ATRIAL FIBRILLATION DETECTION USING CONVOLUTONAL NUERAL NETWORK

A PROJECT REPORT

Submitted by

KOWSALYA S	**710020106016**
NIVAASHINI S	**710020106019**
RASHMIYA J	**710020106022**
ASWINKUMAR S	**710020106303**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

ANNA UNIVERSITY REGIONAL CAMPUS COIMBATORE
COIMBATORE - 641 046

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report titled **"ATRIAL FIBRILLATION DETECTION USING CNN"** is the bonafide work of **"KOWSALYA S (710020106016), NIVAASHINI S (710020106019), RASHMIYA J (710020106022), ASWINKUMAR S (710020106303)"** who carried out the project work under my supervision.

SIGNATURE

Dr.N.Kumaresan B.E, M.E, Ph.D.,
HEAD OF THE DEPARTMENT,
Assistant Professor,
Department of ECE,
Anna University Regional Campus,
Coimbatore – 641 046.

SIGNATURE

Dr.P.Seethalakshmi B.E, M.E, Ph.D.,
SUPERVISOR,
Associate Professor,
Department of ECE,
Anna University Regional Campus,
Coimbatore – 641 046

Submitted for the project Viva-Voce Examination held on _____.

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

First and foremost, we place this project work on the feet of **GOD ALMIGHTY** who is the power of strength in each step of progress towards the successful completion of the project.

We owe a whole hearted sense of reverence and gratitude to our project guide **Dr. P. Seethalakshmi B.E, M.E, Ph.D.,** Assistant Professor, for her efficient and excellent guidance, inspiring discussion, insightful and timely encouragement for the successful completion of the project.

We express our thanks to our Head Of the Department of Electronics and Communication Engineering, **Dr. N. Kumaresan B.E, M.E, Ph.D.,** for giving us the opportunity and providing us with adequate infrastructure and congenial environment.

We thank our Dean, **Dr. M. SARAVANAKUMAR, M.B.A, Ph.D.,** Anna University Regional Campus, Coimbatore for his great support with blessings.

We also extended our heartfelt thanks to all faculties and staff of ECE department who have rendered their valuable help in making this project successful.

KOWSALYA S **710020106016**

NIVAASHINI S **710020106019**

RASHMIYA J **710020106022**

ASWINKUMAR S **710020106303**

ABSTRACT

Atrial Fibrillation (AF) is the most common form of arrhythmia. AF is referred to as rapid heart-beat. AF is conventionally diagnosed by monitoring Electrocardiogram (ECG) signals. This method of diagnosis is time consuming for Cardiologists. In order to address this challenge many Machine Learning (ML) models have been proposed. In this method proposed, the ECG signals are taken as three data-sets, one MIT-BIH dataset from Physio net for training the model. The dataset is pre-processed and from this pre-processed signal, features of ECG are extracted using the Convolutional Neural Network (CNN) algorithm. The extracted features are trained to the model using the U- NET model. This trained model helps in detecting if patients are affected by Atrial Fibrillation. The other two datasets are used in predicting if the patients are affected by Atrial Fibrillation. Testing this at an early stage of AF (Paroxysmal AF) reduces the risk for life. This method is aimed to achieve an accuracy of 96.89% and F1 score of 0.969.

ஆய்வு சுருக்கம்

ஏட்ரியல் ஃபைப்ரிலேஷன் (AF) என்பது அரித்மியாவின் மிகவும் பொதுவான வடிவமாகும். AF என்பது விரைவான இதயத் துடிப்பு என குறிப்பிடப்படுகிறது. எலக்ட்ரோ கார்டியோகிராம் (ECG) சிக்னல்களை கண்காணிப்பதன் மூலம் AF வழக்கமாக கண்டறியப்படுகிறது. இந்த நோயறிதல் முறை இருதயநோய் நிபுணர்களுக்கு நேரத்தை எடுத்துக்கொள்ளும். இந்த சவாலை எதிர்கொள்ள பல இயந்திர கற்றல் (ML) மாதிரிகள் முன்மொழியப்பட்டுள்ளன. முன்மொழியப்பட்ட இந்த முறையில், ECG சிக்னல்கள் மாதிரியைப் பயிற்றுவிப்பதற்காக பிசியோ நெட்டில் இருந்து ஒரு MIT-BIH தரவுத்தொகுப்பு மூன்று தரவுத் தொகுப்புகளாக எடுத்துக் கொள்ளப்படுகின்றன. தரவுத்தொகுப்பு முன்பே செயலாக்கப்பட்டது மற்றும் இந்த முன் செயலாக்கப்பட்ட சமிக்ஞையிலிருந்து, ECG இன் அம்சங்கள் கன்வல்யூஷனல் நியூரல் நெட்வொர்க் (CNN) அல்காரிதம் மூலம் பிரித்தெடுக்கப்படுகின்றன. பிரித்தெடுக்கப்பட்ட அம்சங்கள் U-NET மாதிரியைப் பயன்படுத்தி மாடலுக்குப் பயிற்சியளிக்கப்படுகின்றன. ஏட்ரியல் ஃபைப்ரிலேஷனால் நோயாளிகள் பாதிக்கப்படுகிறார்களா என்பதைக் கண்டறிய இந்தப் பயிற்சியளிக்கப்பட்ட மாதிரி உதவுகிறது. நோயாளிகள் ஏட்ரியல் ஃபைப்ரிலேஷனால் பாதிக்கப்படுகிறார்களா என்பதைக் கணிப்பதில் மற்ற இரண்டு தரவுத்தொகுப்புகள் பயன்படுத்தப்படுகின்றன. AF (Paroxysmal AF) இன் ஆரம்ப நிலையிலேயே இதைச் சோதிப்பது

உயிருக்கான ஆபத்தைக் குறைக்கிறது. இந்த முறை 96.89% துல்லியத்தையும் F1 மதிப்பெண் 0.969ஐயும் அடைவதை நோக்கமாகக் கொண்டுள்ளது.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ML	MACHINE LEARNING
DL	DEEP LEARNING
AF	ATRIAL FIBRILLATION
NAF	NON-ATRIAL FIBRILLATION
ECG	ELECTROCARDIOGRAM
MIMIC	MEDICAL INFORMATION MART FOR INTENSIVE CARE
CNN	CONVOLUTION NEURAL NETWORK
U-NET	U SHAPED NEURAL NETWORK
SVM	SUPPORT VECTOR MACHINE
LSTM	LONG-SHORT TERM MEMORY
RF	RANDOM FOREST
PAC	PREMATURE ATRIAL CONTRACTIONS
PVC	PREMATURE VENTRICULAR CONTRACTIONS
DHR	DIFFERENCE OF HEART RATE
DWT	DISCRETE WAVELET TRANSFORM
NSR	NORMAL SINUS RHYTHM
VT	VENTRICULAR TACHYCARDIA
VF	VENTRICULAR FIBRILLATION
AFL	ATRIAL FLUTTER
DNN	DEEP NEURAL NETWORKS

CHAPTER 1

INTRODUCTION

1.1 ATRIAL FIBRILLATION - OVERVIEW

Atrial fibrillation (AF) stands as one of the most prevalent cardiac arrhythmias globally. Characterized by erratic electrical impulses in the atria, AF disrupts the heart's normal rhythm, leading to symptoms ranging from palpitations to more severe complications like stroke and heart failure. Typical waveform of ECG in sinus rhythm is composed of the P-wave, QRS-complex, and T-wave. However, in AF, chaotic fibrillatory waves can affect the morphology of P-waves, and the RR-interval is not constant among beats.

Atrial Fibrillation typically identified by analysing the ECG signal of the patients manually. This process of detection is time-consuming for medical consultants. In order to reduce the time in detection of Atrial Fibrillation, engineers have played a crucial role in developing Machine Learning Models. These models reduce the time complexity in detecting if patients are AF detected or not.

1.2 CNN IN AF DETECTION:

Convolutional Neural Networks (CNNs) are a class of deep neural networks primarily used for visual recognition tasks such as image classification, object detection, and segmentation. They are inspired by the human visual system and have revolutionized the field of computer vision due to their ability to automatically learn hierarchical features directly from raw input data.

Recent advances in deep learning, particularly Convolutional Neural Networks (CNNs), have shown remarkable promise in various medical imaging and signal processing tasks. CNNs excel in automatically learning hierarchical features directly from raw data, making them well-suited for analysing complex physiological signals like electrocardiograms (ECGs). ECGs, routinely acquired in clinical settings, offer a wealth of information regarding cardiac electrical activity. Leveraging CNNs to extract subtle yet clinically relevant features from ECG data could revolutionize Atrial Fibrillation detection, enabling early identification and intervention. CNNs have the potential to automatically learn discriminative features directly from raw ECG data, eliminating the need for manual feature engineering and enhancing diagnostic accuracy.

In recent years, many Machine Learning methods for detection of AF have been developed. Among these approaches CNN and LSTM are commonly used to achieve an appropriate performance by feature extraction. Many commonly used ML algorithms include Support Vector Machine (SVM) and Random Forest (RF) that could achieve required performance by feature extraction and training the model to predict AF. Compared to traditional Machine Learning methods, Deep Learning (DL) and neural networks can automatically extract features from the ECG signal.

1.3 ELECTROCARDIOGRAM SIGNAL

Electrocardiogram (ECG) signals are used as input for training a Machine Learning model. After training, the model's performance is tested to assess its ability to classify or predict outcomes based on ECG data, aiming to advance atrial fibrillation monitoring and diagnosis of Atrial Fibrillation.

1.3.1 NORMAL ECG SIGNAL

An electrocardiogram (ECG) is a diagnostic test that records the electrical activity of the heart over a period of time. It is a non-invasive procedure commonly used to detect and diagnose heart conditions. A normal ECG signal represents the typical electrical activity of a healthy heart.

A standard ECG tracing consists of several components:

1. P Wave: The P wave represents atrial depolarization, which is the electrical impulse that triggers the contraction of the atria. It is a small, rounded wave that precedes the QRS complex.

