the chest, from the sternum to the posterior directions. These are labeled as V1, V2, V3, V4, V5, and V6.

   The left ventricle can be divided into four sections. The following chest leads are used to observe these sections –

   - Lateral Wall – I, aVL, aVR
   - Anterior Wall – V3, V4
   - Interventricular septum – V1, V2
   - Inferior wall – II, III and aVF
   - Anterolateral wall – V5, V6

4. **Right Leg Electrode(RL/N)**

   Neutral electrode. Removes the artifact

A 12 lead ECG setup has 10 electrodes.

3.1.5 ECG signal waveform and Heart Anomalies

All these leads are used to analyze the performance of the heart and detect possible anomalies. An abnormal graph may indicate the presence of anomalies. Since different leads represent different directions and axis, they are used to locate the possible location of abnormal conduction in the heart. The ECG is of great clinical significance for detecting and diagnosing an issue.

The figure represents a normal heartbeat. A heartbeat on ECG has three major deflections. Firstly, the P wave, which is generated by atrial depolarization, causes contraction of the atrial. The QRS complex is generated by ventricular depolarization, causing ventricular contraction. The T waves indicate ventricular repolarization. A slight elevation immediately after the T wave represents U waves. U waves are more dominantly seen in precordial leads V2 and V3. There may be numerous reasons for upright or inverted U waves. Upright U waves may be due to Hypokalemia, Sinus bradycardia, Hyperthyroidism, etc. Inverted U waves may be present with myocardial infarction or ischemia. U waves are often ischemic.

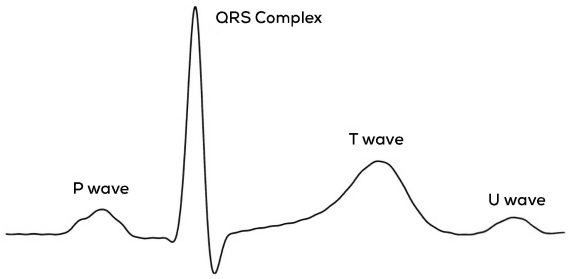

Figure 3.2: A Normal ECG Beat

1. **Normal sinus rhythm**

   Regular Rhythm with an atrial rate between 60-100bpm. Prominent P waves, QRS complex, and T waves. The interval between complexes is regular and easily discernable. Figure 3.1 shows normal signals from ECG ID 9, PTB-XL Dataset.

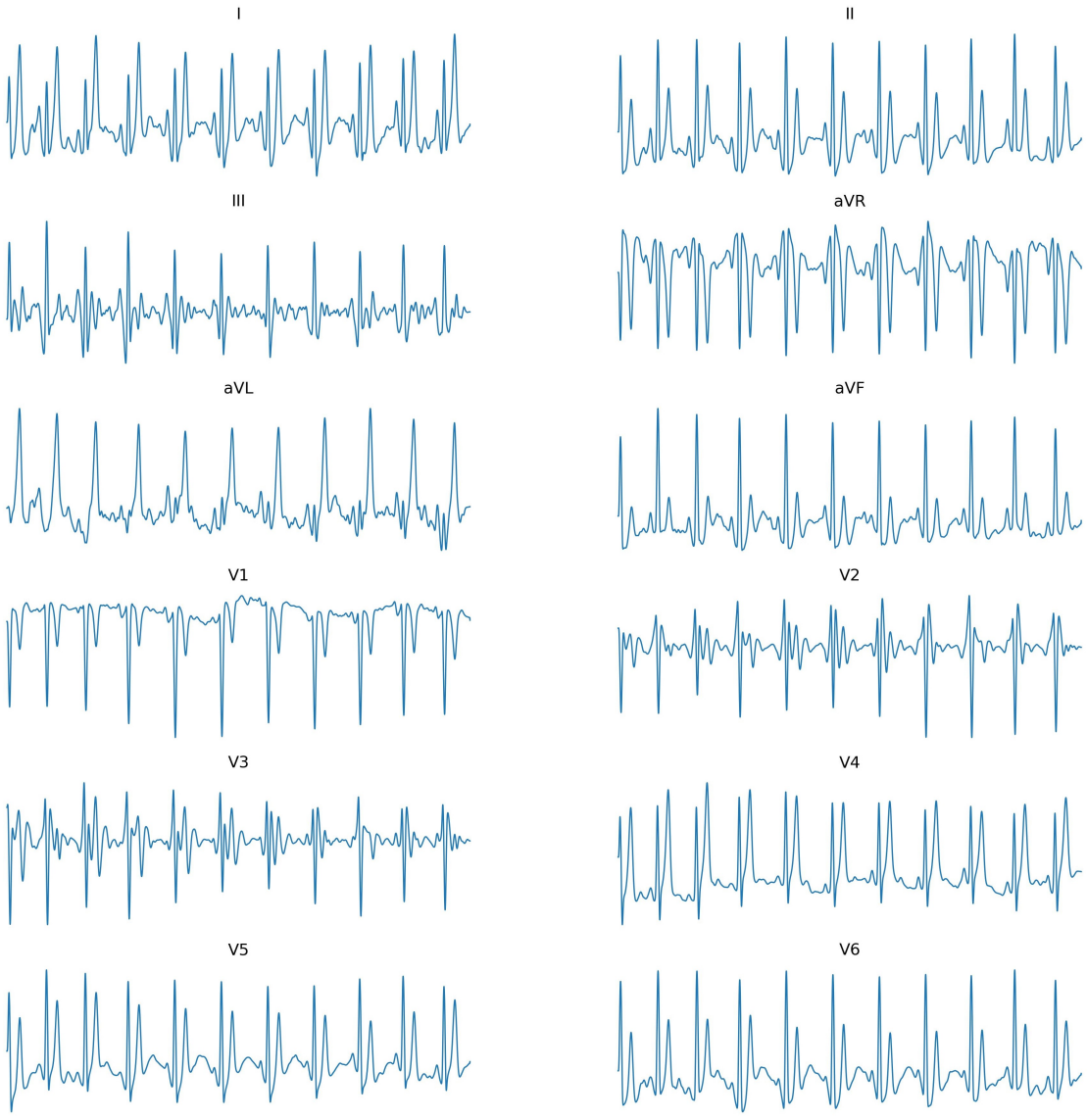

Figure 3.3: Signals for All Leads - ECG ID 9, PTB-XL Dataset

2. **Atrial Fibrillation**

   Irregular rhythm with atrial rate 350-600bpm and ventricular rate 120-200bpm. P waves are absent or irregular. Have normal QRS complex but, PR interval may not be measurable.

   Common Causes: Chronic obstructive pulmonary disease, heart failure, ischemia or hypertension [50].

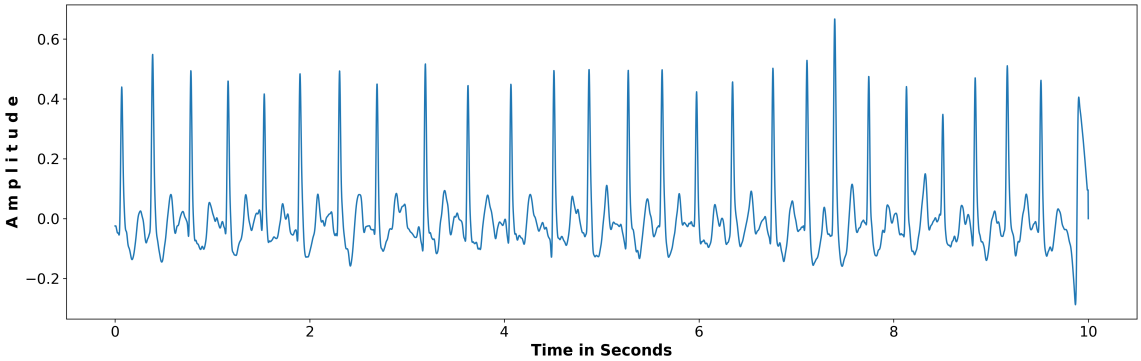

Figure 3.4: Atrial Fibrillation - ECG ID 351, Lead I, PTB-XL Dataset

3. **Atrial Flutter**

   Regular rhythm with fluttery and undefinable or coarse P waves forming a sawtooth pattern. Atrial rate 250-350bpm.It is characterized by several atrial contractions one ventricular contraction.

   Common Causes: Idiopathic, heart failure, pulmonary embolism, inferior wall myocardial infraction or certain drugs [51].

4. **Sinus Tachycardia**

   Normal Rhythm with normal shape and size of R, QRS and T waves. P waves may merge with T waves at faster atrial rates. Atrial rate between 100 to 200. Common Causes: Exercise, anxiety or response to pain [52].

5. **Sinus Bradycardia**

   Normal rhythm with normal shape and size of R, QRS and T waves. Atrial rate is lower than 60bpm. Common Causes: Aging, increased vagal tone after vomiting,

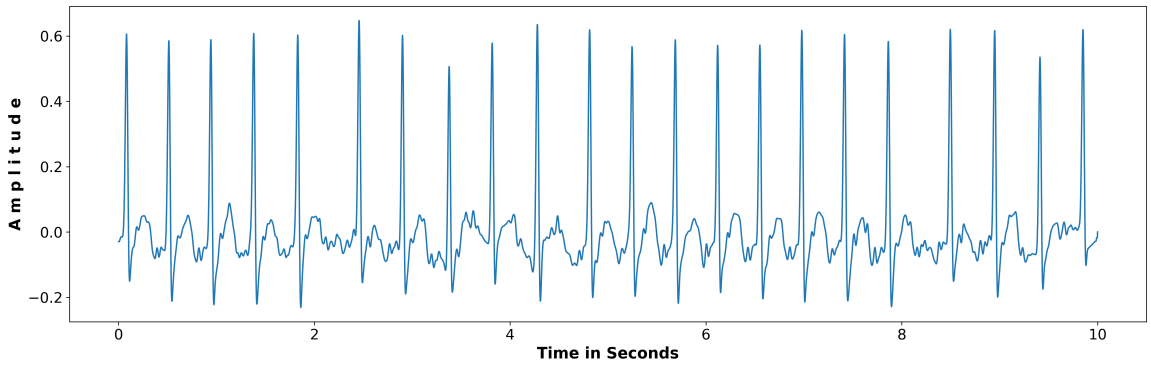

Figure 3.5: Atrial Flutter - ECG ID 18, Lead I, PTB-XL Dataset

emotional stress, extreme fatigue, defecation, etc. Physiological Response: fatigue, shortness of breath, dizziness or fainting if bpm is too low [53]

6. **Ventricular tachycardia**

   Regular rhythm with Ventricular rate 100-250bpm. Wide QRS complex with overlapping P wave. It is characterized by irregular electrical signals to ventricles. RSR or bunny ear pattern in V1 or V2.

   Common Causes: Idiopathic, myocardial infraction, myocardial ischemia, hypercalcemia or hyperkalemia or coronary artery diseases [54].

7. **Ventricular Fibrillation**

   Irregular rhythm with no visible P waves and wide QRS complex, and ventricular rate over 400 bpm.

   Common Causes: Myocardial infraction, myocardial ischemia, alkalosis, aortic stenosis, hypercalcemia, or hyperkalemia or drugs affecting QT duration [55].

8. **First Degree AV blocks**

   Possible Causes: Regular atrial and ventricular rhythm. Normal QRS complex. PR interval greater than 0.20 seconds. First Degree AV blocks are often asymptomatic, but detectable on ECG.

   Common Causes: Natural aging, low thyroid levels, inferior myocardial infraction, hypercalcemia or hyperkalemia, ischemia or increased vagal tone [56].

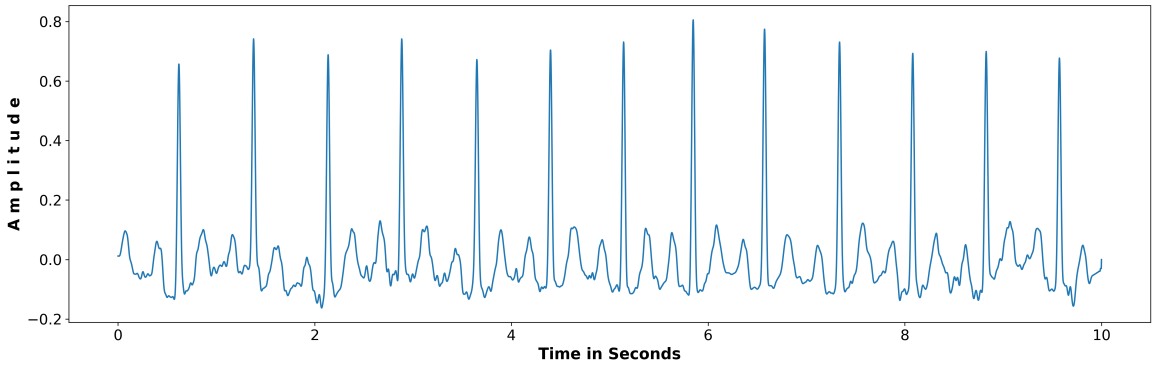

Figure 3.6: First Degree AV Block - ECG ID 2648, Lead II, PTB-XL Dataset

9. **Second Degree AV block Mobitz I (Wenckeback)**

   Regular atrial rhythm with irregular ventricular rhythm. Some P waves may not be followed by a QRS complex. Caused due to supression of AV condution. Some impulses are completely blocked.

   Common Causes: Natural aging, increased vagal tone, anterior wall myocardial infraction, acute myocarditis, coronary artery disease, beta-blockers or digoxin [57][58].

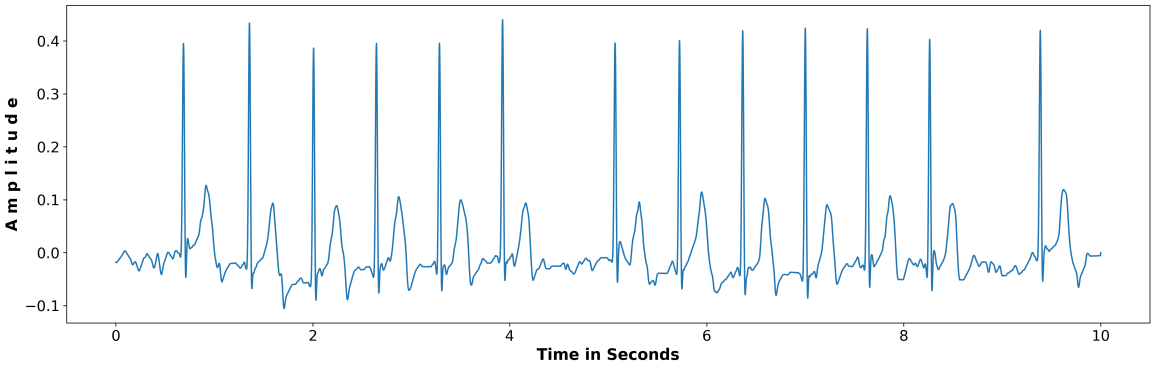

Figure 3.7: Second Degree AV Block - ECG ID 14009, Lead V2, PTB-XL Dataset

10. **Third Degree AV Block**

    Also known as complete heart block. This occurs when the action potentials are completely blocked at AV node. Correlation between P wave and the QRS complex is lost. Cardiac output is diminished.