2. QRS Complex: The QRS complex represents ventricular depolarization, which is the electrical impulse that triggers the contraction of the ventricles. It consists of three waves: the Q wave, R wave, and S wave. The QRS complex is typically larger and more prominent than the P wave.

3. T Wave: The T wave represents ventricular repolarization, which is the recovery phase of the ventricles as they prepare for the next contraction. It is a rounded wave that follows the QRS complex.

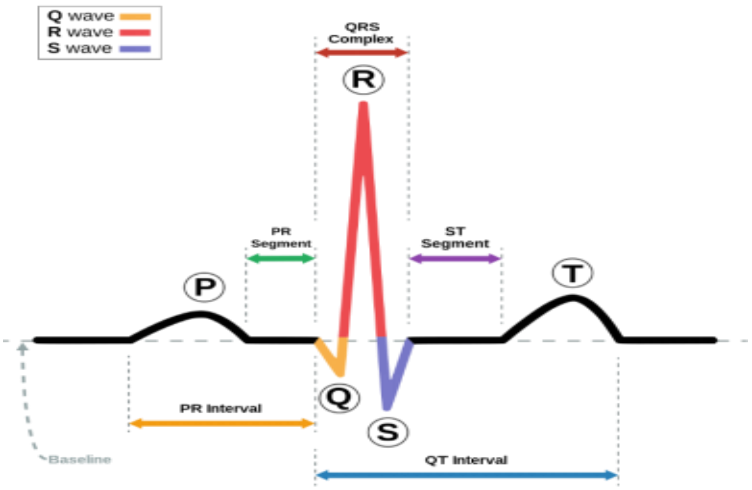

Figure 1.1 Normal ECG Signal

A normal ECG signal typically exhibits consistent waveforms and intervals, with each component occurring in a specific sequence and within normal ranges of duration and amplitude. Deviations from these normal patterns may indicate underlying cardiac abnormalities or conditions, such as arrhythmias, conduction disorders, or ischemic heart disease.

In a normal heartbeat rhythm, the heart typically beats at a rate of 60 to 100 beats per minute (bpm). This range represents the typical variation in heart rate among individuals and reflects the normal functioning of the heart's electrical system.

1.3.2 ABNORMAL ECG SIGNAL

Abnormal ECG signals encompass a range of irregular cardiac rhythms and waveforms indicative of underlying cardiac pathology. These abnormalities may manifest as deviations from the typical ECG pattern observed in normal sinus rhythm (NSR).

Atrial Fibrillation (AF): AF is characterized by irregular and often rapid heartbeats originating from the atria. The ECG tracing in AF shows irregular R-R intervals, absence of distinct P waves, and fibrillatory waves. Heart rate during AF can vary widely, typically ranging from 100 to 175 beats per minute.

Detecting and interpreting these abnormalities on the ECG is crucial for diagnosing underlying cardiac conditions and guiding appropriate treatment interventions. Close monitoring and prompt medical intervention are essential for individuals with abnormal ECG signals to prevent adverse cardiac events and improve outcomes.

Table 1.1 ECG Signal Feature and its Description

Feature	Description	Duration
P wave	The P wave represents depolarization of the atria. Atrial depolarization spreads from the SA node towards the AV node, and from the right atrium to the left atrium.	<80 ms
PR interval	The PR interval is measured from the beginning of the P wave to the beginning of the QRS complex. This interval reflects the time the electrical impulse takes to travel from the sinus node through the AV node.	120 to 200 ms
QRS complex	The QRS complex represents the rapid depolarization of the right and left ventricles. The ventricles have a large muscle mass compared to the atria, so the QRS complex usually has a much larger amplitude than the P wave.	80 to 100 ms
J-point	The J-point is the point at which the QRS complex finishes and the ST segment begins.	
ST segment	The ST segment connects the QRS complex and the T wave; it represents the period when the ventricles are depolarized.	

T wave	The T wave represents the repolarization of the ventricles. It is generally upright in all leads except aVR and lead V1.	160 ms
Corrected QT interval (QTc)	The QT interval is measured from the beginning of the QRS complex to the end of the T wave. Acceptable ranges vary with heart rate, so it must be corrected to the QTc by dividing by the square root of the RR interval.	<440 ms
U wave	The U wave is hypothesized to be caused by the repolarization of the interventricular septum. It normally has a low amplitude, and even more often is completely absent.	

1.4 OBJECTIVES

The ultimate goal is to develop a machine learning-based algorithm for automatic detection of AF episodes in short-term Electrocardiogram (ECG) recordings.

- Reduce the time required for manual detection of Atrial Fibrillation in short-term ECG data by at least 50% compared to current practice.
- Train the machine learning model using a dataset containing Atrial Fibrillation taken from ECG recordings.
- Maximize accuracy and sensitivity in identifying Atrial Fibrillation from ECG data.

1.5 CHAPTER ORGANIZATION

Report consists of 7 chapters that are arranged as follows,

Chapter 2 – Discusses about the literature review related to the work.

Chapter 3 – Discusses about the ECG signal Dataset used.

Chapter 4 – Discusses about the Methodology and Implementation of the model.

Chapter 5 – Gives details about the Hardware used and its specification.

Chapter 6 – Gives details about the Software used and its specification.

Chapter 7 – Discusses about extraction of real-time ECG signal.

Chapter 8 – Discusses about the performance metrics.

Chapter 9 – Discuses the Result of the proposed model.

Chapter 10 – Concludes the project.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

To detect early-stage Atrial Fibrillation (AF), various machine learning (ML) and deep learning (DL) algorithms are employed. These models analyze ElectroCardioGram (ECG) data, aiming for accurate AF detection, enhancing early intervention possibilities for better patient outcomes. Each topic explains the description of a particular method proposed by each author.

2.2 SEMI-SUPERVISED LEARNING FOR AUTOMATIC ATRIAL FIBRILLATION DETECTION IN 24-HOUR HOLTER MONITORING

Peng et al., (2022) proposed a novel approach to automatically detect Atrial Fibrillation (AF) using ElectroCardioGram (ECG) data. They developed a semi-supervised learning method that generated modified low-entropy labels for unlabelled samples, enabling the training of a deep learning model. This model, consisting of a 1D CNN-LSTM architecture, processed RR intervals from ECG signals as input. Remarkably, their method achieved a sensitivity of 97.8%, specificity of 97.9%, and accuracy of 97.9% in detecting paroxysmal AF using only labelled samples from 10 patients for training. This study highlights the potential of semi-supervised learning in enhancing AF detection accuracy with limited labelled data.

2.3 ROBUST DETECTION OF ATRIAL FIBRILLATION FROM SHORT-TERM ELECTROCARDIOGRAM USING CONVOLUTIONAL NEURAL NETWORKS

Siti Nurmaini et al., (2020) proposed a straightforward algorithm integrating Discrete Wavelet Transform (DWT) with one-dimensional convolutional neural networks (1D-CNNs) to classify three classes: Normal Sinus Rhythm (NSR), Atrial Fibrillation (AF), and non-AF (NAF). Their study utilized a combination of three public datasets and one dataset from an Indonesian hospital. The model's robustness was assessed using validation data from four datasets, demonstrating superior performance compared to alternative methods. The 1D-CNNs exhibited high generalization ability, achieving notable accuracy, sensitivity, specificity, precision, and F1-Score for both two-class and three-class classifications. With accuracy rates of 98%, this approach holds promise for enhancing AF diagnosis in clinical settings and patient self-monitoring, facilitating early detection and effective treatment strategies.

2.4 NOVEL DENSITY POINCARE PLOT BASED MACHINE LEARNING METHOD TO DETECT ATRIAL FIBRILLATION FROM PREMATURE ATRIAL/VENTRICULAR CONTRACTIONS

Syed Khairul Bashar et al., (2020) proposed a novel density Poincare plot-based machine learning approach for Atrial Fibrillation (AF) detection from Premature Atrial Contractions (PAC) and Premature Ventricular Contractions (PVC) using electrocardiogram (ECG) recordings. Their method involves generating a density Poincare plot from the Difference of Heart Rate (DHR), capturing overlapping phase-space trajectory information. Image processing techniques such as statistical central moments, template correlation, Zernike moment, discrete wavelet transform, and Hough transform are applied to extract

relevant features from the density Poincare plot. An infinite latent feature selection algorithm ranks these features, followed by classification using K-Nearest Neighbour, Support Vector Machine (SVM), and Random Forest (RF) classifiers. Validation on a subset of the Medical Information Mart for Intensive Care (MIMIC) III database yielded promising results, with SVM achieving 98.99% sensitivity, 95.18% specificity, and 97.45% accuracy during segment-wise 10-fold cross-validation. RF attained the highest accuracy of 91.93% in a subject-wise scenario. Further validation on two additional databases demonstrated 100% PAC detection accuracy without requiring additional training. This method holds potential for enhancing AF detection using wearable armband ECG data and Physionet AFPDB, showcasing its versatility and effectiveness in real-world applications.

2.5 A REVIEW ON THE STATE OF THE ART IN ATRIAL FIBRILLATION DETECTION ENABLED BY MACHINE LEARNING

Ali Rizwan et al., (2019) reviewed and presented in a systematic way covering pertinent concepts and key steps involved in developing a machine learning based solution for AF detection. This paper presents a summarized overview of common causes of AF and risks associated. It provides a brief introduction of AF and its important types for an audience foreign to the concept. It lists popular methods and equipment used for AF data collection. It identifies, reviews and discusses key biomarkers relevant for the design of machine learning driven AF detection models. It elaborates with use cases, each key step involved in data pre-processing and machine learning model development for the detection and prediction of AF.

2.6 SUMMARY

Based on the review made on the various conference papers, journals and book materials it has been concluded as follows

• These literature surveys demonstrate the familiarity with our topic and scholarly context.

• It develops a theoretical framework and methodology for our research.