    Common Causes: Idiopathic, drug toxicity - digoxin, acute ischmic heart disease, fibrosis, electrolyte imbalance or post-operative heart block [59].

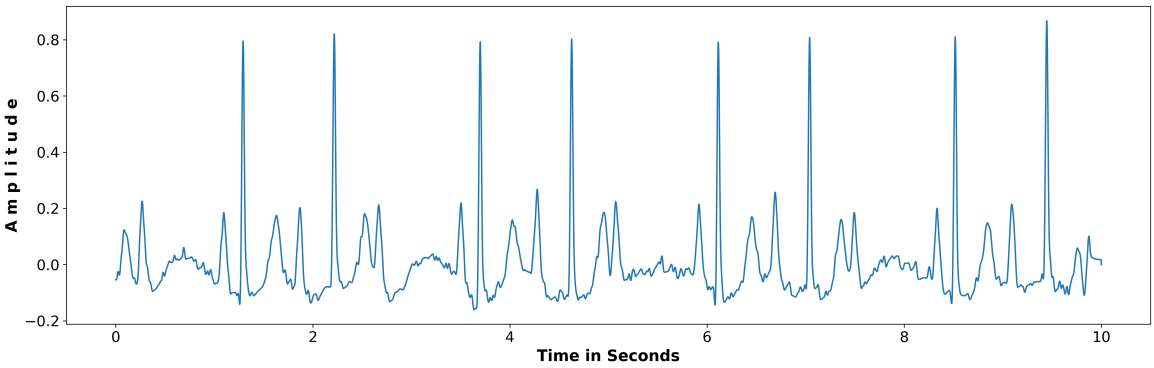

Figure 3.8: Third Degree AV Block - ECG ID 10505, Lead I, PTB-XL Dataset

11. **Paced Beats**

    Paced beats are characterized by spikes. In atrial pacing, sharp spike of about 2ms is followed by a P wave. In ventricular pacing, sharp spike of about 2ms is followed by a QRS complex. Pacing could be either or both.

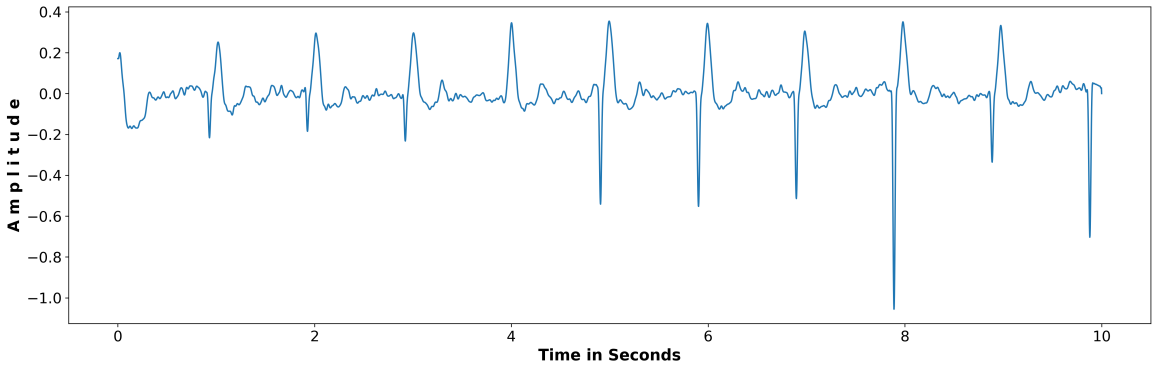

Figure 3.9: Paced Signal - ECG ID 498, Lead I, PTB-XL Dataset

12. **Complete Left Bundle Branch Block**

    Left ventricle contracts a little later than normal. QRS duration is greater than 120ms. Percordial Lead V5/V6 has notched R wave and Q wave us absent. Lead V1 have large S wave and small QS wave. A left bundle branch block is usually associated with underlying heart disease.

    Common causes: aging, myocardial infraction, myocardium, cardiomyopathy, hypertension or coronary artery disease [60] [61].

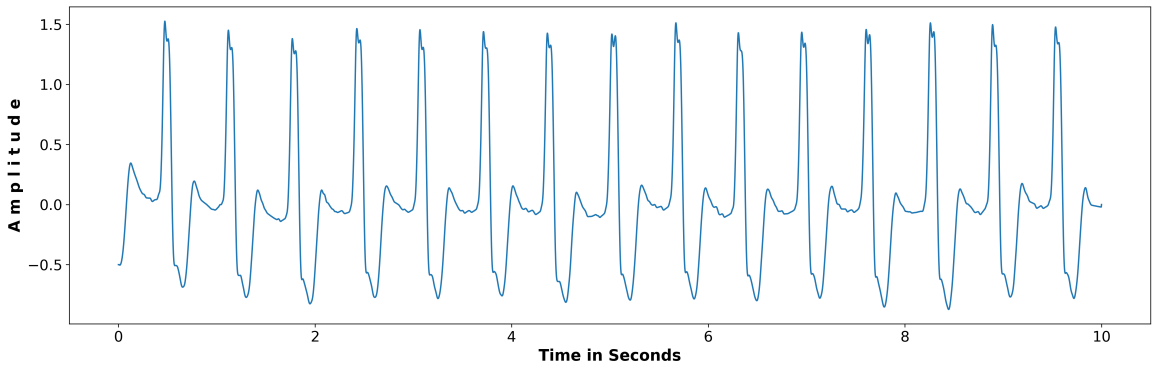

Figure 3.10: Complete Left Bundle Branch Block Lead V5 ECG ID 180, PTB-XL Dataset

13. **Complete Right Bundle Branch Block**

Precordial Lead V1 and V2 present RSR' (bunny ear) pattern. QRS duration is greater than 120 ms. Wide S wave in lead I, with duration more than 40 ms. Bundle branch blocks are more common in older adults.

Common Causes: aging, chronic obstructive pulmonary disease, pulmonary embolism, myocarditis, Lenegre's or Lev's disease [62].

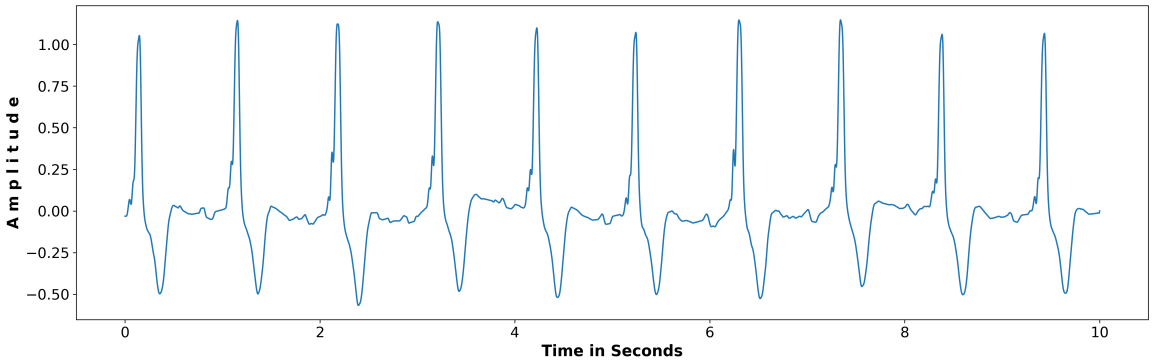

Figure 3.11: Complete Right Bundle Branch Block Lead V1 ECG ID 621, PTB-XL Dataset

3.2 Internet of Things

Over the years, the IoT has been described in numerous ways, but the concept remains more or less the same. The IoT describes a low-powered device that has sensors, processing ability, and technology to communicate with other devices. The IoT devices are often powered by a battery. Common IoT devices include smart bulbs, smart home assistant devices, single board computers, embedded devices or sensors.

The architecture of IoT varies depending upon the use case scenarios, but their architecture can broadly be classified into three layers, from highest to lowest level as:

1. **Application Layer** - comprises software and interface that drives the sensors, user interface, and data processing or forwarding features. This layer can be customized to suit various applications and services.

2. **Network Layer** - sits under the application layer and governs the communication and transmission activities. It handles the information packets and protocols such as BLE, Ethernet or WiFi.

3. **Physical or Perception layer** - is responsible for gathering the information about the surrounding using different sensors such as - light sensor, RFID, humidity sensor or Near Field Communication.

3.3 Biomedical Applications of IoT

In the past few years, there has been a tremendous increase in the number of IoT devices such as smartwatches, virtual assistants, smart plugs/switches, and healthcare devices like smart insulin pumps. With LTE and older technologies, IoT devices were bottlenecked by finite spectrum and data capabilities. 5G offers a larger spectrum, better speed, and lower latency [1]. This is believed to provide an impetus for the rapid growth of IoT devices. There were more than 26 billion connected devices in 2019. The number is expected to reach 75 billion by 2025 and over 500 billion by 2030 [2] [3].

IoT has numerous healthcare applications in a clinical and non-clinical settings. With the increase in operational efficiency of sensors and decrease in manufacturing cost of

sensors, IoT has begun transforming the healthcare industry. The use of IoT has helped in saving costs in healthcare. Tests, which required hospital and pathology visits a few years back, can now be done conveniently at home. Some common applications are:

1. Connected inhalers
2. Ingestible sensors
3. Heart-rate monitoring
4. Glucose monitoring
5. Blood oxygen monitoring
6. Remote patient monitoring
7. Air quality sensors
8. Biometrics scanners
9. Thermal detection
10. Pathogen detection
11. Contact tracing
12. Bluetooth Blood Coagulation Testing

3.4 IoT and Artificial Intelligence

More IoT devices are being incorporated into our lives. This has to lead to new challenges, the solution to which often lies in AI. AI provides a range of features in handling the collected data. The AI models are tailored to suit the application and user needs. Some advantages of using AI are -

1. Enhanced operational efficiency - Conventional methods and manual analysis may not be able to detect underlying patterns and features which could otherwise be

tuned for optimal efficiency. AI helps in detecting those patterns and tuning the parameters for efficiency.

2. Eliminates Costly Unplanned Downtime - Automation of several processes could be accomplished with AI. This helps in both avoiding and handling unplanned service interruptions.

3. Improving Precision Cost - use of AI can help with the evaluation of data and various parameters, some of which would not have been possible by conventional methods. This ensures that an appreciable level of performance and precision is maintained. AI can also filter out the relevant data for further analysis on edge or cloud nodes.

4. Maintenance and automation - AI could be trained to look for parameters indicating failing components or devices like a dimming or flickering LED. In such cases, devices could be marked for maintenance before it actually fails. This helps in saving downtime and maintenance costs.

5. Increased Scalability - In terms of computational power requirements, AI has the capacity to bring down the computational needs to accomplish a task. Combined with the advantages discussed above, AI helps in overall greater scalability of device infrastructure and framework.

3.5 TensorFlow and Keras

TensorFlow[63] is a free and open-source software library for machine learning. It was developed by the Google Brain team and initially released in 2015. Since then, it has come a long way, adding features and even expanding support to embedded devices.

Tensor - A tensor is a vector or a matrix that represents the data - input, output, and intermediate. In simple words, a tensor is a data container. Using Tensor for operations like multiplication or addition in a large data is computationally less demanding than conventional 'for' loops.

Graph - In TensorFlow, all the operations are computed inside a graph. Connected tensors do the computations inside a graph. It uses a graph to represent a function's

computations. Using the graphs provides the advantage of running the model on multiple platforms - CPU, GPU, TPU, embedded, etc. The graphs could also be saved for later use. The graphs could be optimized and transformed to suit the platform. They also support distribution execution.

Keras[64] is a deep learning API written in Python, running on top of the machine learning platform TensorFlow.

3.6 TensorFlow Lite and Raspberry Pi

TensorFlow Lite is an open-source deep learning framework for on-device inference[35]. With the increase in wireless technology, more applications are being explored for IoT devices requiring sophisticated deep learning models on edge and embedded devices.

TensorFlow Lite converts the TensorFlow/Keras saved model to a Flatbuffer. A Flatbuffer is an serialization cross-platform library for C, C++, C, Java or Python. The Flatbuffer could be executed with much fewer resources. During the conversion process, quantization optimizations could be used to reduce the model's size. Following are the significant advantages of using Tensorflow Lite or FlatBuffer:

1. *Memory efficiency* - requires little to no additional memory allocations apart from the buffer. This increases the speed and efficiency and decreases the latency of the system.

2. *Cross-Platform Code* - The flat buffer could be run on any platform without changes. This helps greatly in coding for embedded devices as a flat buffer generated on a powerful machine could be transferred to the embedded device.

3. *Parsing/unpacking not required* - The flat buffers are ready to be used. The only piece of code they need is an interpreter. This schematic is great for embedded devices often constrained by memory and cannot have multiple libraries.

4. *Small code size* - Since it is free from dependencies, the code size is significantly smaller. A 16-bit or int8 flat buffer could also be generated to reduce the size further to fit a constrained memory.

3.7 Convolutional Neural Network

A convolution is a mathematical operation on two functions that produces a third function. The convolutions expression 3.1 is an integral expression that expresses the amount of overlap of one function g, as it is shifted over another function f. Convolution has applications in probability, image processing, signal processing or statistics.

$$(f^{*}g)\,(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\tau' \tag{3.1}$$

τ' = first derivative of $g(\tau)$

$(f^{*}g)\,(t)$ = convoluted functions

$g(\tau)$ = convolution of $f(\tau)$

t = real number variable of functions f and g

Although multiple variants of CNN are available - one dimensional, two dimensional, three dimensional, CNN coupled with LSTM, etc., we will discuss the one relevant to this project, which is 1D CNN.

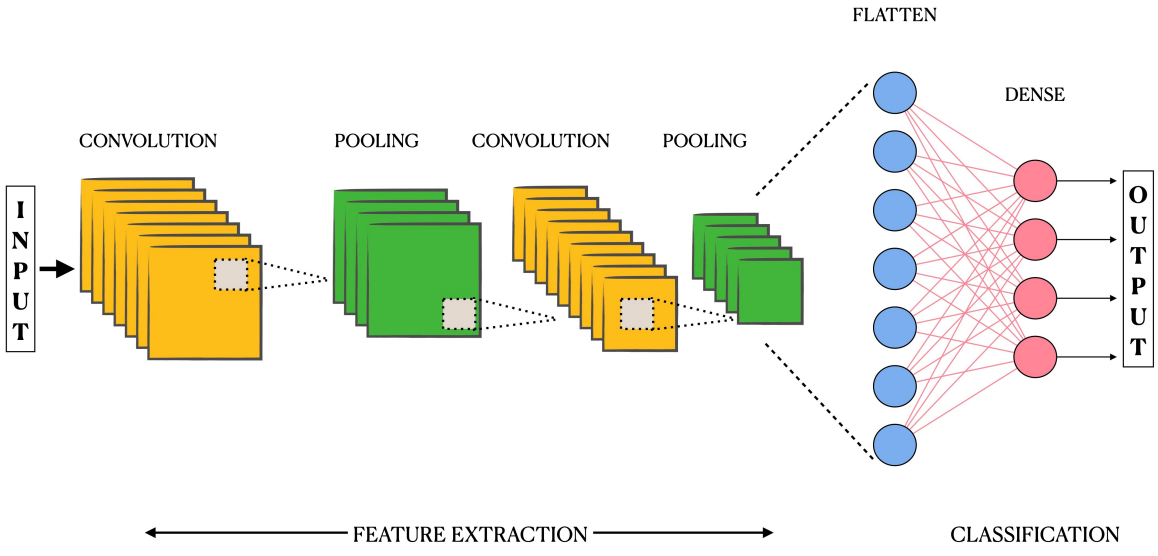

Figure 3.12: A Typical Convolutional Neural Network

One Dimensional Convolutional Neural Network (CNN) is extensively used for time series classification and prediction problems. Convolutional Neural Networks are regu-

larized variants of multi-layer perception. A CNN has an input layer, an output layer, and several hidden layers and parameters, enabling it to learn complicated patterns. A CNN was first created by Professor Yann LeCunn of Bell Labs in the 1990s. Mathematically, convolution is the way of combining two signals to form a third signal. The inputs of a CNN are convoluted to generate a feature map. The CNN uses the same weights and biases for all neurons. Different filters could be created for each layer to capture different aspects of the signal. The kernel is multiplied with the input data (dot product) to enhance the desired features, such as edges.

Basic Architecture of Convolutional Neural Network (CNN)

1. **Convolutional layers**

   This is the layer where primary computation occurs. A convolutional layer has specific input parameters which could be tuned for efficiency or performance. Some of the commonly used parameters are:

   Kernel - The kernel is used to extract the features from the data. The kernel matrix is usually much smaller than the input data. It is traversed and multiplied with the input data to enhance the desired features. Although the size of the kernel could be set to anything, the majority of deep-learning practitioners commonly use sizes 3 or 5 to ensure that local features are extracted.

   Filter - Multiple kernels stacked together to form a 3D structure are termed filters. Although the term filter or kernel is often used invariably from an application point of view, they are not the same theoretically.

   Strides - The kernel moves over the input data by stride value. Stride is the distance by which the kernel moves over the input matrix.

   Padding - Padding is used when the input image fails to fit precisely in the kernel. Padding can be done in a few ways:

   Full Padding - The zeros are added to the borders or edge of the signal/input data. It increases the dimension of the output data. This type of

padding is often used in image processing to center the image.

Valid Padding - No padding is done. The areas of input that the specified kernel and stride cannot cover are dropped out.

Same Padding - This padding is used to ensure that input and output dimensions are the same. The kernel and the stride can cover the input fully. For stride 1, the output is the same as that of the input.

Input shape - This parameter tells the shape of the input. For example, a time-series signal with 1000 data points and 12 similar channels will have an input shape of (1000,12).

2. **Activation Layers**

   Each layer in the neural network has many nodes/neurons which compute the weighted average of its inputs, and these weighted averages are passed through a non-linear activation function. There is a variety of activation functions available to choose from. Some popularly used functions are:

   (a) Leaky ReLu

   (b) Softmax

   (c) Tanh

   (d) Sigmoid

   (e) ReLu

   (f) Linear

   Over the past few years, the use of ReLu has seen a substantial increase in the machine learning community. A ReLu activation function produces a zero output for negative inputs, whereas the output of positive inputs is preserved. A leaky ReLu, on the other hand, accommodates outputs for negative values within the set threshold. Compared with many other complex activation functions, ReLu is computationally less demanding, which is probably a significant contributor to its success. It also makes ReLu approximately six times faster than *tanh* and *sigmoid*.

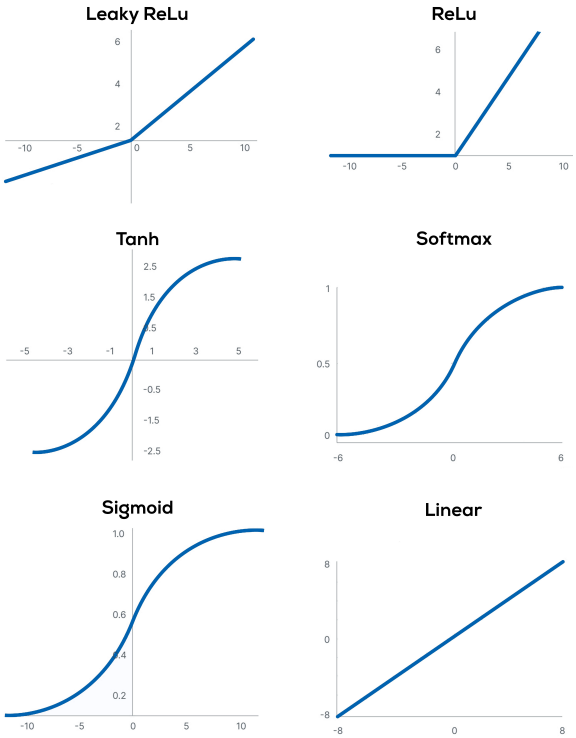

Figure 3.13: Popular Activation Functions

3. **Pooling Layers**

   The pooling layers use the downsize of the output received from the convolutional layer. Say the output of the convolutional layer was (1000,1), and the pooling layer is set with parameter in Keras (2,1) or to half the dimension, it will reduce and produce the output with dimension (500,1). Pooling could be done in numerous ways depending upon the features we are trying to extract. Pooling helps in reducing the computational cost for further layers. The most popular pooling methods are -

   Average Pooling - the average value is calculated from the feature map, which is used to create the downsized output.

   Max Pooling - the maximum value is calculated from the feature map used to create the downsized output.

   Global Max Pooling - Gives the maximum value among all the pooling layers. As, they don't have learnable parameters, they are less prone to overfitting.

   Global Average Pooling It takes the feature map of the last convolutional layer.

It is used to substitute fully connected layers or flatten layer, as it generates one feature map for each category. Therefore, it forces feature map to categories

4. **Dense layers or fully connected layers**

   The features generated using the convolutional layers are passed to the Dense or fully connected layers. The fully connected layer is connected to all the features generated in the preceding layer. Each node has a distinct weight and bias. These dense layers are used to classify based on the input received from the convolutions layers.

5. **Flatten**

   A flattened layer reshapes the tensors to have a shape equal to the total number of elements contained in the tenors. Flatten helps in converting the multi-dimensional arrays to one-dimensional arrays. Flatten is often used right before the dense layers.

6. **Dropout Layers**

   The dropout layer is used to handle the issue of over-fitting, which is a pervasive issue with training data with relatively limited samples. Dropout randomly sets the outgoing edges of the hidden neurons to zero while updating each training phase. In other words, it nullifies the contribution of some neurons to the next training iteration. If dropout is not applied, the samples in the first training batch influence the entire learning process in a biased and disproportionately high manner. It prevents the model from learning features that come up in later samples.

3.7.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are a type of RNN (Recurrent Neural Networks) capable of learning long-term dependencies in sequence/time-series prediction problems. They are often used in classification and prediction problems.

Conventional Recurrent Neural Network (RNN) cannot remember long-term dependencies and are prone to vanishing/exploding gradient problems. LSTMs were designed to overcome these issues.

LSTM Architecture

An LSTM unit consists of an input gate, an output gate, and a forget gate.

1. **Input Gate** The input receives the input and determines its importance. The previous hidden state and the current input are passed into a sigmoid function to decide which values need to be updated based on their importance. The current state and the hidden input are passed through the tanh function. The dot product of tanh and the sigmoid outputs is determined to decide whether the information needs to be stored or not.

2. **Forget Gate** The forget gate decides on the information that needs to be kept and thrown. The current and previous state information is passed through a sigmoid function. Based on the output of the sigmoid function, the information is kept if the output is closer to 1 or discarded when it's closer to 0.

3. **Cell State** Unlike the hidden units, the cell state provides the ability to store long-term memory. This memory may not be from the immediately previous event. The cell state is calculated by multiplication with forget vector. The input gate's output is multiplied to obtain/update the cell state.

4. **Output Gate** The information from the previous hidden state and the current input is passed into a sigmoid function. The sigmoid function generates an output between 0 and 1. Meanwhile, the modified cell state is passed through the tanh, multiplied with the sigmoid output to decide on the hidden state information.

   The output gate provides an output based on the current and previously stored information.

Mathematicall LSTM could be represented as -

$$
\begin{aligned}
i_t &= \sigma\left(W_i h_{t-1} + U_i x_t + b_i\right) \\
f_t &= \sigma\left(W_f h_{t-1} + U_f x_t + b_f\right) \\
o_t &= \sigma\left(W_o h_{t-1} + U_o x_t + b_o\right) \\
\widetilde{c}_t &= \tanh\left(W h_{t-1} + U x_t + b\right) \\
c_t &= f_t * c_{t-1} + i_t * \widetilde{c}_t \\
h_t &= o_t * \tanh\left(c_t\right) \\
y_t &= h_t
\end{aligned}
\tag{3.2}
$$

f_t = forget gate

i_t= input gate

c_t=cell state

o_t = output gate

h_t= hidden state

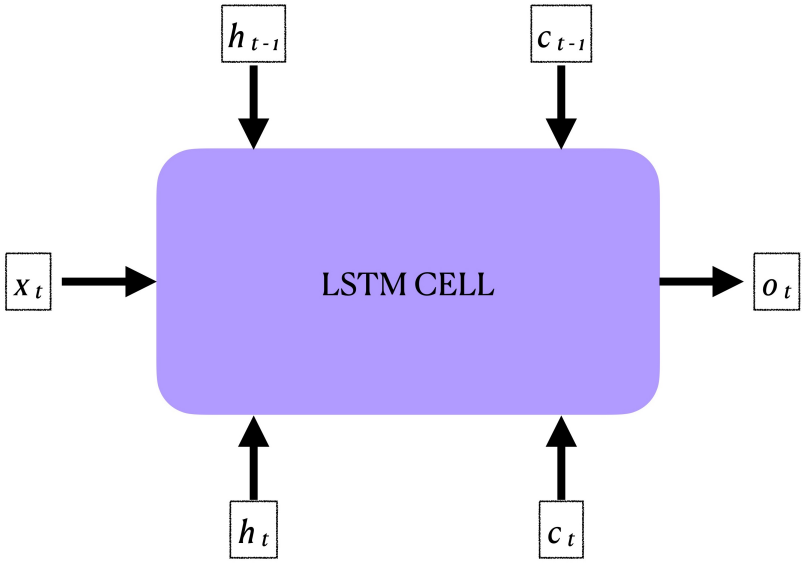

Figure 3.14: A LSTM Cell

3.8 Public ECG Databases

There are numerous public ECG databases available for analysis. However, most of them are not sufficiently large for developing efficient training models. PTB-XL [15] [14] turns out to be a viable choice for this project due to a large number of records and patients, which provides sufficiently large variations in time series for the model to learn the patterns. Although attempts have been made to pool the data from all the databases, that adds more variations due to different sensors, bit-rate, sampling rate, etc. Therefore, PTB-XL seems to be the ideal choice among some of the databases mentioned in the table below.

Database	Year	Subjects	Records	Sampling Rate	No. of Leads
AHA [65]	1980	155	155	250	12
MIT-BIH [66]	1990	48	48	360	12
PTB [67]	1995	290	549	10,000	12+3
INCART [68]	2007	32	75	257	12
THEW(AMI) [69]	2012	93	160	200	3
PhysioBC [70]	2016	91	182	1000	12+3
PTB-XL [15]	2020	18885	21837	100/500	12

Table 3.1: Open Access ECG Databases

3.9 Raspberry Pi CM4 by Seeed - reTerminal

Raspberry Pi Compute Module (CM) 4 is a compact form factor for deeply embedded applications. It has an ARM Cortex-A72 quad-core processor with 4GB of RAM and 32GB of embedded Multi-Media Card (eMMC) onboard memory. The eMMC memory provides faster startup times and better read-write performance. It has a built-in accelerometer and light sensor. The reTerminal from Seeed comes with an expanded IO interface and 5-inch IPS capacitive touch display, which makes it great for prototyping projects like ours. The reTerminal comes with a full 40-Pin General Purpose Input Output (GPIO) interface which could be used to connect various sensors in the future expansion of this project. It also has an ethernet port and dual-band WiFi which clients can use to access the prediction data. Though reTerminal doesn't have a battery, it could be operated using a power bank, which adds to its portability.

Raspberry Pi is capable of running a Linux-based operating system called Raspbian, which provides leverage for executing existing Linux programs and libraries. Therefore, machine learning programs, database management systems like SQL could easily be deployed on raspberry pi.

In this project, we have hosted an Apache web server along with a few PHP pages to display the predictions from our TensorFlow Lite model.