Table 2.1 Literature Survey

S.NO.	PAPER NAME	AUTHOR NAME	PUBLI-SHED YEAR	ALGORI-THM USED	DATASET	IDE PLAT-FORM	ACCRACY
1.	Semi-Supervised Learning for Automatic Atrial Fibrillation Detection in 24-Hour Holter Monitoring https://ieeexplore.ieee.org/document/9772345	Peng Zhang , Member, IEEE, Yuting Chen, Fan Lin , Sifan Wu, Xiaoyun Yang , and Qiang Li	2022	CNN, LSTM	Long term dynamic 12-lead ECG recordings collected from 1000 patients with paroxysmal AF from the Cardiac Function Examination Center	Python	97.9%
2.	Novel Density Poincare Plot Based Machine Learning Method to Detect Atrial Fibrillation from Premature Atrial/Ventricular Contractions https://pubmed.ncbi.nlm.nih.gov/32746035/	Syed Khairul Bashar, Student Member, IEEE, Dong Han, Student Member, IEEE, Fearass Zieneddin, Eric Ding, Timothy P. Fitzgibbons, Allan J. Walkey, David D. McManus, Bahram Javidi, Fellow, IEEE and Ki H. Chon	2020	SVM, RF	MIMIC III Waveform, wearable armband ECG data and the Physionet AFPDB	Python	91.93%

3.	Detection of Atrial Fibrillation from Variable-Duration ECG Signal Based on Time-Adaptive Densely Network and Feature Enhancement Strategy https://ieeexplore.ieee.org/document/9946392	X. Zhang, M. Jiang, K. Polat, A. Alhudhaif, J. Hemanth and W. Wu	2020	CNN	MIT-BIH Atrial Fibrillation, Physionet Atrial Fibrillation	Python	87.98%
4.	DDCNN: A Deep Learning Model for AF Detection From a Single-Lead Short ECG Signal https://ieeexplore.ieee.org/document/9832485	Zhaocheng Yu, Junxin Chen , Senior Member, IEEE, Yu Liu, Yongyong Chen, Tingting Wang	2022	CNN, LSTM	MIT-BIH database from physio net	Python	93%

CHAPTER 3

DATASET

3.1 INTRODUCTION

There are three datasets used in this project. One, for training the model about the features of Atrial Fibrillation Signal. Other two datasets are used in predicting if the model is trained accurately and effectively to detect Atrial Fibrillation. One of the two datasets used in prediction contains signals of patients affected by AF and the other one contains Normal ECG signals.

3.2 MIT-BIH ATRIAL FIBRILLATION DATABASE

MIT-BIH Atrial Fibrillation database includes 25 long-term ECG recordings of human subjects with atrial fibrillation (mostly paroxysmal).Of these, 23 records include the two ECG signals (in the .dat files); records 00735 and 03665 are represented only by the rhythm (.atr) and unaudited beat (.qrs) annotation files.

The individual recordings last for 10 hours each and include two ECG signals sampled at 250 samples per second with a resolution of 12 bits, covering a range of ±10 millivolts. These recordings were originally conducted at Boston's Beth Israel Hospital, now known as the Beth Israel Deaconess Medical Center, using ambulatory ECG recorders with a typical recording bandwidth ranging from approximately 0.1 Hz to 40 Hz.

The rhythm annotation files, denoted with the suffix .atr, were manually prepared and include annotations for rhythms such as atrial fibrillation (AF),

atrial flutter (AFL), AV junctional rhythm (J), and all other rhythms marked as (N). The original rhythm annotation files used AF, AFL, J, and N, but have been revised for consistency with the MIT-BIH Arrhythmia Database.

Beat annotation files, identified with the suffix .qrs, were generated using an automated detector and have not undergone manual correction. For some records, manually corrected beat annotation files, indicated by the suffix .qrsc, are available. The .qrs annotations may be valuable for studies involving automated AF detection, where robustness to typical QRS detection errors is necessary. However, for basic studies focusing on AF itself, the .qrsc annotations may be preferred as they account for QRS detection errors, which could otherwise confound the results. It's important to note that in both .qrs and .qrsc files, all beats are labelled as if normal, without distinguishing between beat types.

3.3 LONG TERM AF DATABASE

This database includes 84 long-term ECG recordings of subjects with paroxysmal or sustained atrial fibrillation (AF). Each record contains two simultaneously recorded ECG signals digitized at 128 Hz with 12-bit resolution over a 20 mV range; record durations vary but are typically 24 to 25 hours.

Two sets of annotations are available here:

- The qrs annotations were produced by an automated QRS detector; in these annotation files, all detected beats (including occasional ventricular ectopic beats) are labelled N, detected artifacts are labelled '|', and AF terminations are labelled T.
- The atr annotations were obtained by manual review of the output of an automated ECG analysis system (see below); in these annotation files, all detected beats are labelled by type, and rhythm changes are also annotated.

The AF Termination Challenge Database consists of 80 one-minute excerpts of a subset of these records (those numbered 00 through 75). As a guide to selecting the excerpts, “T” was inserted manually in the qrs annotation files of this subset of the records, to mark spontaneous terminations of AF episodes with durations of at least one minute.

3.4 MIT-BIH NORMAL SINUS RHYTHM DATABASE

This database includes 18 long-term ECG recordings of subjects referred to the Arrhythmia Laboratory at Boston's Beth Israel Hospital (now the Beth Israel Deaconess Medical Center). Subjects included in this database were found to have had no significant arrhythmias; they include 5 men, aged 26 to 45, and 13 women, aged 20 to 50. The recordings, spanning 10-hour intervals, capture two ECG signals sampled at 250 samples per second with 12-bit precision, encompassing a range of ±10 millivolts.

CHAPTER 4

METHODOLOGY

4.1 INTRODUCTION:

The proposed methodology takes advantage of the state-of-the-art signal processing and Deep Learning algorithms and consists of the four main steps discussed below.

Data Preprocessing: Raw ECG signals will be pre-processed to remove noise, baseline wander, and other artifacts that may interfere with the detection of Atrial Fibrillation.

Feature Extraction: A CNN architecture is designed to extract relevant features directly from pre-processed ECG signals. The CNN will consist of multiple layers, including convolutional layers for feature extraction and pooling layers for dimensionality reduction.

Training Phase: The CNN will be trained using a dataset of annotated ECG recordings, where Atrial Fibrillation labels are provided. We have employed supervised learning techniques to optimize the CNN parameters and learn discriminative features for Atrial Fibrillation detection.

Testing and Validation: The trained CNN is evaluated using a separate test dataset to assess its performance in detecting Atrial Fibrillation. Performance based on accuracy will be calculated to quantify the system's diagnostic capability.

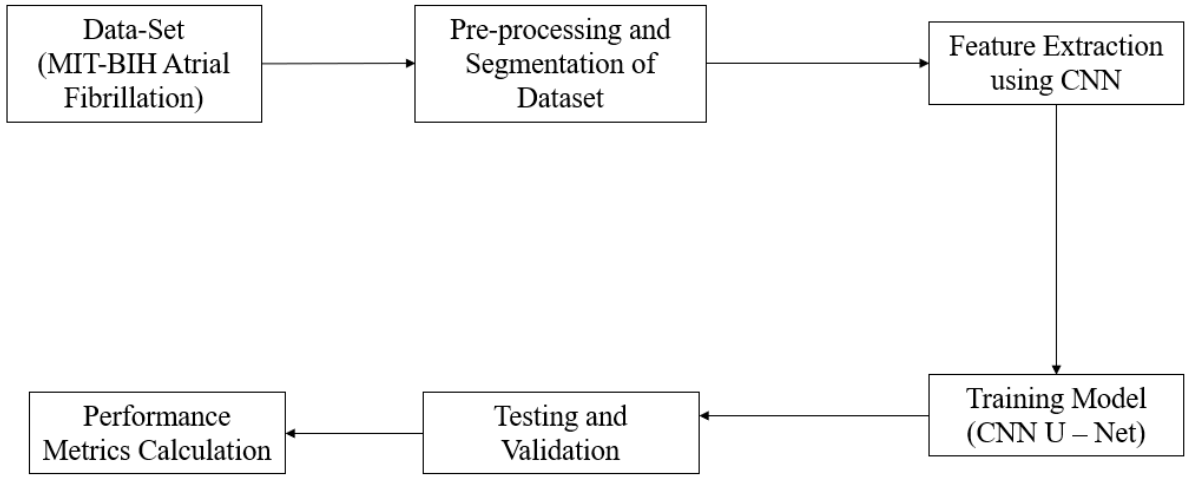

Figure 4.1 Block Diagram for Training the model for Atrial Fibrillation Detection

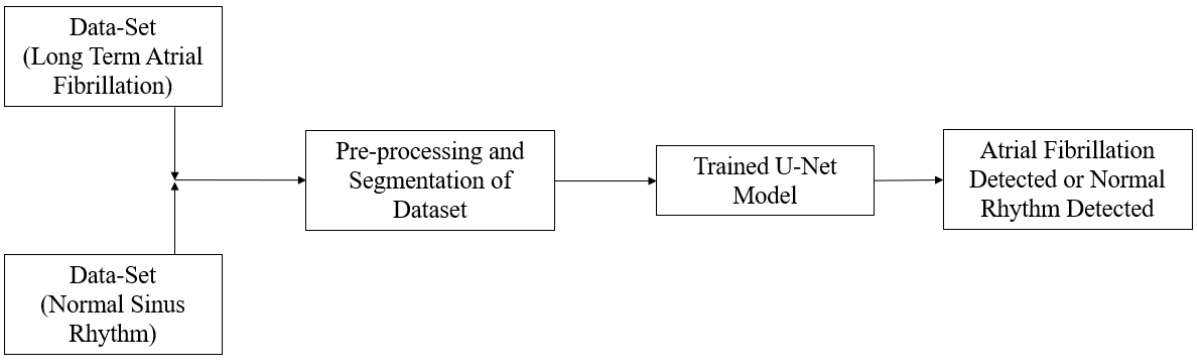

Figure 4.2 Block Diagram for Prediction and Detection of Atrial Fibrillation

4.2 DATA PREPROCESSING

The preprocessing phase of electrocardiogram (ECG) signals is essential for refining raw data before analysis. This phase comprises three main processes: normalization, noise removal, and segmentation.