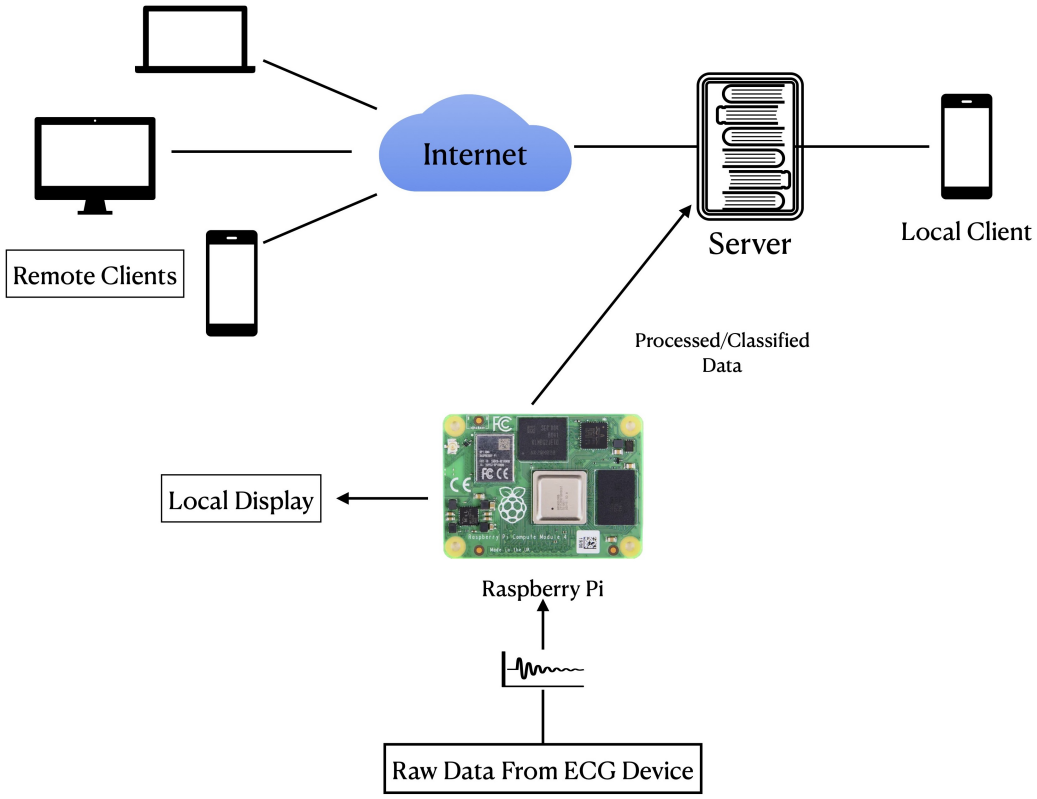

Figure 3.15: General Architecture

Chapter 4

Model and Prototype Design

4.1 Problem Overview

The number of IoT devices and their applications is increasing rapidly. Some credit for this rapid growth goes to the advent of 5G networks which are designed to support IoT devices with their lower latency, larger frequency spectrum, and better data transfer speeds. This has paved the way for better real-time applications of IoT devices. Despite the improvements in communications technology, the core design issue of IoT devices remains the same i.e., small battery size and limited processing power. Some sensors have high sampling rates like ECG, EEG, and nerve activity sensors. Computing such data locally could be inefficient and power-intensive for IoT devices. Therefore many devices such as 'ECG Recorder with AI Analysis' prefer to record the data and process it later on a much powerful machine. Such an approach is not suitable for real-time applications. This gave rise to the concept of offloading computationally demanding data to nearby node/s, where data could be stored, processed, and analysed in real-time.

In a multi-user multi-sensory home and hospital environment, there is constant monitoring of different body vitals. These sensors record a huge amount of data that needs to be interpreted by physicians. In order to provide some assistance for reliably interpreting the data by users/physicians, artificial intelligence is increasingly being explored. This calls for the development of new technologies and methods for optimizing the current health monitoring system for greater reliability, increased portability, and longer battery life. This has provided impetus towards more research in biomedical applications of mobile computing, edge computing, edge-cloud computing, distributed computing, etc. But,

the majority of these require transmitting large data to a much powerful system either physically or over wifi, which is a time and power-consuming process.

A survey paper published by O. Shahryari[9] compares the different architectures for offloading and data processing. Many of such architectures managed to attain the goal of acceptable delay and computational needs, but most of them were expensive to implement and not suitable for small-scale personal, home, or hospital use. Also, the majority of the frameworks are not easily expandable and might not be suited for future needs.

Additionally, implications of emerging technology like TensorFlow Lite have not yet been explored much in this regard. The efficiency of TensorFlow Lite models could vary and needs to be studied according to our use case scenario.

4.2 Prototype Design

From our problem statement, we know that reliability, portability, power, and cost-efficiency are the key issues for IoT devices. To handle large computational and processing requirements we developed an expandable framework that employs Artifical Intelligence (AI) on Raspberry Pi 4, for analysis and is capable of rendering the results to a web server for storage, display, and future analysis.

Firstly, we used Raspberry Pi 4 in our test bench. Raspberry Pi is a low-cost, credit-card-sized computer. It is one of the most widely used platforms for IoT devices due to its versatility. Raspberry Pi 4 comes with onboard Bluetooth/BLE 5.0, which can be used for communication with various other sensors. For now the raw data was read from the device memory but, it may also be acquired via wired connection using General Purpose Input Output (GPIO) pins. We coded for the Raspberry Pi to be able to receive the raw data and process it. We also programmed the Raspberry Pi 4 for sending the processed data to a web server.

Secondly, we primarily used TensorFlow and Keras for creating and training a neural network along with some supporting libraries like MatPlotLib, Pandas, Numpy, etc. In this project, we used both 1D -CNN and Long Short-Term Memory (LSTM) which is a type of Recurrent Neural Network (RNN). 1D-CNN and LSTM are used for the classification of time-series data. LSTM is a kind of RNN which are capable of learning long-term

dependencies. For ECG signals PTB-XL database was used. All the ECG recordings used are publicly available.

Thirdly, to train the network on a computer, ECG data from PTB-XL was processed and segmented. The data was then scaled and balanced as necessary. This would make our training data. This training data was then used to train the 1D-CNN and LSTM model. The trained TensorFlow model was deployed as a TensorFlow Lite model on Raspberry Pi 4. The TensorFlow Lite model is expected to have a similar or marginally reduced efficiency, but the exact effect needs to be studied accordingly. The TensorFlow model could be tweaked a bit in order to attain better results on the model deployed on Raspberry Pi 4. The optimum methods for modeling our network and implementing still need to be studied and experimented with. The Raspberry Pi 4, will independently classify the data and forward the results to a web page on a server.

Fourthly, The server is responsible for displaying the results. This server have the ability to connect to multiple sensors and devices, which would be advantageous for future expansions of this framework.

4.3 Implementation Process

4.3.1 Setting Up TensorFlow/Keras in Anaconda

As we have already discussed that TensorFlow/Keras are open source machine learning libraries. The process to set up the development environment is:

1. Downloading and installing Anaconda Navigator 3 from the website https://www.anaconda.com/
2. Creating a new development environment with Python 3.9.11
3. Installing TensorFlow 2.6, matplotlib, numpy, scikit-learn, wfdb and other dependencies.
4. Since, we are using a GPU for training we need to install Keras and TensorFlow GPU as well.

5. During the course of implementation we have found that version mismatch could break things, due to some features being deprecated or migrated.

4.3.2 Data Pre-Processing

The PTB-XL dataset has 21837 records from 18885 patients. Each record is 10 seconds long and has 12 channels [15]. Meta-data such as sex, weight, height, and diagnostic class is also available. Statements about the signal category are present in 'scp_codes.' The dataset contains more information about the signal labels, which is out of the scope of this work. Although most signals are high quality [54], some signals occasionally contain powerline noise, baseline drift, burst noise, and static noise. Therefore, removing these artifacts is crucial before implementing the machine learning model.

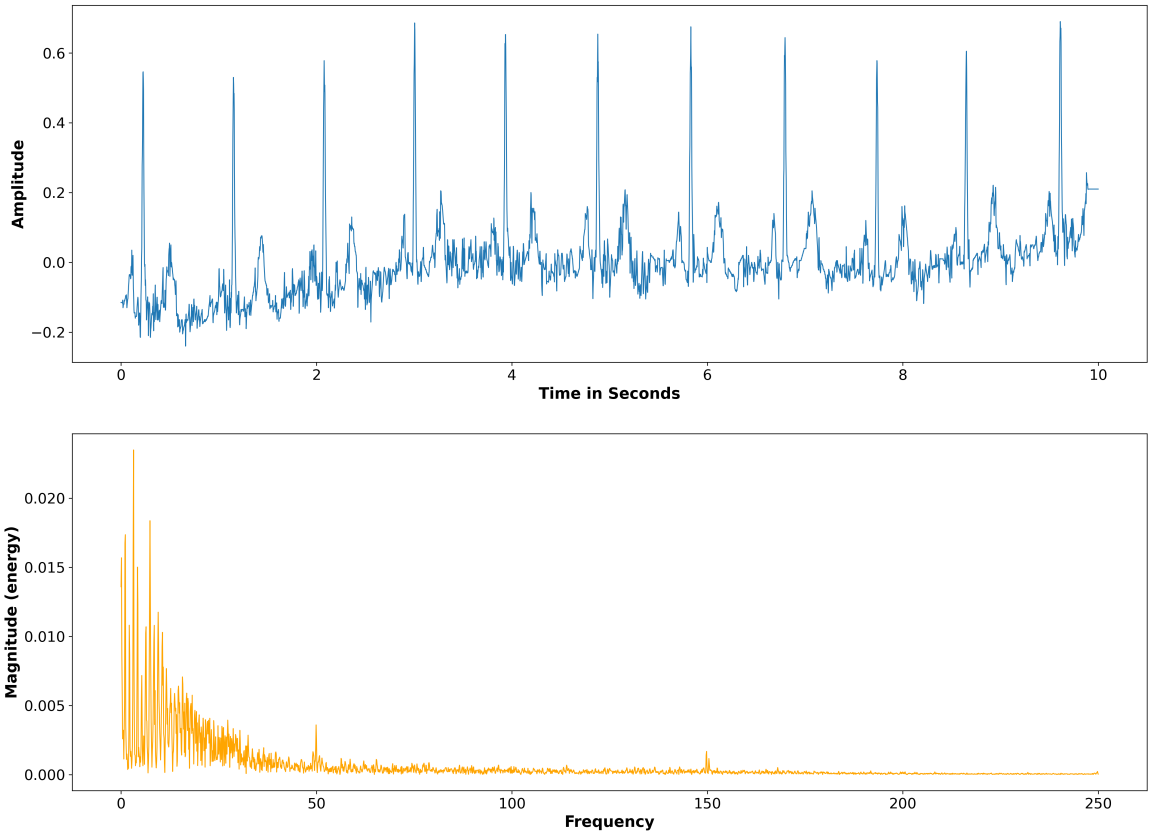

Figure 4.1: Raw Signal and Spectrogram for ECG ID 1, Lead I, PTB-XL

Powerline Noise Removal

Occasionally, some channels present a powerline noise of 50-150 Hz. It is important to note that powerline noise may be present in some channels while absent in others for a given patient. For removing powerline noise, we have used a low pass filter with a cutoff at 45Hz with order 15, as the most significant frequencies for the machine learning model are under that. Using order 15th order ensures firm damping at the cutoff frequency.

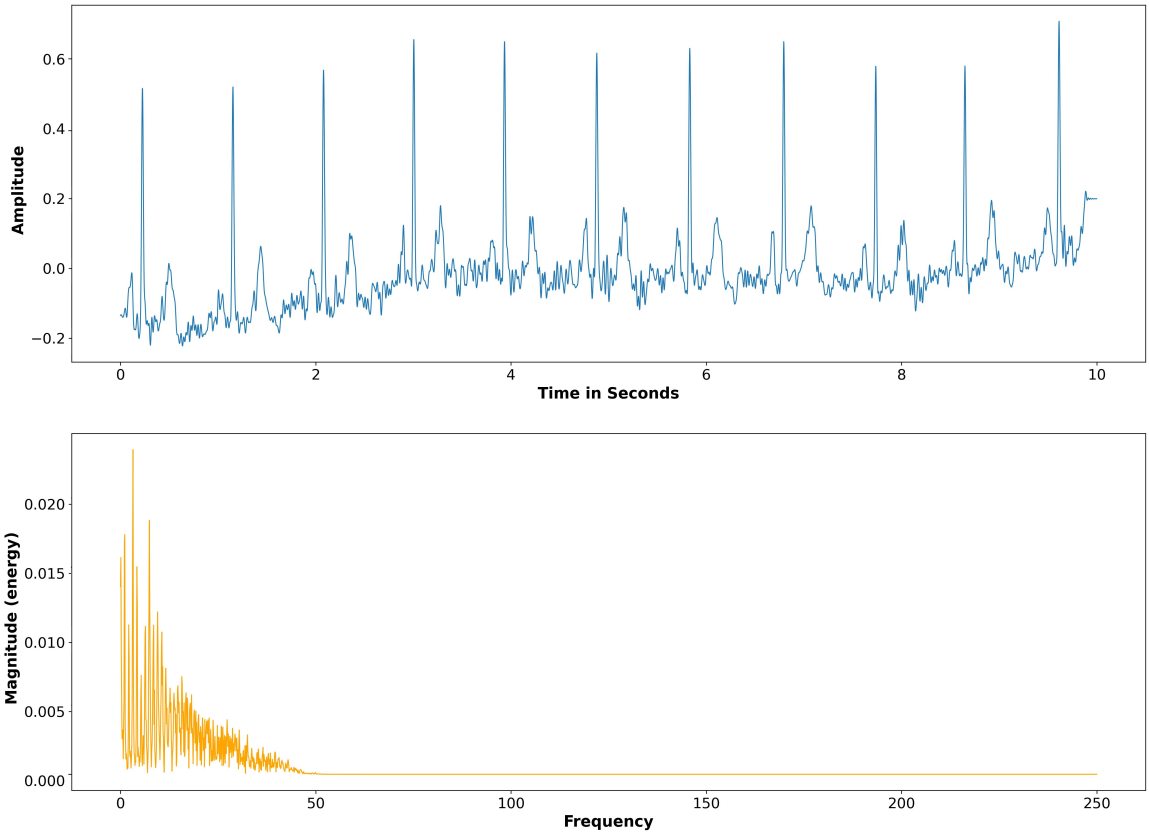

Figure 4.2: Filtered Signal and Spectrogram for ECG ID 1, Lead I, PTB-XL

Baseline Wander Correction

Wavelets are mathematical functions specifically tailored for a specific application. Wavelet analysis helps in the simultaneous representation of a signal in frequency and time domains. It makes wavelets a powerful tool to describe the local behaviour of a signal. In recent years, wavelets have had profound applications in signal analysis, denoising, and compression [71][72]. Although conventional methods like sliding window Fourier

transform provide insight into local signal changes, wavelets are more suitable for shorter signals. Fourier transforms are computationally more intensive. Therefore, wavelets are emerging as a preferred tool for complex signal analysis.

There are numerous types of wavelets which could be tailored to suit the application. A wavelet has two properties: dilation and location. The parameter dilation determines the range of frequencies that could be captured, whereas the parameter location defines the position of the wavelet with respect to time.

Some popularly used wavelets are:

1. Haar
2. Daubechies
3. Symmlet
4. Morlet

A wavelet transform is a mathematical tool that can extract local spectral and temporal information from a signal simultaneously. It decomposes a signal into its components.

There are two types of wavelet transforms:

1. Continuous Wavelet Transform - is ideal for analyzing non-stationary signals. Every possible wavelet is traversed over a range of dilation and location. This helps capture a large range of frequency, whether it is rapidly changing, transient or have slow variations.
2. Discrete Wavelet Transform - uses a finite set of wavelets with specific dilation and locations. This finite subset helps in reducing the computational resource by analyzing within a specific domain. They are extensively used in image compression and signal denoising.

A significant amount of baseline wander is present in many signals. Although some baseline wanders could be corrected using a high pass filter, this method causes a substantial reduction in the frequency domain, which negatively impacts the machine learning model's performance. Therefore, we have used multi-resolution decomposition based

method as demonstrated by A. Sargolzaei [73]. We determine the baseline for reconstructing the correcting signals. This method corrects the baseline while preserving important frequency information. The equation below represents a discrete wavelet transform where n and m control translation and dilation.

$$\varphi_{m,n}(t) = \frac{1}{\sqrt{a^m}}\varphi\left(\frac{t - nb_0a_0^m}{a_0^m}\right) \tag{4.1}$$

The equation 4.2 represents the DWT of a continuous signal, which returns the detail coefficients. Discrete dyadic wavelets with a_0 and b_0 as 2 and 1 respectively ensure that wavelet coefficient $T_{m,n}$ isn't repeated and allows complete signal regeneration.

$$T_{m,n} = \int_{-\infty}^{+\infty} x(t)\frac{1}{a_0^{m/2}}\varphi\left(a_0^{-m}t - nb_0\right)dt \tag{4.2}$$

$$S_{m,n} = \int_{-\infty}^{+\infty} x(t)\varphi_{m,n}(t)dt \tag{4.3}$$

Equations 4.4 and 4.5 are responsible for high and low pass filtering, respectively, providing multi-resolution decomposition.

$$S_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k c_k S_{m,2n+k} = \frac{1}{\sqrt{2}}\sum_k c_{k-2n}S_{m,k} \tag{4.4}$$

$$T_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k b_k S_{m,2n+k} = \frac{1}{\sqrt{2}}\sum_k b_{k-2n}S_{m,k} \tag{4.5}$$

$$\int |f(t)|^2 dt = \sum_{l=-\infty}^{\infty} |c_l|^2 + \sum_{j=0}^{\infty}\sum_{k=-\infty}^{\infty} |d_{jk}|^2 \tag{4.6}$$

The baseline is determined based on the equations mentioned above, which is subtracted from the original signal to obtain the wander-free time series.

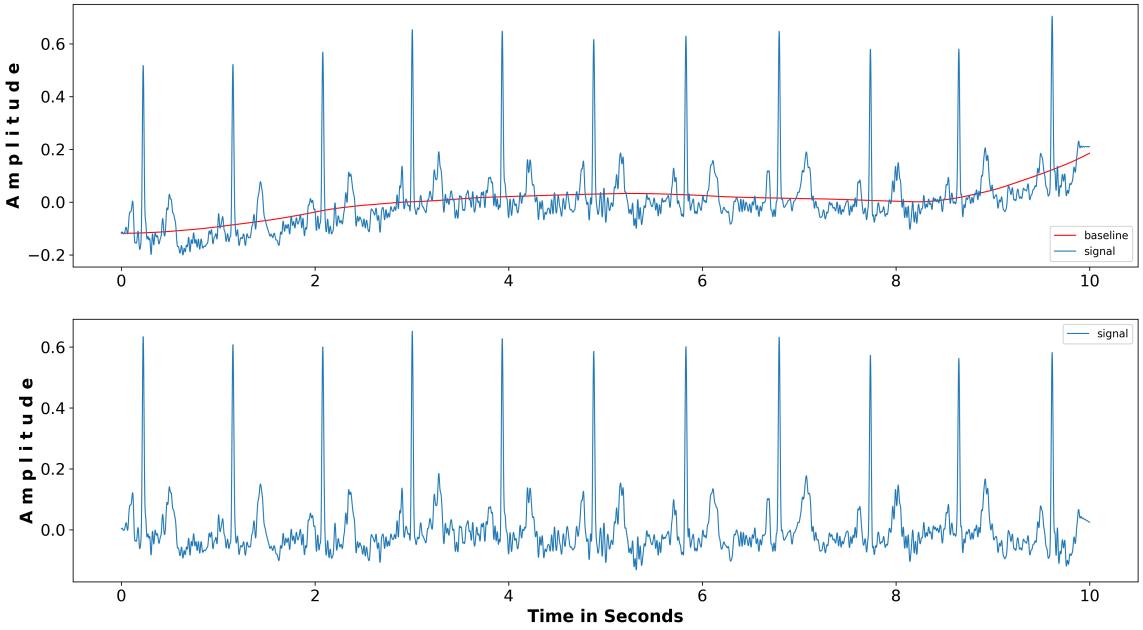

Figure 4.3: Baseline Correction for ECG ID 1, Lead I, PTB-XL

Rolling Mean

Rough peaks were present in some signals while absent in others belonging to the same classification. It would impact the model's learning performance. As a method of standardizing signals to some extent, the rolling mean was determined with a window size of 100 samples.

4.3.3 Determining Training Labels

The dataset lists SCP labels used in the scp_statements.csv file, which describes diagnostic class, form, and rhythm. The dataset mentions multiple SCP labels and their percentages of likelihood corresponding to each record. Although most of them are sorted and easy to interpret as they belong to a single superclass with high confidence, some are unsorted or have low confidence. ECG ID 39 and 63 are such example with labels ['IMI': 15.0, 'LNGQT': 100.0, 'NST_': 100.0, 'DIG': 100.0, 'ABQRS': 0.0, 'SR': 0.0] and ['ASMI': 15.0, 'ABQRS': 0.0, 'SR': 0.0] respectively. This work selected labels with the highest confidence for the training process. Fifty unique labels were identified and merged with their respective five superclasses. The superclasses as represented as Normal—NORM, ST/T

change—STTC, Myocardial Infarction—MI, Hypertrophy—HYP, and Conduction Disturbance—CD. A new label 'OTHER' was created for signals not falling within the five superclasses. Following categorization was used to mere the labels:

NORM — ['NORM']

STTC — ['NDT', 'NST_', 'DIG', 'LNGQT', 'ISC_', 'ISCAL', 'ISCIN', 'ISCIL', 'ISCAS', 'ISCLA', 'ANEUR', 'EL', 'ISCAN']

MI — ['IMI', 'ASMI', 'ILMI', 'AMI', 'ALMI', 'INJAS', 'LMI', 'INJAL', 'IPLMI', 'IPMI', 'INJIN', 'INJLA', 'PMI', 'INJIL']

HYP — ['LVH', 'LAO/LAE', 'RVH', 'RAO/RAE', 'SEHYP']

CD — ['LAFB', 'IRBBB', '1AVB', 'IVCD', 'CRBBB', 'CLBBB', 'LPFB', 'WPW', 'ILBBB', '3AVB', '2AVB']

OTHER — ['AFLT', 'AFIB', 'PSVT', 'STACH', 'PVC', 'PACE', 'PAC']

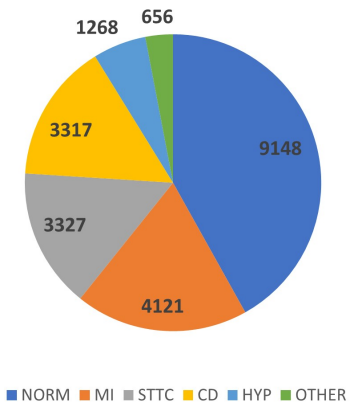

Figure 4.4: Distribution of Occurrences after Sorting and Merging Labels

In binary classification, we will classify signals between normal and abnormal. As of now, we have six labels, namely - NORM, MI, STTC, CD, HYP, and OTHER. To create training data for binary classification, we will merge the abnormal labels into a single label - "ABNORMAL". NORM forms are "NORMAL" class. Although 'OTHER' is an imbalanced minority class, it is still used to estimate real-life implementation scenarios better. Since the data is imbalanced, class weights will be used in the training process.

ABNORMAL — ['MI', 'STTC', 'CD', 'HYP', 'OTHER']

NORMAL — ['NORM']

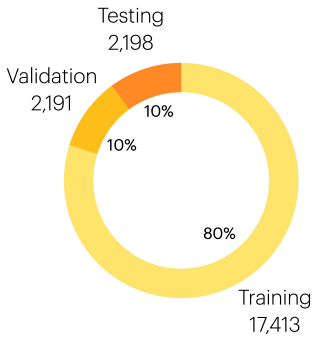

Figure 4.5: Training, Validation and Testing Distribution

4.3.4 TensorFlow/Keras Model for 2-Class Classification

We have used multiple convolutional layers for extracting the features from the training data. We modeled with 1D CNN as they are more resource-efficient for our Lite Model. The model was designed with the view that it is lightweight and convertible to TensorFlow Lite Flatbuffer. Although some quantizations for this model are not well supported yet we chose to balance between performance and resource constraints.

The input data is passed through successive convolutional layers. A Batch Normalization layer and a max-pooling layer with pool_size 2 follow each convolution layer. The variations in filter and kernel sizes help capture significant features from the input signal. The result of the convolutional layer is converted to a 1-D vector by passing them through a Flatten layer. Finally, the 1-D vector is processed by the two fully-connected layers. The final fully connected layer has one unit. The output of this dense layer provides the probability of a signal belonging to the class 'Normal' or 'Abnormal.' Multiple batch-normalization layers were used to prevent initial random weight bias. Although we have tested numerous activation functions, 'LeakyReLu' yielded better results. Loss function 'binary_crossentropy' and 'adam' optimizer were used. DataGenerator with batch size 32 was used for training on GPU. Inputs were shuffled with each batch to prevent the overfitting of data.

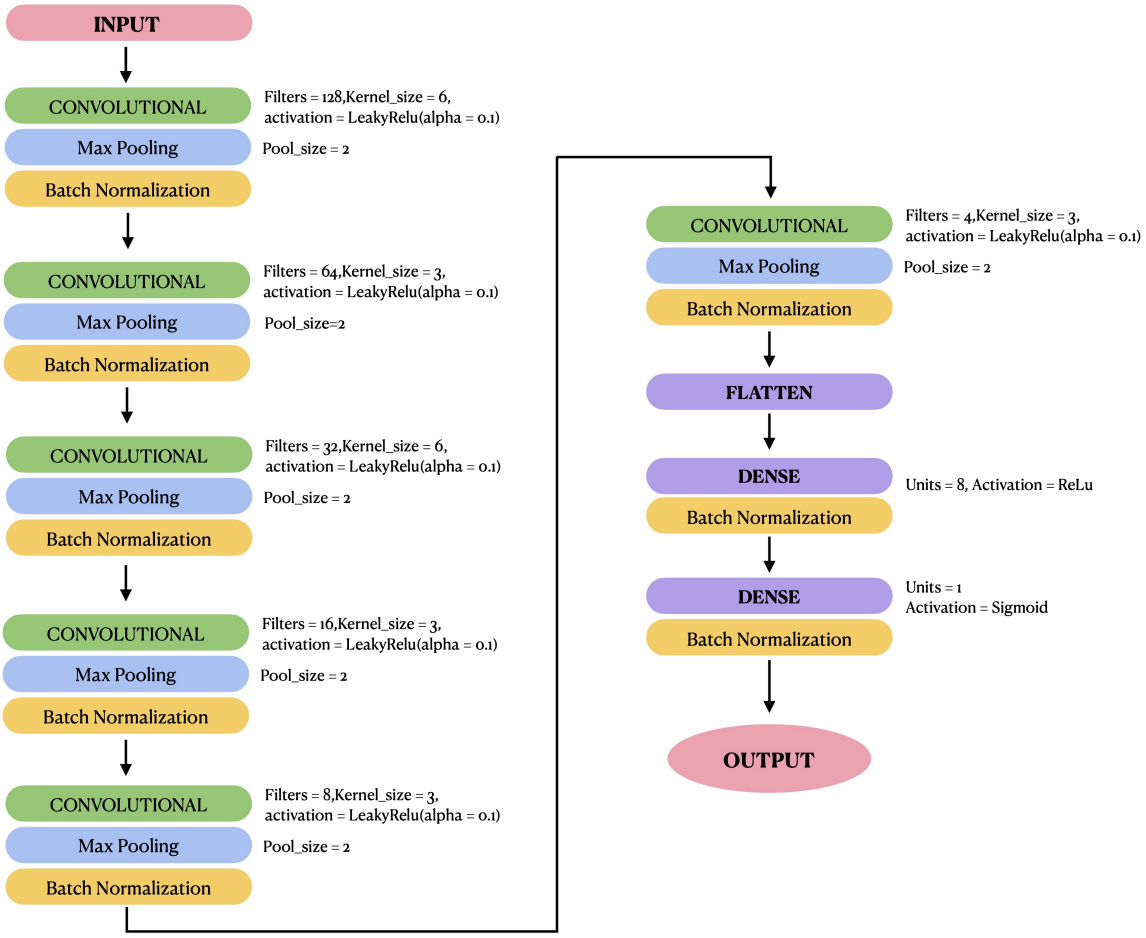

Figure 4.6: Binary Classification Model Architecture

4.3.5 TensorFlow/Keras Model for 5-Class Classification

We have used multiple convolutional layers for extracting the features from the training data. We modeled with 1D-CNN as they are more resource-efficient for our Lite Model. The model was designed with the view that it is lightweight and convertible to TensorFlow Lite Flatbuffer. Although some quantizations for this model are not well supported yet we chose to balance between performance and resource constraints. The input data is passed through successive convolutional layers with varying kernels and filter sizes. The variations in filter and kernel sizes help capture significant features from the input signal. A Batch Normalization layer and a max-pooling layer with pool_size 2 follow each convolution layer. The convolutional layers learn local spatial coherence, i.e., the local information present in one dimensional patches. This information is then used to identify

similar patterns occurrences at other positions and signals. The convolutional layers serve as a pre-processor for our LSTM layer. The convolutional layer helps in downsampling the data that LSTM processes for better application in resource-constrained platforms.

Long Short-Term Memory (LSTM) layers follow the convolutional layers. LSTM is a Recurrent Neural Network (RNN) that can learn long-term dependencies. LSTMs can learn and store information from arbitrary duration, giving our model an advantage for learning distinct features in time series. Unlike conventional RNN, LSTMs are not prone to vanishing gradient problems. The result of the LSTM layer is converted to a 1-D vector by passing them through a Flatten layer. Finally, the 1-D vector is processed by fully-connected layers. The final fully connected layer has one unit. The output of this dense layer provides the probability of a signal belonging to the class "Normal" or "Abnormal." Multiple batch-normalization layers were used to prevent initial random weight bias. Although we have tested numerous activation functions, 'LeakyReLu' yielded better results. Loss function 'catagorical_crossentropy' and 'adam' optimizer were used. Each model is run for ten epochs. DataGenerator with batch size 32 was used for training on GPU. Inputs were shuffled with each batch to prevent the overfitting of data.