Normalization is crucial as it standardizes the scale of different features within the ECG signal. Raw ECG signals often contain features with varying magnitudes, making it challenging to compare and analyse them accurately. By scaling all features to a uniform range, typically between 0 and 1, normalization ensures consistent analysis across different parts of the signal. This step not only facilitates easier interpretation but also reduces computational complexity, potentially improving the efficiency of subsequent algorithms and analyses.

Noise removal is another critical preprocessing step, aiming to eliminate unwanted artifacts and disturbances from the ECG signal. Various types of noise, such as baseline wander and power line interference, can obscure the underlying cardiac activity, affecting the accuracy of diagnostic interpretations. Denoising techniques are applied to attenuate these unwanted components while preserving the integrity of the ECG waveform. Filters, wavelet transforms, and adaptive algorithms are commonly used for noise removal, tailored to address specific types of noise without distorting the essential features of the cardiac signal.

Segmentation involves dividing the ECG signal into discrete cardiac cycles or segments, typically by identifying the onset and offset points of each heartbeat. Accurate segmentation is crucial for isolating individual cardiac events, such as P waves, QRS complexes, and T waves, which carry diagnostic significance. By segmenting the signal, clinicians and researchers can analyze each cardiac cycle independently, allowing for detailed examination of rhythm and morphology. Additionally, segmentation facilitates tasks such as heart rate calculation and arrhythmia detection, which rely on precise identification of cardiac events within the ECG waveform.

4.3 FEATURE EXTRACTION

The input data for machine learning algorithms aimed at detecting or predicting AF comprises features derived from the ECG signal. Features are extracted from specific patterns observed in the ECG of AF patients. These features, such as statistical attributes of the f-wave, RR interval, and QRS complexes, serve as inputs for ML algorithms tasked with AF detection.

The main AF characteristics based on irregularity in atrial activity or ventricular response are as follow:

The signature wave of atrial activity known as P-wave is absent or replaced with f-wave. f-wave reflects irregular atrial beat rate that varies between 240 and 540 with an average of 350 beats/min. It is a low amplitude fibrillatory wave.

QRS complex, representing the ventricular response on the ECG, is disturbed. It is mainly reflected by irregular RR intervals also known as “irregularly irregular”. For their measurement the first task is to detect the QRS complex, then detect R-peaks in QRS complex, and then measure the duration between consecutive R peaks. In research, therefore, machine learning algorithms for AF detection commonly use features extracted from the ECG data of atrial activity, ventricular response or combination of both.

Convolutional Neural Networks (CNNs) excel in medical imaging and signal processing tasks by autonomously discerning intricate patterns from raw data. This adaptability renders them well-suited for interpreting complex physiological signals such as electrocardiograms (ECGs). ECGs, regularly acquired in clinical settings, offer profound insights into cardiac electrical activity. Leveraging CNNs enables the extraction of subtle yet clinically relevant features directly from ECG data.

CNNs' effectiveness lies in their hierarchical feature extraction capabilities, allowing them to identify nuanced patterns indicative of cardiac abnormalities or pathologies. By analysing the temporal and spatial relationships within ECG signals, CNNs can detect subtle deviations from normal cardiac activity, which may signify conditions like atrial fibrillation or ventricular arrhythmias.

4.4 NATIVE U-NET MODEL ARCHITECTURE

4.4.1 SUPERVISED LEARNING

Supervised learning is a category of machine learning that uses labelled datasets to train algorithms to predict outcomes and recognize patterns. Supervised machine learning algorithms make it easier for organizations to create complex models that can make accurate predictions. The data used in supervised learning is labelled — meaning that it contains examples of both inputs (called features) and correct outputs (labels). The algorithms analyze a large dataset of these training pairs to infer what a desired output value would be when asked to make a prediction on new data.

4.4.2 U-NET MODEL

U-Net is a widely used deep learning architecture that was first introduced in the “U-Net: Convolutional Networks for Biomedical Image Segmentation”. The architecture of U-Net is unique in that it consists of a contracting path and an expansive path. The contracting path contains encoder layers that capture contextual information and reduce the spatial resolution of the input, while the expansive path contains decoder layers that decode the encoded data and use the

information from the contracting path via skip connections to generate a segmentation map.

The contracting path in U-Net is responsible for identifying the relevant features in the input image. The encoder layers perform convolutional operations that reduce the spatial resolution of the feature maps while increasing their depth, thereby capturing increasingly abstract representations of the input. This contracting path is similar to the feedforward layers in other convolutional neural networks. On the other hand, the expansive path works on decoding the encoded data and locating the features while maintaining the spatial resolution of the input. The decoder layers in the expansive path upsample the feature maps, while also performing convolutional operations. The skip connections from the contracting path help to preserve the spatial information lost in the contracting path, which helps the decoder layers to locate the features more accurately.

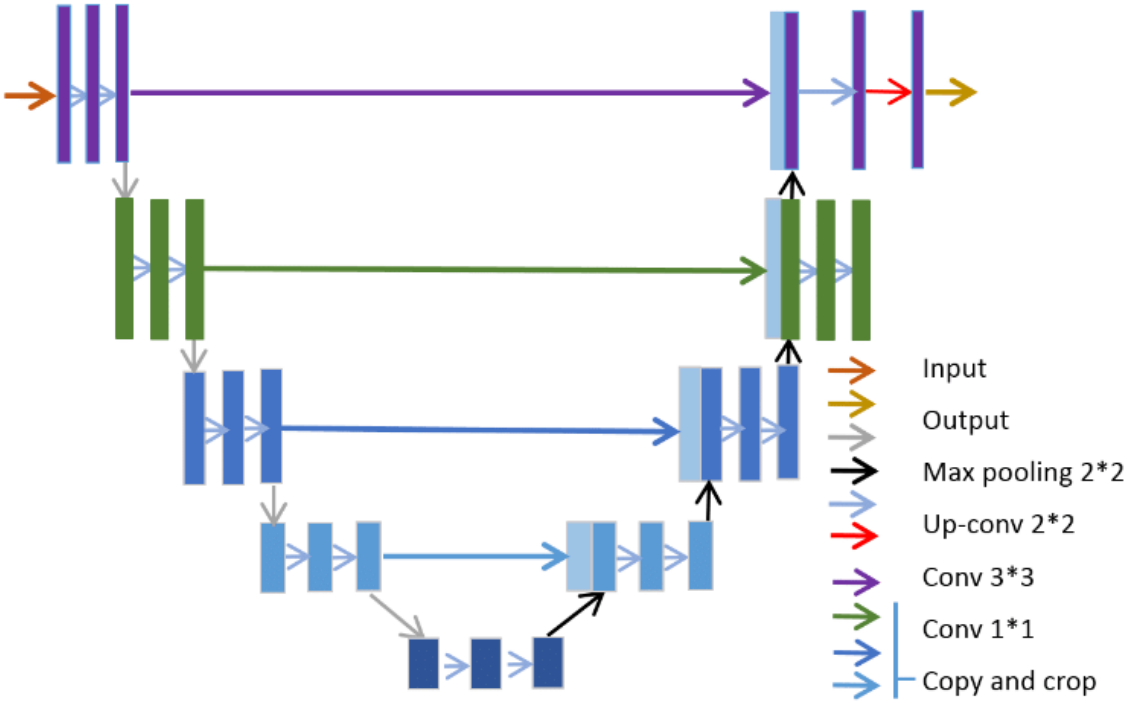

Figure 4.1 Native U-NET Model Architecture

4.5 TRAINING PHASE

The U-NET model, comprising 26 layers, undergoes training using patient data collected over a 10-hour monitoring period, totalling one million data points per patient. To facilitate efficient model training, this extensive dataset is segmented into 128 smaller sections. Following segmentation, the dataset is then partitioned into training, validation, and testing subsets. Specifically, 60% of the data is allocated for training, while 40% is split between validation and testing. Within this 40%, 20% is used for validation, and another 20% is reserved for testing. The dataset is divided in this way, in order to maintain certain model's effective performance. This meticulous division ensures that the model is effectively trained and evaluated on diverse data samples, enhancing its robustness and accuracy in medical data analysis. The U-Net model architecture consists of a total of 26 layers distributed across its different components. Specifically, the encoder segment comprises 12 layers responsible for feature extraction and representation. Following the encoder, there are 2 bottleneck layers designed to compress and abstract the learned features. Subsequently, the decoder section unfolds with 11 layers aimed at reconstructing the spatial information. Lastly, a single layer in the output stage synthesizes the final prediction or output. Collectively, these components contribute to the overall architecture, totalling 26 layers in the U-Net model.

4.6 TESTING AND VALIDATION

Once the Convolutional Neural Network (CNN) has been trained, it undergoes evaluation using an independent test dataset to gauge its proficiency in identifying Atrial Fibrillation. This evaluation step is pivotal as it offers

insights into the model's operational effectiveness and trustworthiness when deployed in practical settings. To ascertain the CNN's diagnostic prowess accurately, performance metrics predominantly centered on accuracy will be computed. Accuracy provides a fundamental measure of the model's overall correctness in classifying instances of atrial fibrosis within the test dataset. However, to gain a deeper and more nuanced perspective on the CNN's performance, supplementary evaluation metrics such as precision, recall, and F1 score may also be employed. Precision indicates the proportion of correctly identified instances of atrial fibrosis among all instances labelled as positive by the model. Recall, on the other hand, denotes the proportion of correctly identified instances of atrial fibrosis among all actual instances of atrial fibrosis present in the dataset. F1 score, which is the harmonic mean of precision and recall, synthesizes these two metrics into a single numerical value, offering a balanced assessment of the model's performance across both precision and recall. By leveraging these evaluation metrics in tandem with accuracy, a comprehensive evaluation of the CNN's diagnostic capabilities can be attained, thereby facilitating informed decision-making regarding its deployment and utilization in real-world clinical scenarios.

Table 4.1 Performance Metrics

Performance Metrics	**Proposed Model**
Accuracy	96.5%
Precision	0.966
Recall	0.965
F1 Score	0.964
Specificity	0.996

CHAPTER 5
HARDWARE SPECIFICATION

5.1 ARDUINO UNO SPECIFICATION

UNO is a microcontroller board based on 8-bit ATmega328P microcontroller. Along with ATmega328P, it consists other components such as crystal oscillator, serial communication, voltage regulator, etc. to support the microcontroller.