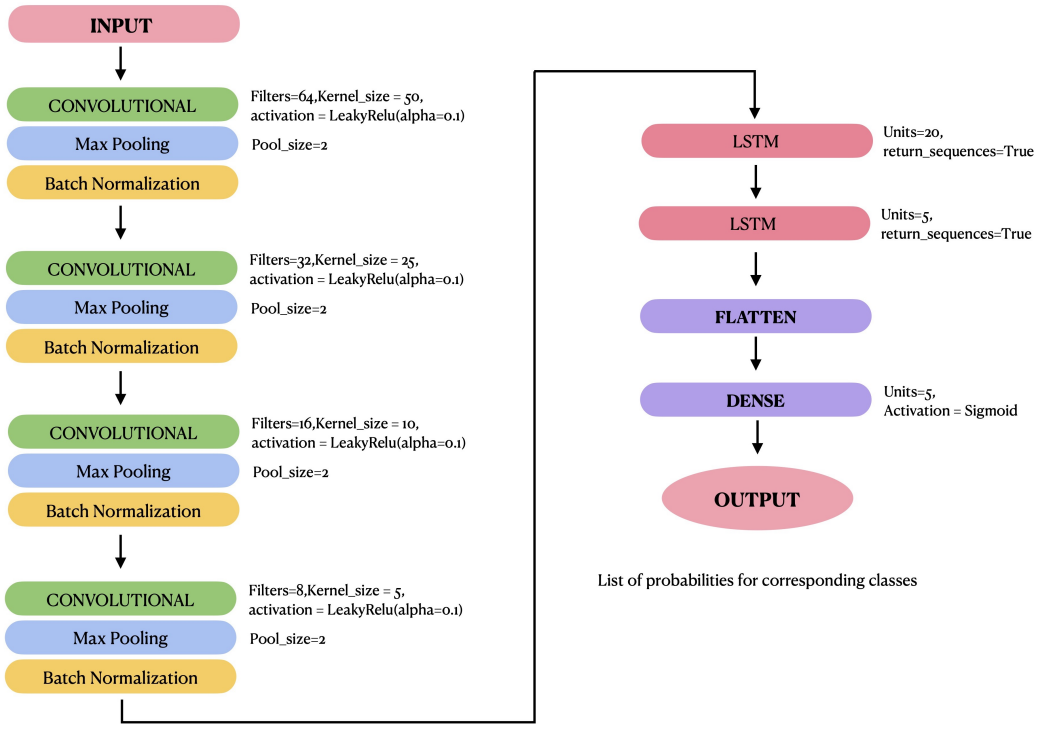

Figure 4.7: 5-Class Classification Model Architecture

4.3.6 Determining Training Labels for Classification in 5 Classes

As suggested by the database authors, the data has been split into training, validation, and test sets [15]. Fold 1-8 forms the training data, fold nine forms the validation set, and fold ten to make the testing data. Since folds 9 and 10 were verified by a human, it would help determine the model's actual performance. Other methods that randomly split the data were not used. The abnormal class was formed by merging MI, STTC, CD and HYP. 'NORM' created our 'normal' class. Labels not belonging to any diagnostic class were removed.

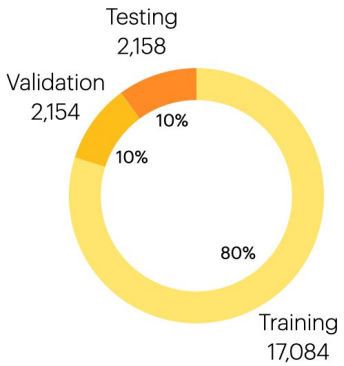

Figure 4.8: Train, Test and Validation split - 5 Class

4.3.7 Installing Raspbian OS

ReTerminal by Seeed comes with preinstalled Raspbian OS but, it's always a great idea to give things a fresh start if the installation was faulty or device was being used with another project. Installation process for eMMC based Raspberry Pi is a bit different that the ones which use SD card. To write the data on the eMMC chip, the jumper pin on the compute module need to be switched from boot mode to write mode. At this point, Compute module could be connected to any system and it should show up the eMMC storage. The eMMC storage needs to be written with Raspbian OS files. Finally, Raspbian OS could be used after changing the jumper back to boot mode. A detailed guide is available at [74].

4.3.8 Migrating TensorFlow model to Raspberry Pi

As we have already discussed in the previous sections, we'll be using TensorFlow Lite Flatbuffer to execute our trained model on the Raspberry Pi. Here we have coded

for the conversion of our trained model to TensorFlow Lite. We have converted multiple TensorFlow Lite versions with `float32`, `float16` and `int8` to compare their relative accuracy.

4.3.9 Setting up Data Storage and Rendering

To store the data and to provide easy integration with other devices we are using MYSQL server on raspberry pi. We are using MariaDB with 'phpmyadmin'.

To make things functional we have two broad objective:

1. Storing results from TensorFlow Lite on MYSQL server
2. Retrieving stored results from MYSQL and displaying on a dynamic web page

Steps involved in setting up MYSQL Server:

1. Installing Apache server
   ```
   sudo apt install apache2 -y
   ```
2. Installing PHP
   ```
   sudo apt install php -y
   ```
3. Installing MariaDB MYSQL
   ```
   sudo apt install mariadb-server php-mysql
   sudo mysql_secure_installation
   ```
4. PHP my admin
   ```
   sudo apt install phpmyadmin -y
   ```
5. Coding for the Web Pages
   We have created a very simple web page that updates when it sees new data using AJAX, HTML and PHP.

Chapter 5

Evaluation and Discussion

5.1 Evaluation Method

We have trained the model using different leads to determine the best model for Raspberry Pi implementation. In our first model, we have used only lead I. We have used leads I, II, and III in our second model. In our third model, we have used leads I, II, III, aVL, aVR, and aVF. We have also used Lead I, II and III in combination with chest lead V1 and V6. Since V1 is easiest to locate, it could easily be used by anyone for home applications. Finally, we have used all the leads for our model. This series of experiments will help us evaluate the usefullness of each combination.

5.1.1 Evaluation Metrics

To evaluate the model's performance we will use four widely used metrics - Accuracy, Precision, Recall and F1 Score. These metrics help us to determine the overall performance of our models in various domains.

1. **Accuracy** - It is the ratio of correctly predicted results to the total number of results. Although accuracy proves a general idea about model's performance when data is symmetric, but fails with asymmetric data.

$$\text{Accuracy } = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

2. **Precision** - Precision helps in overcoming the disadvantage of accuracy with asym-

metric dataset.High precision coorelates to low false positive rate. Precision is preferred when focus is on minimizing false prositives.

$$\text{Precision } = \frac{TP}{TP + FP} \tag{5.2}$$

3. **Recall** - Recall indicates information about the missed positive predictions. Precision is preferred when focus is on minimizing false negatives.

$$\text{Recall } = \frac{TP}{TP + FN} \tag{5.3}$$

4. **F1 Score** - F1 leverages both Precision and Recall. It is the weighted average of Precision and Recall. F1 Score provides a better estimate than accuracy in imbalanced dataset. It provides a better measure for multi-class dataset as the distribution is taken into account.

$$F1 \text{ score } = \frac{2 \times \text{ Precision } \times \text{ Recall}}{\text{Precision } + \text{ Recall}} \tag{5.4}$$

- **True Positive (TP)** - Correctly predicted positive values
- **True Negatives (TN)** - Correctly predicted negative values
- **False Positive (FP)** - incorrectly predicted positives
- **False Negatives (FN)** - incorrectly predicted negatives

5.1.2 TensorFlow Lite Evaluation

Each model was saved and converted to TensorFlow Lite Flat buffer. The TensorFlow Lite model was run on Raspberry Pi 4.0 to evaluate its performance on the test dataset. Two TensorFlow Lite FlatBuffers were generated using each model, each quantized as `float32` and `float16`. The models were deployed and tested on Raspberry Pi CM 4. The raw data was read from the stored memory and processed results were stored in MYSQL database. Dynamic pages were created to display the results in real-time. The data could be accessed and displayed on any system on the same network using a web browser, providing a unified storage and monitoring framework. The size and performance of each model was determined and recorded.

5.2 Results from 2-Class Classification Model

5.2.1 Confusion Matrix

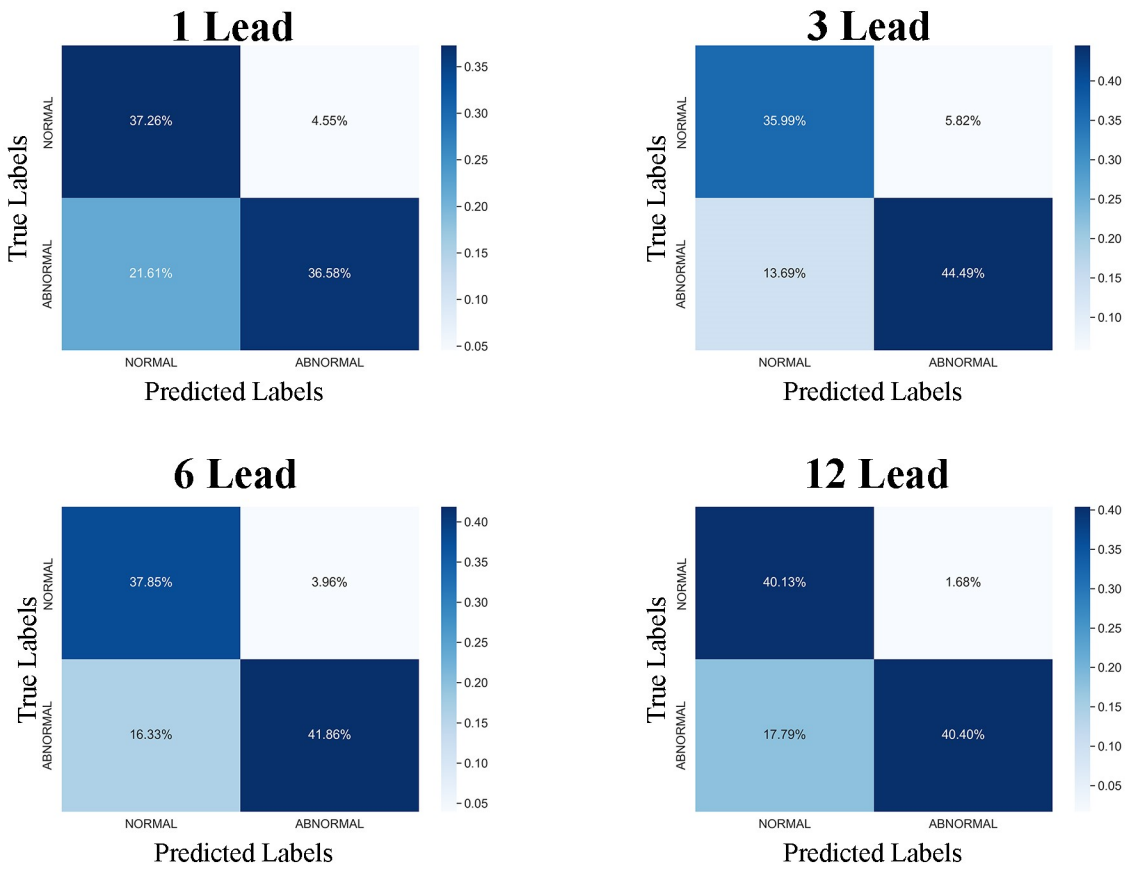

Figure 5.1: 2-Class Classification Confusion Matrix

5.2.2 Model Evaluation

The F1 score suggests that using single lead configuration gives the worst performance. As the number of leads increases beyond three, no significant gain in F1-score is observed. When one chest lead is used along with six limb leads, the performance is slightly better and comparable to twelve lead configuration.

Channels Used	**Accuracy** (%)	**Precision** (%)	**Recall** (%)	**F1 Score** (%)
1	73.55 ± 0.28	85.63 ± 0.27	62.55 ± 0.29	72.29 ± 0.22
3	80.24 ± 0.22	88.17 ± 0.21	75.90 ± 0.30	81.57 ± 0.21
6	79.50 ± 0.19	90.64 ± 0.30	71.30 ± 0.22	79.82 ± 0.21
12	81.03 ± 0.16	93.50 ± 0.23	71.77 ± 0.15	81.06 ± 0.17
6+V1	81.20 ± 0.15	89.11 ± 0.16	76.50 ± 0.19	82.32 ± 0.14

Table 5.1: 2-Class TensorFlow Keras Model Evaluation

5.2.3 TensorFlow Lite Evaluation

For all the models, we observe significant size reduction when converted to TensorFlow lite. The accuracy of the TensorFlow lite models is comparable to the accuracy of the original model. These model sizes are suitable for implementing a cheap 512KB microcontroller.

Channels Used	**TF Lite Float32**		**TF Lite Float16**		**Original Model**
	Size(KB)	Accuracy(%)	Size(KB)	Accuracy(%)	Size(MB)
1	196 ± 0.90	73.54 ± 0.28	111 ± 0.50	73.56 ± 0.28	1.37 ± 0.01
3	202 ± 1.10	80.20 ± 0.21	115 ± 0.50	80.28 ± 0.22	1.41 ± 0.01
6	213 ± 1.05	79.52 ± 0.20	119 ± 0.70	79.51 ± 0.18	1.43 ± 0.01
12	229 ± 2.30	81.05 ± 0.18	127 ± 1.10	81.03 ± 0.20	1.49 ± 0.02
6+V1	217 ± 1.20	80.10 ± 0.11	121 ± 0.60	79.98 ± 0.16	1.44 ± 0.01

Table 5.2: 2-Class TensorFlow Lite Model Evaluation

5.3 Results from 5-Class Classification Model

5.3.1 Confusion Matrix

A total of six models with different lead configurations were trained and evaluated. The confusion matrix for all the models has been recorded in table 5.3. Figure 5.2 shows the confusion matrix corresponding to five different classes for the model using all 12 leads.