The Arduino Uno comes with USB interface, 6 analog input pins, 14 I/O digital ports that are used to connect with external electronic circuits. Out of 14 I/O ports, 6 pins can be used for PWM output. It allows the designers to control and sense the external electronic devices in the real world.

The software used for Arduino devices is called IDE (Integrated Development Environment) which is free to use and required some basic skills to learn it. It can be programmed using C and C++ language.

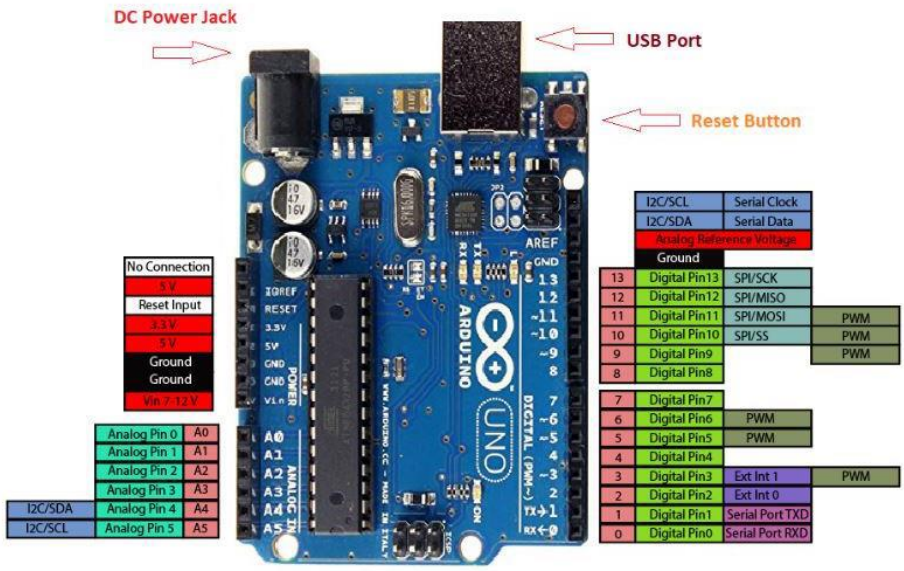

Figure 5.1 Arduino Uno PIN Description

5.2 ARDUINO IDE

The Integrated Development Environment (IDE) is a simple and easy to learn software for writing Arduino codes. Arduino has huge flexibility with which you can make almost anything you imagine. It can be easily connected to a variety of modules like fire sensors, obstacle sensors, presence detectors, GPS modules, GSM Modules, or anything with which you wish to give wings to your dream project.

IDE is equally compatible with Windows, MAC or Linux Systems. However, Windows is preferable to use. It is an open-source platform where anyone can modify and optimize the board based on the number of instructions and task they want to achieve.

5.3 ECG SENSOR – AD8232

An AD8232 sensor is used to calculate the electrical activity of the heart. This is a small chip and the electrical action of this can be charted like an ECG (Electrocardiogram). Electrocardiography can be used to help in diagnosing different conditions of the heart. This article provides an overview of the AD8232 ECG Sensor.

The AD8232 ECG sensor is a commercial board used to calculate the electrical movement of the human heart. This action can be chart like an Electrocardiogram and the output of this is an analog reading. Electrocardiograms can be very noisy, so to reduce the noise the AD8232 chip can be used. The working principle of the ECG sensor is like an operational amplifier to help in getting a clear signal from the intervals simply.

The heart rate monitoring sensor like AD8232 includes the pins like SDN pin, LO+ pin, LO- pin, OUTPUT pin, 3.3V pin, and GND pin. So that we can connect this IC to development boards like Arduino by soldering pins.

Additionally, this board includes pins like the right arm (RA), left arm (LA) & right leg (RL) pins to connect custom sensors. An LED indicator in this board

is used to indicate the heartbeat rhythm of humans. The AD8232 sensor comprises a function like quick restore, used to decrease the length of long resolving tails.

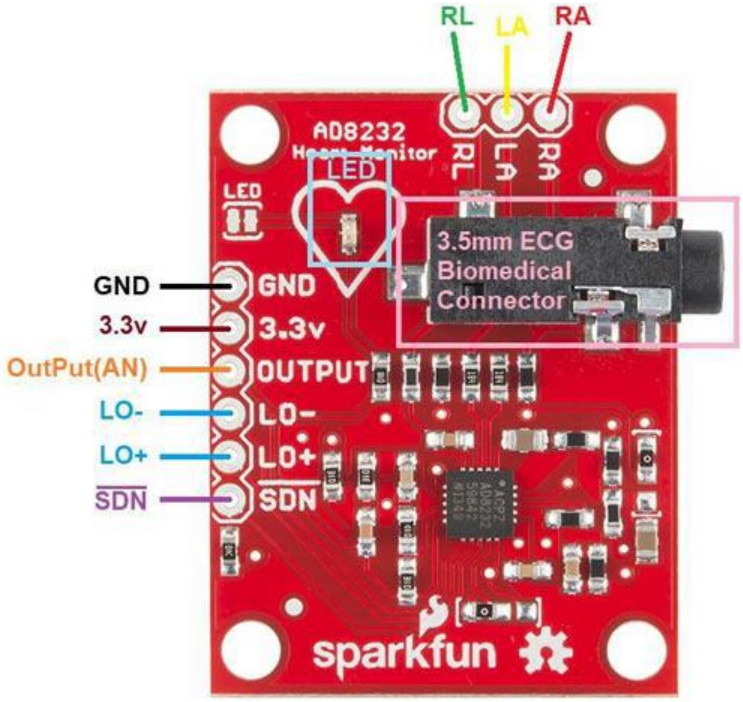

Figure 5.2 AD8232 PIN Description

CHAPTER 6
SOFTWARE SPECIFICATION

6.1 GOOGLE COLAB (PYTHON DISTRIBUTION)

Google Colab, short for Google Colaboratory, is an online platform provided by Google that allows users to write and execute Python code in a browser-based environment. Google Colab is a powerful and versatile platform that provides free access to computational resources and makes it easy for users to write, run, and share Python code in a collaborative environment. It's particularly popular among data scientists, machine learning engineers, and researchers who need access to powerful computing resources for their projects.

6.2 OVERVIEW

Google Colab offers free access to powerful computational resources, including CPU, GPU, and TPU instances, catering to users tackling machine learning, data analysis, and other demanding tasks. It seamlessly integrates with Jupyter notebooks, providing an interactive coding environment ideal for tasks like data exploration, prototyping, and collaborative code sharing. With built-in collaboration features, users can share notebooks for real-time editing and feedback exchange, enhancing teamwork efficiency. Integration with Google Drive enables easy storage, access, and sharing of Colab notebooks across devices, simplifying organization. Google Colab comes with a comprehensive selection of pre-installed Python libraries such as TensorFlow, PyTorch, and more, with the option to install additional packages using pip or conda. As a Google product, Colab seamlessly integrates with various Google services like Drive, Sheets, and Cloud Platform, augmenting its versatility and utility for users.

6.3 INTEGRATION WITH GOOGLE DRIVE

Google Colab integrates seamlessly with Google Drive, allowing users to save their Colab notebooks directly to their Google Drive account. This makes it easy to access and share notebooks across devices, and it also provides a convenient way to store and organize your code and data.

6.4 SUPPORT FOR LIBRARIES AND PACKAGES

Google Colab comes pre-installed with many popular Python libraries and packages, including TensorFlow, PyTorch, Keras, Pandas, NumPy, and Matplotlib, among others. Additionally, users can install additional packages using pip.

CHAPTER 7

REAL-TIME ECG SIGNAL

7.1 INTRODUCTION

An ECG Sensor with disposable electrodes attaches directly to the chest to detect every heartbeat. The electrodes of the ECG sensor will convert heartbeats into electric signals. ECG Sensors are very lightweight, slim, and accurate in measuring continuous heartbeats and providing heart rate data. This device is always used by trained doctors and medical assistants.

The electrodes of the ECG Sensor have 3 pins and are connected by a cable that is 30 inches in length. This makes the ECG sensor easy to connect with a controller and can be placed at the waist or in a pocket. Additionally, the plug-in for the cable is a male sound plug, which makes it easy to remove or insert the cable into the amplifier board. The sensor is assembled on an arm pulse and a leg pulse. Each sensor electrode has methods for assembly in the body.

7.2 HARDWARE CIRCUIT

The connection of the AD8232 with Arduino is as follows:

- Arduino 3.3V is connected to the 3.3V pin of the AD8232.
- Arduino pin 10 is connected to the L0+ pin of the AD8232.
- Arduino pin 11 is connected to the L0- pin of the AD8232.
- Arduino Analog 1 (A1) is connected to the Output pin of the AD8232.
- Arduino Ground is connected to the Ground pin of the AD8232.

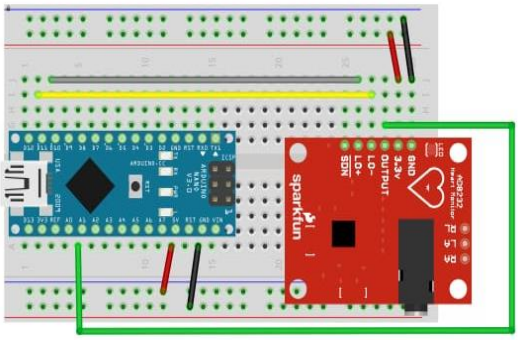

Figure 7.1 Schematic Diagram

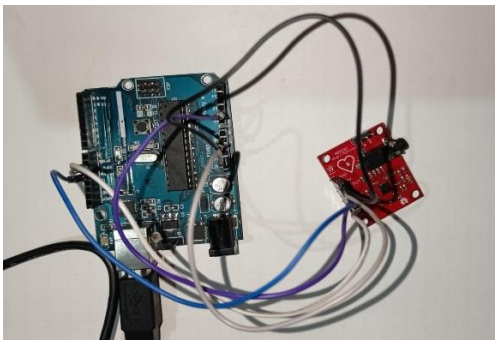

Figure 7.2 Hardware Setup

7.3 ECG SENSOR LEAD PLACEMENT

The electrode pad locations are shown in the figure.