Channels Used	Classes	True Positive	True Negative	False Positive	False Negative
1	CD	0.40 ± 0.02	0.95 ± 0.01	0.60 ± 0.02	0.05 ± 0.01
	HYP	0.25 ± 0.02	0.96 ± 0.01	0.75 ± 0.02	0.04 ± 0.01
	MI	0.31 ± 0.03	0.92 ± 0.02	0.69 ± 0.03	0.08 ± 0.02
	NORM	0.78 ± 0.04	0.76 ± 0.03	0.22 ± 0.04	0.24 ± 0.03
	STTC	0.46 ± 0.01	0.93 ± 0.02	0.54 ± 0.01	0.07 ± 0.02
3	CD	0.55 ± 0.02	0.93 ± 0.02	0.45 ± 0.02	0.07 ± 0.02
	HYP	0.35 ± 0.02	0.95 ± 0.01	0.65 ± 0.02	0.05 ± 0.01
	MI	0.49 ± 0.02	0.90 ± 0.01	0.51 ± 0.02	0.10 ± 0.01
	NORM	0.81 ± 0.03	0.79 ± 0.02	0.19 ± 0.03	0.21 ± 0.02
	STTC	0.64 ± 0.02	0.89 ± 0.01	0.36 ± 0.02	0.11 ± 0.01
6	CD	0.54 ± 0.02	0.94 ± 0.02	0.46 ± 0.02	0.06 ± 0.02
	HYP	0.40 ± 0.03	0.95 ± 0.02	0.60 ± 0.03	0.05 ± 0.02
	MI	0.55 ± 0.01	0.91 ± 0.02	0.45 ± 0.01	0.09 ± 0.02
	NORM	0.87 ± 0.03	0.79 ± 0.02	0.13 ± 0.03	0.21 ± 0.02
	STTC	0.54 ± 0.02	0.95 ± 0.01	0.46 ± 0.02	0.05 ± 0.01
12	CD	0.71 ± 0.02	0.89 ± 0.02	0.29 ± 0.02	0.11 ± 0.02
	HYP	0.38 ± 0.02	0.97 ± 0.01	0.62 ± 0.02	0.03 ± 0.01
	MI	0.65 ± 0.03	0.89 ± 0.02	0.35 ± 0.03	0.11 ± 0.02
	NORM	0.86 ± 0.01	0.83 ± 0.02	0.14 ± 0.01	0.17 ± 0.02
	STTC	0.65 ± 0.03	0.94 ± 0.01	0.35 ± 0.03	0.06 ± 0.01
6+V1	CD	0.64 ± 0.02	0.95 ± 0.01	0.36 ± 0.02	0.05 ± 0.01
	HYP	0.33 ± 0.01	0.96 ± 0.01	0.67 ± 0.01	0.04 ± 0.01
	MI	0.60 ± 0.01	0.88 ± 0.02	0.40 ± 0.01	0.12 ± 0.02
	NORM	0.93 ± 0.02	0.74 ± 0.03	0.07 ± 0.02	0.26 ± 0.03
	STTC	0.58 ± 0.02	0.94 ± 0.01	0.42 ± 0.02	0.06 ± 0.01

Table 5.3: Confusion Matrix for all Multi-Label Classification CNN-LSTM Models

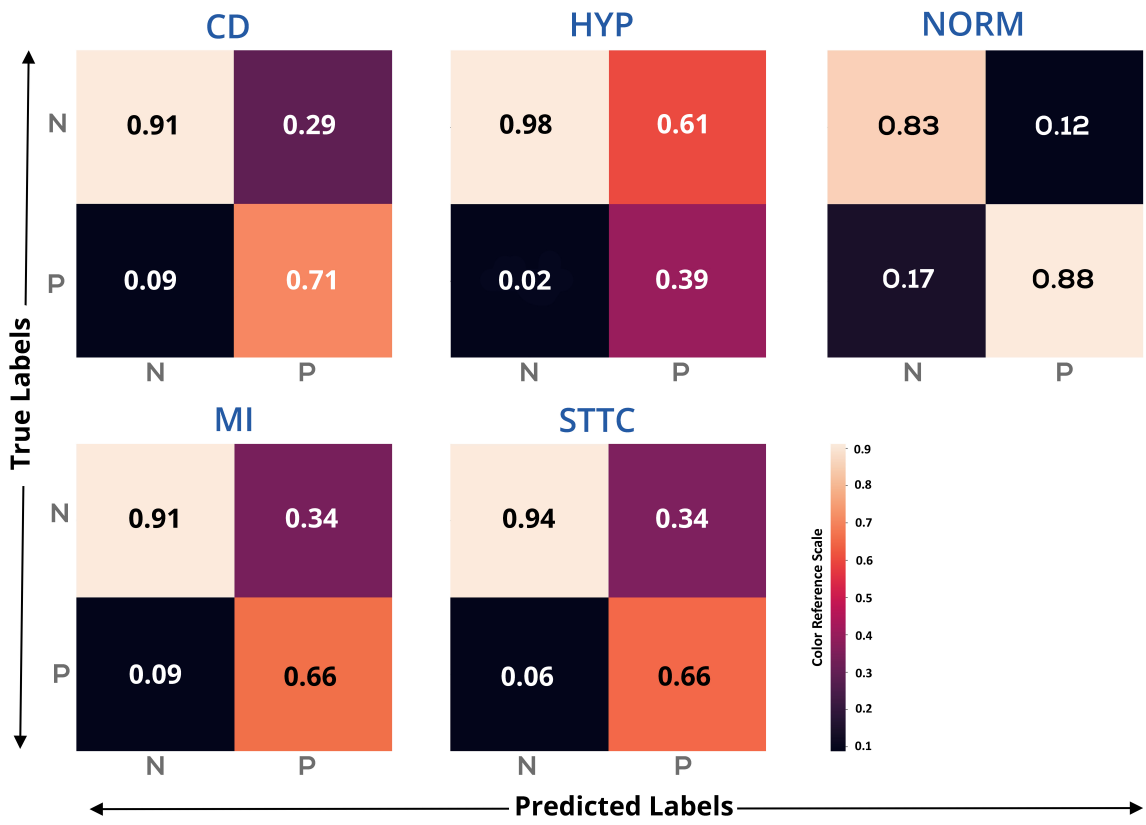

Figure 5.2: 5-Class Classification Confusion Matrix

5.3.2 Model Evaluation

Channels Used	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
1	55.80 ± 0.17	68.83 ± 0.15	51.01 ± 0.19	58.59 ± 0.14
3	61.69 ± 0.11	73.42 ± 0.09	64.16 ± 0.12	68.59 ± 0.10
6	63.48 ± 0.14	73.46 ± 0.09	65.44 ± 0.11	69.22 ± 0.08
12	70.42 ± 0.13	77.48 ± 0.12	73.23 ± 0.13	75.29 ± 0.10
6+V1	66.25 ± 0.10	75.19 ± 0.14	69.51 ± 0.08	72.24 ± 0.09

Table 5.4: 5-Class TensorFlow/Keras Model Evaluation

The F1 score suggests that using a single lead configuration gives the worst performance. The F1 score improves when the number of leads increases. No significant gain is observed when more than three limb leads are used. The best performance was observed when all twelve leads were used. When one chest lead was used along with six limb leads,

the performance was significantly better, making it a better choice for implementation in non-clinical in-home applications.

5.3.3 TensorFlow Lite Evaluation

Although a significant reduction in model size was achieved when converted to TensorFlow lite model, the model's accuracy was significantly reduced. Twelve lead configuration still performs the best, followed by limb leads with one chest lead. The reason for this loss of accuracy will be explored in our future work.

Channels Used	**TF Lite Float32**		**TF Lite Float 16**		**Original Model**
	Size(KB)	Accuracy(%)	Size(KB)	Accuracy(%)	Size(MB)
1	611 ± 1.95	43.70 ± 0.15	337 ± 0.94	43.70 ± 0.14	6.85 ± 0.01
3	637 ± 2.15	53.21 ± 0.13	347 ± 0.07	53.31 ± 0.71	6.89 ± 0.01
6	673 ± 2.20	54.42 ± 0.09	365 ± 1.10	54.33 ± 0.08	7.01 ± 0.03
12	748 ± 3.59	58.31 ± 0.14	405 ± 1.15	58.31 ± 0.10	7.25 ± 0.04
6+V1	682 ± 2.34	56.21 ± 0.08	371 ± 0.06	56.17 ± 0.12	7.10 ± 0.03

Table 5.5: 5-Class TensorFlow Lite Model Evaluation

5.3.4 Edge Node/Raspberry Pi Web Interface

The models were deployed and tested on Raspberry Pi CM 4. The raw data was read from the stored memory, and processed results were stored in the MYSQL database. Dynamic pages were created to display the results in real time. The data could be accessed by any system on the same network using Raspberry Pi's local IP address in a web browser. A static domain name or a Dynamic DNS service could be used to access data over the web. Such a method would require port forwarding enabled on the external gateway router, providing a means for the external traffic to reach our server.

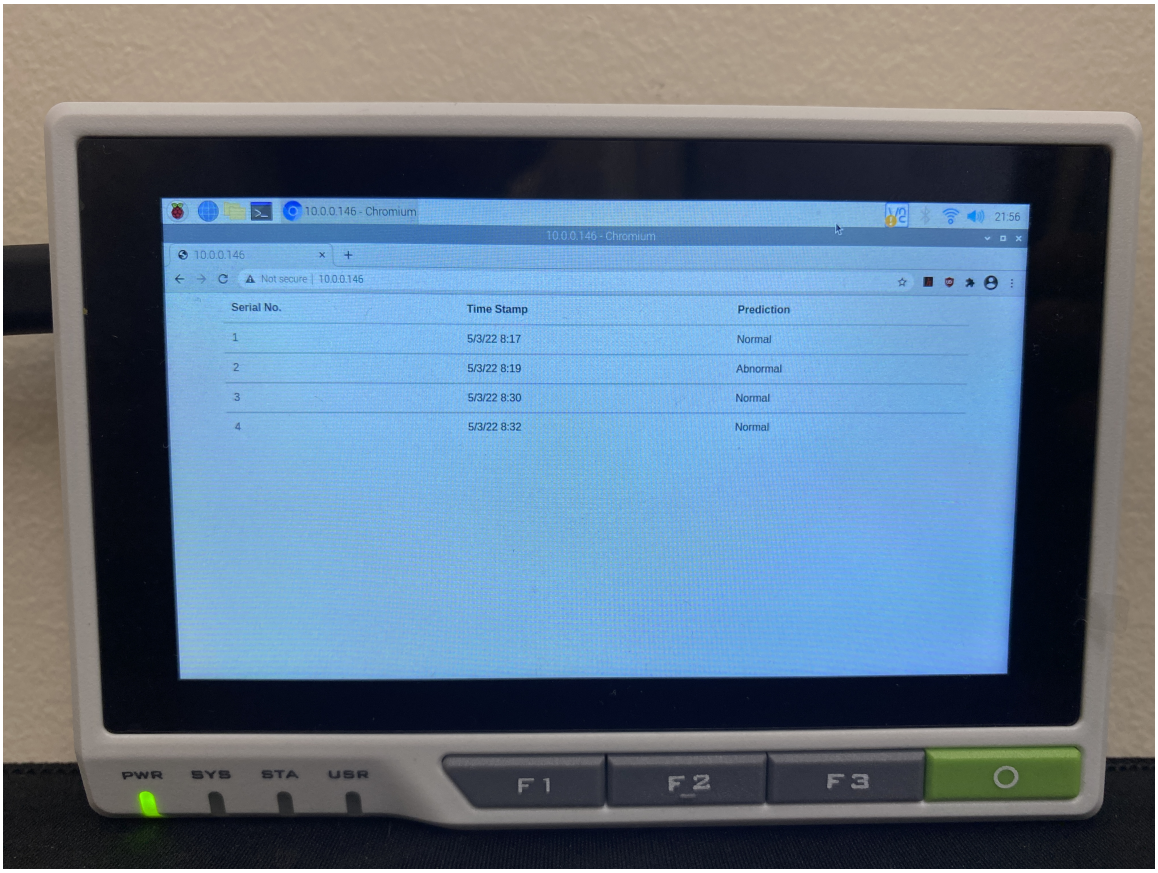

Figure 5.3: Edge Node/Raspberry Pi Web Interface

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Both 2-class and 5-class classifications present unique and exciting results. For 2-class classification, this work demonstrated that using all twelve leads for classification yields the best accuracy. We can also observe that using at least three 'leads' over just one increases the classification accuracy by about 7%. However, precision increases when more than three 'leads' are used, with a minor increase in accuracy. We also demonstrated the use of TensorFlow Lite for implementing machine learning models while maintaining accuracy. The accuracy of all the TensorFlow lite models was about the same as that of the original models. The 'float32' model presented about 86% decrease in model size, whereas the 'float16' model presented about 94% decrease.

For 5-class classification, we aimed to develop an efficient deep learning model for multi-label classification, which could be deployed on a resource constrained platform with TensorFlow Lite Flatbuffer. We demonstrated that using all twelve leads for classification yields the best accuracy. We can also observe that using at least three 'leads' over just one increases the classification accuracy by around 6%. Using all twelve leads provides maximum accuracy and precision. We evaluated TensorFlow Lite's implementation of the models. The accuracy of TensorFlow lite models drops by about 12% compared to the accuracy of original models. The 'float32' model presented about 90% decrease in model size, whereas the 'float16' model presented about 94.5% decrease.

6.2 Future Work

One of the goals of this work was to develop an efficient deep learning model that could easily be deployed on resource-constrained IoT devices for ECG signal classification into 'Normal' and 'Abnormal' classes. Since PTB-XL is one of the largest open public databases available, our second goal was to evaluate its performance with our models.

Although the PTB-XL dataset is quite large, we still need to explore ways to tackle the small number of samples for numerous minority classes. One way could be merging samples from other datasets to expand this scope.

To improve the model's performance, more complex and elaborate models need to be explored and tested with TensorFlow Lite conversion. Since TensorFlow lite is still expanding its support for conventionally used features in modeling, it is vital to explore more convertible models yielding higher accuracy. Additionally, we would focus on int8 and int16 quantizations for microcontroller implementations. This would help make way for more resource-efficient ECG-IoT devices, which could be deployed in clinical and non-clinical settings.

Once we have high-performing models, we will try to scale sensor-specific real-time ECG data for the models. We intend to test the model on larger samples and run small-scale trials to estimate the real-time performance in anomaly detection alongside conventional clinical equipment.

The effect of other body vitals and meta-data such as sex, age, height, or body oxygen saturation level needs to be studied to help get better ECG estimates.

Bibliography

[1] W. Lardier, Q. Varo, and J. Yan, "Dynamic Reduced-Round Cryptography for Energy-Efficient Wireless Communication of Smart IoT Devices," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020. DOI: 10.1109/icc40277.2020.9149305.

[2] Y. B. Zikria, R. Ali, M. K. Afzal, and S. W. Kim, "Next-Generation Internet of Things (IoT): Opportunities, Challenges, and Solutions," *Sensors*, vol. 21, no. 4, p. 1174, 2021. DOI: 10.3390/s21041174.