- RA is connected to Input 1.
- LA is connected to Input 2.
- RL is connected to Input 3.

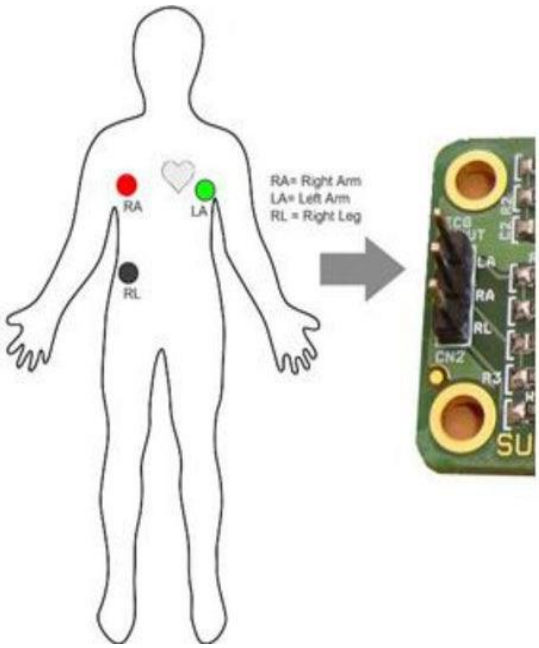

Figure 6.3 Electrode Placement

7.4 ECG SIGNAL EXTRACTION AS CSV

The Arduino code is uploaded from the Arduino IDE software to the Arduino UNO with the above-mentioned hardware setup. The ECG sensor readings are extracted in the Arduino IDE. The Real-Time ECG readings are converted into .csv file using CoolTerm software. CoolTerm is a simple serial port terminal application that is geared towards professionals with a need to exchange data with hardware connected to serial ports such as servo controllers, robotic kits, GPS receivers, microcontrollers.

CHAPTER 8

PERFORMANCE METRICS

8.1 INTRODUCTION

Performance metrics for classification models include accuracy (overall correctness), precision (accuracy of positive predictions), and recall (sensitivity to true positives), essential for evaluating model effectiveness in various applications.

8.2 PERFORMANCE METRICS

- Accuracy: The proportion of correctly classified instances out of the total instances.
- Precision: The proportion of true positive predictions out of all positive predictions.
- Recall (Sensitivity): The proportion of true positive predictions out of all actual positives.
- F1 Score: The harmonic mean of precision and recall, providing a balance between the two metrics.
- Specificity: The proportion of true negatives correctly identified by a model out of all actual negatives.
- Confusion Matrix: A table showing the counts of true positives, true negatives, false positives, and false negatives.

8.2.1 ACCURACY

The accuracy of a deep learning model is typically defined as the proportion of correctly predicted instances (or observations) out of the total

number of instances. It's calculated by dividing the number of correct predictions by the total number of predictions made by the model and multiplying by 100 to express it as a percentage.In classification tasks, accuracy measures the model's ability to correctly classify instances into their respective classes.

The accuracy of a DL model is calculated using the following formula:

$$\text{Accuracy} = \frac{\text{Total Number of Predictions}}{\text{Number of Correct Predictions}} \times 100$$

8.2.2 PRECISION

Precision is a fundamental evaluation metric used in classification tasks to assess the accuracy of a model's positive predictions. In a classification context, precision measures the proportion of true positive predictions (correctly identified instances of the positive class) out of all instances predicted as positive by the model.

Mathematically, precision is defined as:

$$\text{Precision} = \frac{\text{True Positives+False Positives}}{\text{True Positives}}$$

Components of precision:

True Positives (TP): The number of instances correctly classified as positive by the model.

False Positives (FP): The number of instances incorrectly classified as positive by the model (instances that are actually negative but predicted as positive).

Precision focuses on the accuracy of positive predictions and is particularly useful in scenarios where false positives are costly or undesirable. For example, in medical diagnosis, precision is crucial because it measures the model's ability to correctly identify patients with a particular condition without incorrectly labelling healthy individuals as positive cases.

A high precision score indicates that the model has a low rate of false positives, providing greater confidence in its positive predictions. Conversely, a lower precision score suggests that the model may be incorrectly classifying negative instances as positive, leading to false alarms or misdiagnoses.

Overall, precision is an essential metric for assessing the reliability and effectiveness of classification models, helping practitioners understand the model's performance in correctly identifying positive instances within a dataset.

8.2.3 RECALL (SENSITIVITY)

Recall, also known as sensitivity or true positive rate, is a key evaluation metric used in classification tasks to measure the ability of a model to correctly identify all positive instances in a dataset.

Mathematically, recall is defined as:

$$\text{Recall} = \frac{\text{True Positives}+\text{False Negatives}}{\text{True Positives}}$$

Components of recall:

True Positives (TP): The number of instances correctly classified as positive by the model.

False Negatives (FN): The number of instances that are actually positive but incorrectly classified as negative by the model.

Recall focuses on capturing as many positive instances as possible, minimizing the number of false negatives. It is particularly important in scenarios where missing positive instances (false negatives) can have severe consequences or where the cost of false negatives is high. For example, in medical diagnosis, recall measures the model's ability to correctly identify patients with a particular condition, ensuring that no positive cases are missed.

A high recall score indicates that the model has a low rate of false negatives, meaning it effectively captures the majority of positive instances in the dataset. Conversely, a lower recall score suggests that the model may be missing some positive instances, potentially leading to underdiagnosis or missed opportunities for intervention.

Overall, recall complements precision in providing a comprehensive understanding of a classification model's performance. While precision focuses on the accuracy of positive predictions, recall emphasizes the model's ability to detect all positive instances, providing insights into its sensitivity to the positive class.

8.2.4 F1 SCORE

The F1 score is a performance metric used to evaluate the accuracy of a classification model, particularly in scenarios where the classes are imbalanced. It is the harmonic mean of precision and recall, providing a balance between these two metrics.

The F1 score is calculated using the following formula:

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

It ranges between 0 and 1, where a higher F1 score indicates better model performance. The F1 score reaches its best value at 1 and worst value at 0. It is particularly useful when there is an uneven class distribution, as it takes both false positives and false negatives into account, providing a single score that reflects the overall performance of the model in terms of both precision and recall.

8.2.5 SPECIFICITY

Specificity is a statistical measure used in binary classification tasks to evaluate the performance of a model, particularly in medical diagnostics, but it's applicable in various other fields as well.

Specificity measures the proportion of actual negatives (class 0) that are correctly identified by the model. In medical terms, it refers to the ability of a diagnostic test to correctly identify the patients without a particular condition.

True Negative (TN): The model correctly predicts the negative class (e.g., absence of a disease) as negative.

False Positive (FP): The model incorrectly predicts the negative class as positive (e.g., predicts a disease when it's not present).

The formula for specificity is:

$$\text{Specificity} = \frac{\text{True Negative}}{\text{True Negative} + \text{False Positive}}$$

In medical diagnostics, a high specificity indicates that the model is good at correctly identifying patients who do not have the condition, minimizing the number of false alarms or unnecessary treatments. It complements sensitivity, which measures the ability of a test to correctly identify patients with the condition. Together, specificity and sensitivity provide a comprehensive evaluation of the performance of a diagnostic test.

8.2.6 CONFUSION MATRIX

A confusion matrix is a table that is often used to evaluate the performance of a classification model. It allows us to visualize the performance of a classification algorithm by presenting a summary of the correct and incorrect classifications made by the model on a specific dataset.

Components of a confusion matrix

True Positives (TP): The number of instances that were correctly predicted as positive by the model.

True Negatives (TN): The number of instances that were correctly predicted as negative by the model.

False Positives (FP): The number of instances that were incorrectly predicted as positive by the model (i.e., the model predicted positive, but the actual label was negative).

False Negatives (FN): The number of instances that were incorrectly predicted as negative by the model (i.e., the model predicted negative, but the actual label was positive).

Table 8.1 Confusion Matrix in Classification Models

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

From the confusion matrix, various performance metrics can be derived, such as accuracy, precision, recall, F1 score, and specificity. It provides insights into the model's strengths and weaknesses, particularly in terms of its ability to correctly classify instances belonging to different classes.

Overall, the confusion matrix is a valuable tool for understanding the classification performance of a model and identifying areas for improvement.

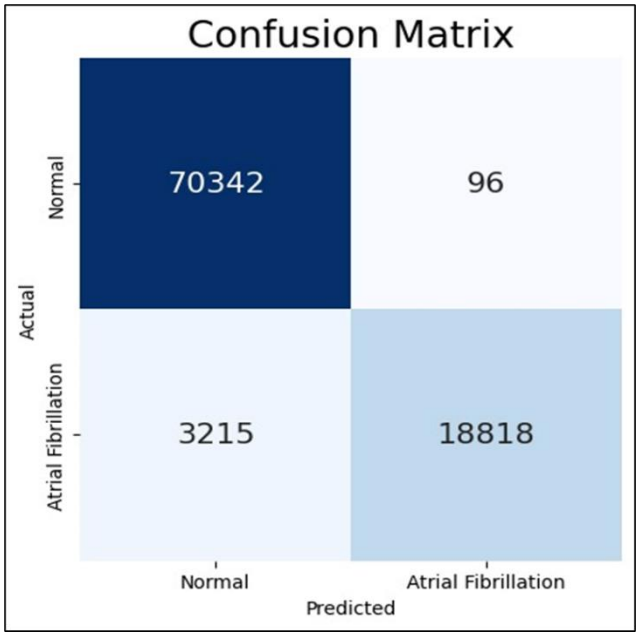

Figure 8.1 Confusion matrix of proposed model

8.3 SUMMARY

Performance metrics are essential tools for evaluating the effectiveness of machine learning models. They help assess how well a model is performing on a given task and provide insights into its strengths and weaknesses.

CHAPTER 9

RESULTS AND DISCUSSION

9.1 INTRODUCTION

This chapter discusses the results obtained from the Deep Learning model implemented using U-Net. The results are obtained from the Google Colaboratory online platform with the data files loaded onto the Google Drive. The R- Peak annotation, Predicted AF signal and the plotted graphs are shown here.