[3] F. B. Insights, *Internet of things (iot) in healthcare market worth usd 446.52 billion at 25.9% cagr by 2028 owing to presence of large population in asia-pacific*, 2021. [Online]. Available: https://www.globenewswire.com/news-release/2021/10/12/2312160/0/en/Internet-of-things-IoT-in-Healthcare-Market-Worth-USD-446-52-Billion-at-25-9-CAGR-by-2028-Owing-to-Presence-of-Large-Population-in-Asia-Pacific.html.

[4] X. Liu and X. Zhang, "Rate and Energy Efficiency Improvements for 5G-Based IoT with Simultaneous Transfer," *IEEE Internet of Things Journal*, vol. 6, no. 4, 5971–5980, 2019. DOI: 10.1109/jiot.2018.2863267.

[5] A. P. Plageras, K. E. Psannis, C. Stergiou, H. Wang, and B. B. Gupta, "Efficient IoT-based sensor BIG Data collection–processing and analysis in smart buildings," *Future Generation Computer Systems*, vol. 82, pp. 349–357, 2018.

[6] H. Elahi, K. Munir, M. Eugeni, S. Atek, and P. Gaudenzi, "Energy Harvesting towards Self-Powered IoT Devices," *Energies*, vol. 13, no. 21, p. 5528, 2020.

[7] J Li, M Bhuiyan, X Huang, B McDonald, T. Farrell, and E. Clancy, "Reducing Electric Power Consumption when Transmitting ECG/EMG/EEG using a Bluetooth Low Energy Microcontroller," in *2018 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, IEEE, 2018, pp. 1–3.

[8] M. Jia, Z. Yin, D. Li, Q. Guo, and X. Gu, "Toward Improved Offloading Efficiency of Data Transmission in the IoT-cloud by Leveraging Secure Truncating OFDM," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4252–4261, 2018.

[9] O.-K. Shahryari, H. Pedram, V. Khajehvand, and M. D. TakhtFooladi, "Energy-Efficient and delay-guaranteed computation offloading for fog-based IoT networks," *Computer Networks*, vol. 182, p. 107 511, 2020.

[10] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms: Transparent Computing, Mobile Edge Computing, Fog Computing, and Cloudlet," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–36, 2019.

[11] M. Laska, S. Herle, R. Klamma, and J. Blankenbach, "A Scalable Architecture for Real-Time Stream Processing of Spatiotemporal IoT stream Data—Performance analysis on the Example of Map Matching," *ISPRS International Journal of Geo-Information*, vol. 7, no. 7, p. 238, 2018.

[12] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[13] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren, and G. Fortino, "Autonomic computation offloading in mobile edge for IoT applications," *Future Generation Computer Systems*, vol. 90, pp. 149–157, 2019.

[14] P. Wagner, N. Strodthoff, R.-D. Bousseljot, W. Samek, and T. Schaeffter, *Ptb-xl, a large publicly available electrocardiography dataset*, 2022. DOI: 10.13026/KFZX-AW45. [Online]. Available: https://physionet.org/content/ptb-xl/1.0.3/.

[15] P. Wagner *et al.*, "PTB-XL, a large publicly available electrocardiography dataset," *Scientific Data*, vol. 7, no. 1, 2020. DOI: 10.1038/s41597-020-0495-6.

[16] J. Zhang, A. Liu, D. Liang, X. Chen, and M. Gao, "Interpatient ECG Heartbeat Classification with an Adversarial Convolutional Neural Network," *Journal of Healthcare Engineering*, vol. 2021, 2021.

[17] T. D. Pham, "Time—frequency time—space LSTM for robust classification of physiological signals," *Scientific reports*, vol. 11, no. 1, pp. 1–11, 2021.

[18] S. A. Mirsalari, N. Nazari, S. A. Ansarmohammadi, S. Sinaei, M. E. Salehi, and M. Daneshtalab, "ELC-ECG: Efficient LSTM Cell for ECG Classification based on Quantized Architecture," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2021, pp. 1–5.

[19] M. Mittal, S. Tanwar, B. Agarwal, and L. M. Goyal, "Energy conservation for iot devices," *Concepts, Paradigms and Solutions, Studies in Systems, Decision and Control, In Preparation. Springer Nature Singapore Pte Ltd., Singapore*, pp. 1–356, 2019.

[20] B. Martinez, M. Monton, I. Vilajosana, and J. D. Prades, "The Power of Models: Modeling Power Consumption for IoT Devices," *IEEE Sensors Journal*, vol. 15, no. 10, pp. 5777–5789, 2015.

[21] Bluetooth, *Bluetooth technology overview*. [Online]. Available: `https://www.bluetooth.com/learn-about-bluetooth/tech-overview/`.

[22] Zigbee, *Zigbee: Complete iot solution*, 2022. [Online]. Available: `https://csa-iot.org/all-solutions/zigbee/`.

[23] P. Paudel, S. Kim, S. Park, and K.-H. Choi, "A Context-Aware IoT and Deep-Learning-Based Smart Classroom for Controlling Demand and Supply of Power Load," *Electronics*, vol. 9, no. 6, p. 1039, 2020.

[24] B. Rababah and R. Eskicioglu, "Distributed Intelligence Model for IoT Applications Based on Neural Networks," *International Journal of Computer Network & Information Security*, vol. 13, no. 3, 2021.

[25] K. Weimann and T. O. Conrad, "Transfer learning for ECG classification," *Scientific reports*, vol. 11, no. 1, pp. 1–12, 2021.

[26] *Tensorflow lite: Ml for mobile and edge devices*. [Online]. Available: `https://www.tensorflow.org/lite`.

[27] A. B. Rad *et al.*, "ECG-Based Classification of Resuscitation Cardiac Rhythms for Retrospective Data Analysis," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 10, pp. 2411–2418, 2017.

[28] F. Funk, T. Bucksch, and D. Mueller-Gritschneder, "ML Training on a Tiny Microcontroller for a Self-adaptive Neural Network-Based DC Motor Speed Controller," in *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*, Springer, 2020, pp. 268–279.

[29] R. David *et al.*, "TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[30] G. Demosthenous and V. Vassiliades, "Continual Learning on the Edge with TensorFlow Lite," *arXiv preprint arXiv:2105.01946*, 2021.

[31] *Overview*. [Online]. Available: `https://google.github.io/flatbuffers/`.

[32] H. Li and P. Boulanger, "A Survey of Heart Anomaly Detection Using Ambulatory Electrocardiogram (ECG)," *Sensors*, vol. 20, no. 5, p. 1461, 2020. DOI: `10.3390/s20051461`.

[33] E. J. d. S. Luz, W. R. Schwartz, G. Cámara-Chávez, and D. Menotti, "ECG-based heartbeat classification for arrhythmia detection: A survey," *Computer methods and programs in biomedicine*, vol. 127, pp. 144–164, 2016.

[34] G. J. Warmerdam, R. Vullings, L. Schmitt, J. O. Van Laar, and J. W. Bergmans, "Hierarchical Probabilistic Framework for Fetal R-Peak Detection, Using ECG Waveform and Heart Rate Information," *IEEE Transactions on Signal Processing*, vol. 66, no. 16, pp. 4388–4397, 2018.

[35] H. B. Seidel, M. M. A. da Rosa, G. Paim, E. A. C. da Costa, S. J. Almeida, and S. Bampi, "Approximate Pruned and Truncated Haar Discrete Wavelet Transform VLSI Hardware for Energy-Efficient ECG Signal Processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1814–1826, 2021.

[36] J.-S. Park, S.-W. Lee, and U. Park, "R Peak Detection Method Using Wavelet Transform and Modified Shannon Energy Envelope," *Journal of healthcare engineering*, vol. 2017, 2017.

[37] A. Giorgio, M. Rizzi, and C. Guaragnella, "Efficient Detection of Ventricular Late Potentials on ECG Signals Based on Wavelet Denoising and SVM Classification," *Information*, vol. 10, no. 11, p. 328, 2019.

[38] M. R. Fikri, I. Soesanti, and H. A. Nugroho, "ECG Signal Classification Review," *IJITEE (International Journal of Information Technology and Electrical Engineering)*, vol. 5, no. 1, p. 15, 2021. DOI: 10.22146/ijitee.60295.

[39] V. Leon, S. Mouselinos, K. Koliogeorgi, S. Xydis, D. Soudris, and K. Pekmestzi, "A Tensorflow Extension Framework for Optimized Generation of Hardware CNN Inference Engines," *Technologies*, vol. 8, no. 1, p. 6, 2020.
DOI: 10.3390/technologies8010006.

[40] J. Li, Y. Si, T. Xu, and S. Jiang, "Deep convolutional Neural Network Based ECG Classification System Using Information Fusion and One-Hot Encoding Techniques," *Mathematical Problems in Engineering*, vol. 2018, 1–10, 2018.
DOI: 10.1155/2018/7354081.

[41] A. Daamouche, L. Hamami, N. Alajlan, and F. Melgani, "A wavelet optimization approach for ECG Signal Classification," *Biomedical Signal Processing and Control*, vol. 7, no. 4, 342–349, 2012. DOI: 10.1016/j.bspc.2011.07.001.

[42] M. Nazarahari, S. Ghorbanpour Namin, A. H. Davaie Markazi, and A. Kabir Anaraki, "A multi-wavelet optimization approach using similarity measures for electrocardiogram signal classification," *Biomedical Signal Processing and Control*, vol. 20, 142–151, 2015. DOI: 10.1016/j.bspc.2015.04.010.

[43] D. Zhang *et al.*, "An ECG Signal De-Noising Approach Based on Wavelet Energy and Sub-Band Smoothing Filter," *Applied Sciences*, vol. 9, no. 22, p. 4968, 2019.
DOI: 10.3390/app9224968.

[44] P. Bakucz, S. Willems, and B. Hoffmann, "Fast detection of Atrial Fibrillation using wavelet transform," in *World Congress on Medical Physics and Biomedical Engineering, September 7-12, 2009, Munich, Germany*, Springer, 2009, pp. 81–84.

[45] P. S. Addison, "Wavelet transforms and the ECG: a review," *Physiological Measurement*, vol. 26, no. 5, 2005. DOI: 10.1088/0967-3334/26/5/r01.

[46] I. M. Dremin, O. V. Ivanov, and V. A. Nechitailo, "Wavelets and their uses," *Physics-Uspekhi*, vol. 44, no. 5, 447–478, 2001.
DOI: 10.1070/pu2001v044n05abeh000918.

[47] S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 3, pp. 664–675, 2015.

[48] S. Saadatnejad, M. Oveisi, and M. Hashemi, "LSTM-Based ECG Classification for Continuous Monitoring on Personal Wearable Devices," *IEEE journal of biomedical and health informatics*, vol. 24, no. 2, pp. 515–523, 2019.

[49] Chruścik and et al., "Chapter VI Cardiovascular System," in *Fundamentals of Anatomy and Physiology. Australian Edition*. University of Southern Queensland, 2021.

[50] Z. Nesheiwat, A. Goyal, and M. Jagtap, *Atrial fibrillation*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK526072/.

[51] M. Ziccardi, A. Goyal, and C. Maani, *Atrial flutter*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK540985/.

[52] A. Henning and C. Krawiec, *Sinus tachycardia*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK553128/.

[53] Y. Hafeez and S. Grossman, *Sinus bradycardia*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK493201/.

[54] C. Foth, M. Gangwani, and H. Alvey, *Ventricular tachycardia*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK532954/.

[55] D. Ludhwani, A. Goyal, and M. Jagtap, *Ventricular fibrillation*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK537120/.

[56] S. Oldroyd, B. Rodriguez, and A. Makaryus, *First degree heart block*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK448164/#::text=First“%2Ddegree“%20atrioventricular“%20(AV),discovered“%20only“%20on“%20routine“%20ECG..

[57] A. Kashou, A. Goyal, T. Nguyen, and L. Chhabra, *Atrioventricular block*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK459147/.

[58] M. Mangi, W. Jones, M. Mansour, and L. Napier, *Atrioventricular block second-degree*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK482359/.

[59] K. V, C. L, and S. M, *Third-degree atrioventricular block*, 2022. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/31424783/.

[60] D. Scherbak and G. Hicks, *Left bundle branch block*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK482167/#::text=Left“%20bundle“%20branch“%20block“%20(LBBB,His“%2DPurkinje“%20system“%20is“%20compromised..

[61] S. D and H. GJ, *Left bundle branch block*, 2022. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/29489192/.

[62] W. Harkness and M. Hicks, *Right bundle branch block*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK507872/#::text=The“%20characteristic“%20ECG“%20findings“%20for,is“%20greater“%20than“%2040“%20milliseconds.

[63] Tensorflow, *Tensorflow*. [Online]. Available: https://www.tensorflow.org/.

[64] T. Keras, *Simple. flexible. powerful.* [Online]. Available: https://keras.io/.

[65] AHA, *Aha database sample excluded record*, 2003. [Online]. Available: https://physionet.org/content/ahadb/1.0.0/.

[66] G. Moody and R. Mark, *Mit-bih arrhythmia database*, 2005. [Online]. Available: https://physionet.org/content/mitdb/1.0.0/.

[67] R.-D. Bousseljot, *Ptb diagnostic ecg database*, 2004. [Online]. Available: https://www.physionet.org/content/ptbdb/1.0.0/.

[68] E. Yakushenko, *St petersburg incart 12-lead arrhythmia database*, 2008. [Online]. Available: https://physionet.org/content/incartdb/1.0.0/.

[69] *Telemetric and holter ecg warehouse*. [Online]. Available: http://www.thew-project.org/.

[70] *Physiobc*. [Online]. Available: http://www.physiobc.org/.

[71] C. C. FR; *Comparing different wavelet transforms on removing electrocardiogram baseline wanders and special trends*. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/33380333/.

[72] Y. Xu, M. Luo, T. Li, and G. Song, “Ecg signal de-noising and baseline wander correction based on ceemdan and wavelet threshold,” *Sensors*, vol. 17, no. 12, p. 2754, 2017. DOI: 10.3390/s17122754.

[73] A. Sargolzaei, K. Faez, and S. Sargolzaei, "A new robust wavelet based algorithm for baseline wandering cancellation in ECG signals," in *2009 IEEE International Conference on Signal and Image Processing Applications*, IEEE, 2009, pp. 33–38.

[74] B. Zuo, *Getting started with reterminal.* [Online]. Available: `https://wiki.seeedstudio.com/reTerminal/`.

Publication

Sharma, K., Eskicioglu, R. (2022). Deep learning-based ECG classification on Raspberry Pi using a TensorFlow lite Model based on PTB-XL Dataset. International Journal of Artificial Intelligence Applications, 13(4), 55–66.
https://doi.org/10.5121/ijaia.2022.13404