9.2 R-PEAK DETECTION

We used the MIT-BIH AF database from PhysioNet to test the effectiveness of the proposed CNN based detection of AF. This database contains twenty-five ECG records of ten hours each and an ECG signal was sampled at 250 samples per second. All R-peaks are annotated in this database which is shown in the Fig 9.1.

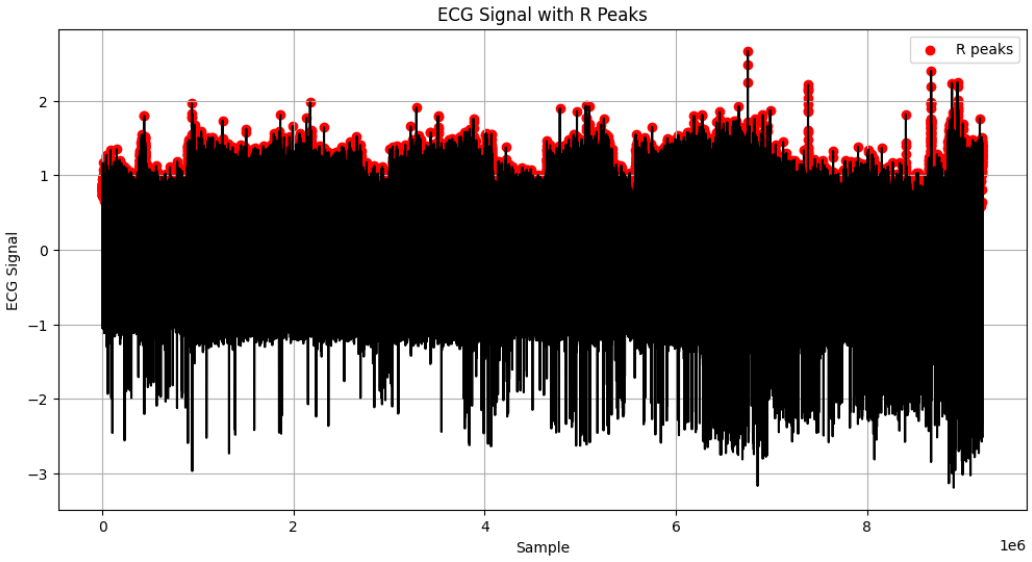

Figure 9.1 R-peak detection

9.3 AF PREDICTION IN LONG-TERM AF DATASET

The long-term AF dataset is pre-processed and given to the trained U-Net model to detect if the patient is AF Detected or not. It displays the signal and detects the peak. If it is AF detected it labels the sample at which it is detected with the blue line. A dialogue box displays the result if AF is detected in red colour and if not, AF is not detected in green colour.

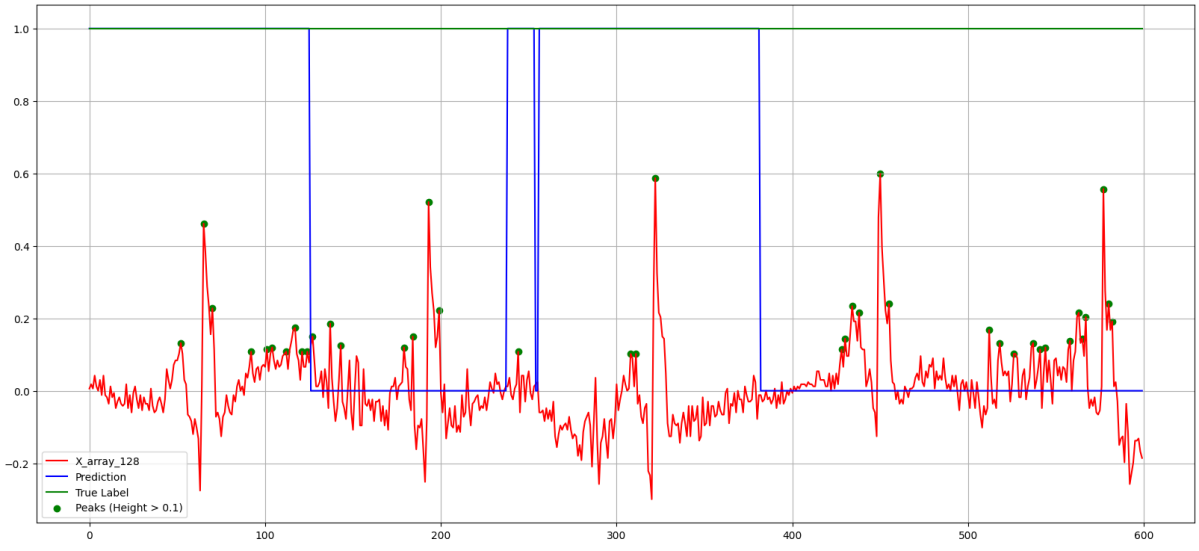

Figure 9.2 AF Detection in Long-term AF Dataset

Figure 9.3 Display of AF Detection in Long-term AF Dataset

9.4 AF PREDICTION IN NORMAL SINUS RHYTHM DATASET

The normal sinus rhythm dataset is pre-processed and given to the trained U-Net model to detect if the patient is AF Detected or not. It displays the signal and detects the peak. If it is AF detected it labels the sample at which it is detected

with the blue line. A dialogue box displays the result if AF is detected in red colour and if not AF is not detected in green colour.

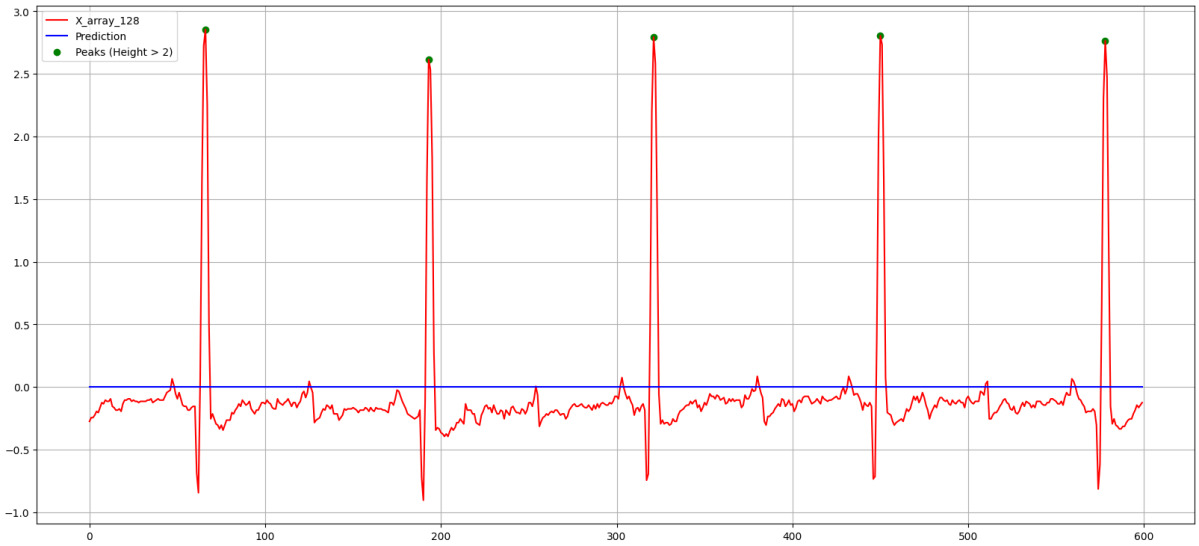

Figure 9.4 AF Detection in Normal Sinus Rhythm Dataset

Figure 9.5 Display of AF Detection in Normal Sinus Rhythm Dataset

9.5 PERFORMANCE ANALYSIS AND CONFUSION MATRIX

In Fig 9.6, the plot illustrates the Training Vs Validation accuracy. The x-axis represents the dataset, while the y-axis denotes accuracy. The red line corresponds to the training dataset, and the blue line represents the validation dataset. Fig 9.7 represents the Training Vs Loss Plot.

Figure 9.6 Training vs Validation Accuracy

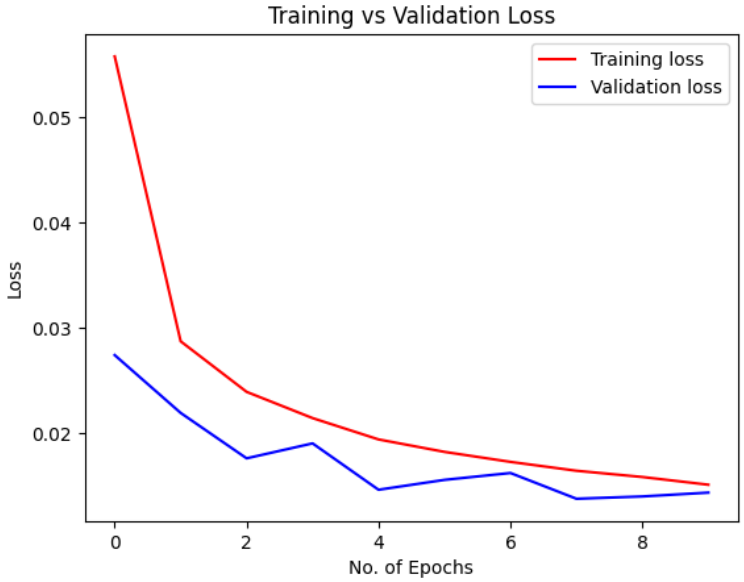

Figure 9.7 Training vs Validation Loss

Utilizing the suggested approach, the model underwent training and testing procedures using the MIT-BIH database. Subsequently, the following outcomes were derived from the evaluation process and Confusion Matrix is obtained in Fig 9.8.

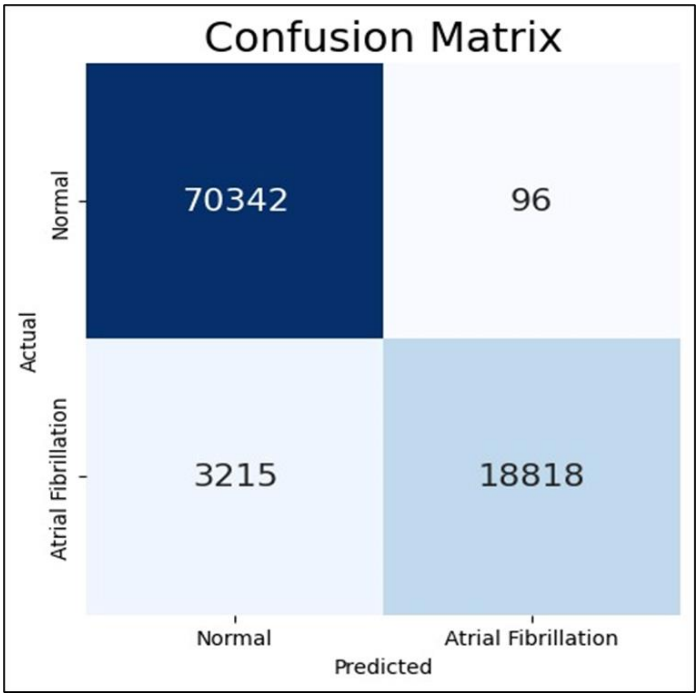

Figure 9.8 Confusion Matrix

Table 9.1 Performance Metrics of Proposed model

Performance Metrics	**Proposed Model**
Accuracy	96.5%
Precision	0.966
Recall	0.965
F1 Score	0.964
Specificity	0.996

Table 9.2 Comparison with other methods

Performance Metrics	Existing Methods				Proposed Method
	CNN/LSTM	**SVM, RF**	**1D-CNN**	**DDCNN**	**CNN/U-Net model**
Accuracy	97.9%	97.45%	87.98%	91%	96.89 %
Precision	-	-	-	-	97%
Recall/Sensitivity	97.8%	98.99%	-	78%	96.9%
F1 Score	-	-	0.84	0.83	0.969
Specificity	0.97	0.95	-	0.901	0.996

9.6 REAL-TIME ECG SIGNAL

The ECG signal is sensed and extracted in real-time using Arduino UNO and ECG sensor. The extracted signal is plotted using Arduino IDE and CoolTerm.

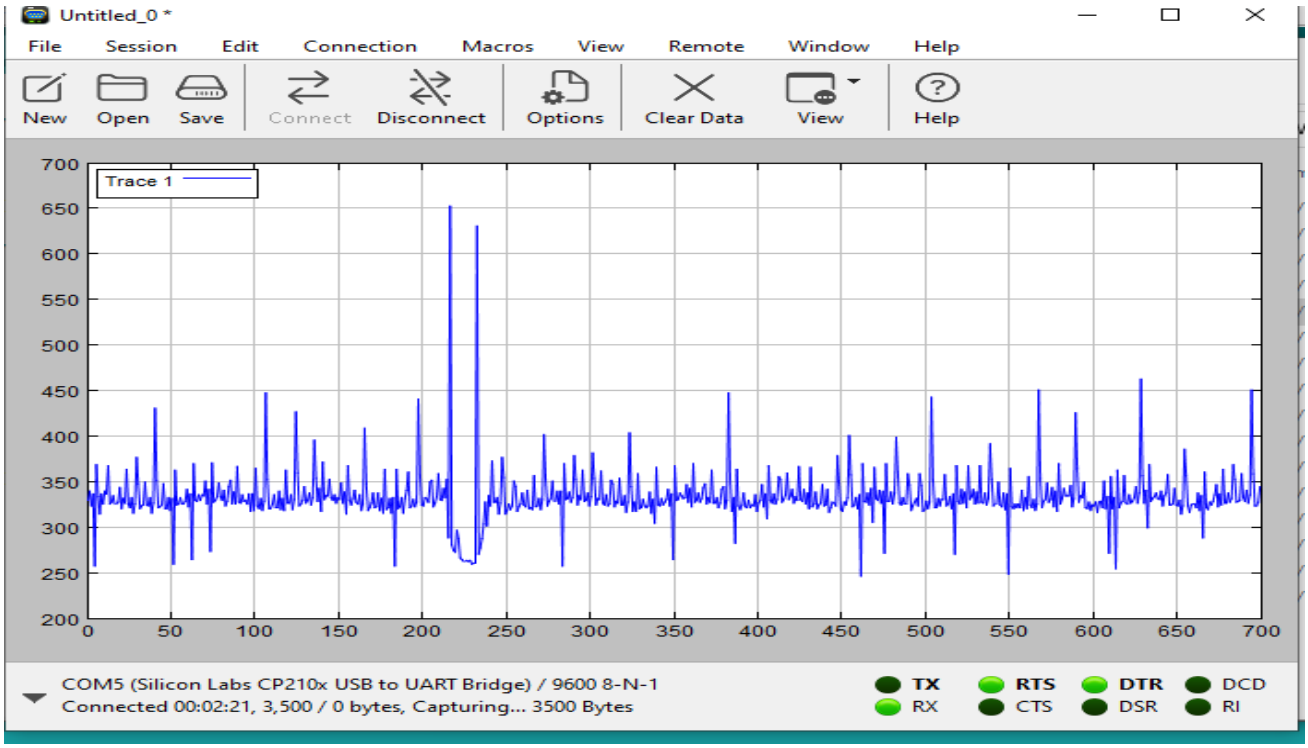

Figure 9.9 Real-time ECG Signal

9.7 SUMMARY

The results shown here are the prediction of the presence of Atrial Fibrillation. The datasets are taken from physionet for both containing Normal Sinus Rhythm and Long-term Atrial Fibrillation. As an improvement in the project, the real-time signals are taken for Atrial Fibrillation detection. The file format of the real-time data is not supported by the developed ML model.

CHAPTER 10

CONCLUSION

In this project, the focus lies on leveraging a CNN classifier trained using a U-Net architecture to detect Atrial Fibrillation (AF) and Non-Atrial Fibrillation (NAF) signals efficiently. The U-Net model, designed with a reduced number of layers, aims to streamline computation time while ensuring precise prediction outcomes. The motivation behind this endeavor stems from the pressing need to identify AF early, thereby mitigating the associated risks such as stroke, heart failure, and even death.

By harnessing the power of deep learning, particularly through Convolutional Neural Networks (CNNs) and the specialized U-Net architecture, this project endeavors to provide a solution that significantly expedites the process of AF prediction compared to manual methods. With manual prediction often consuming a considerable amount of time up to half an hour per patient, the proposed automated approach holds the promise of revolutionizing AF detection in clinical settings.

The model's impressive accuracy of 96.89% shows it's really good at spotting Atrial Fibrillation (AF) from other heart signals. Its precision of 0.97 means it rarely makes mistakes, and its F1 score and recall rate of 0.96 show it's strong in both accuracy and completeness.

This model isn't just about working faster; it's about improving patient care. By helping doctors quickly and accurately identify AF, it could save lives. This shows how AI can revolutionize healthcare by making it more precise and efficient for better patient outcomes.

APPENDIX

SOURCE CODE:

Code for Model Training

```
!pip install wfdb

import pandas as pd

import numpy as np

import wfdb

import random

import seaborn as sns

import tensorflow as tf

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras import layers, losses

from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D,
BatchNormalization,concatenate,Conv1DTranspose,Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import Activation

from tensorflow.keras.losses import binary_crossentropy

from tensorflow.keras.models import Model
```

```
from tensorflow.keras import layers
from tensorflow.keras.models import Model
from wfdb import processing
random.seed(42)
import warnings
warnings.filterwarnings("ignore")
from google.colab import drive
drive.mount('/content/drive')

data = '/content/drive/My Drive/Final/atrial-fib/files/'
def load_ecg(file,filetype):
    record = wfdb.rdrecord(file)
    annotation = wfdb.rdann(file, filetype)
    p_signal = record.p_signal
    atr_sym = annotation.symbol
    atr_sample = annotation.sample

    return p_signal, atr_sym, atr_sample
def af_data(qrs_sample,atr_sample):
    AF_samples_array = []
    NonAF_samples_array = []
    for i in range (len(atr_sample)):
```

```
        if  (i % 2 == 0) & (i < len(atr_sample)-1) :
            NonAF_samples = qrs_sample[( qrs_sample >= atr_sample[i] ) &
(qrs_sample < atr_sample[i+1])  ]
            NonAF_samples_array.append(NonAF_samples)
        if  (i % 2 == 1) & (i < len(atr_sample)-1) :
            AF_samples = qrs_sample[(qrs_sample >= atr_sample[i] ) &
(qrs_sample < atr_sample[i+1])  ]
            AF_samples_array.append(AF_samples)
        if  (i % 2 == 0)  & (i == len(atr_sample)-1) :
            NonAF_samples = qrs_sample [ qrs_sample > atr_sample[i] ]
            NonAF_samples_array.append(NonAF_samples)
        if  (i % 2 == 1)  & (i == len(atr_sample)-1) :
            AF_samples = qrs_sample [ qrs_sample > atr_sample[i] ]
            AF_samples_array.append(AF_samples)


    return
np.array(AF_samples_array,dtype='object'),np.array(NonAF_samples_array,dty
pe='object')

patients=['04015','04048','04126','04746','04908','04936','05091','05261','06453','
06995']

y_128 = []

X_array_128 = []

for pt in patients:
```

```
    file = data + pt
    p_signal, atr_sym, atr_sample = load_ecg(file,'atr')
    p_signal, qrs_sym, qrs_sample = load_ecg(file,'qrs')
    AF_samples_array, NonAF_samples_array = af_data(qrs_sample,atr_sample)
    if not (pt == '07162' or pt == '07859'):
        AF_samples_array = np.concatenate(AF_samples_array,axis=0)
        NonAF_samples_array = np.concatenate(NonAF_samples_array,axis=0)
    totalsamples =
np.concatenate((AF_samples_array,NonAF_samples_array),axis=0)
    p_signal = p_signal[:,0]
    for i in range (1,np.size(totalsamples)-3,1):
        if totalsamples[i]>= 750:
            if (totalsamples[i] in AF_samples_array):
                X  = p_signal[(totalsamples[i]-64):(totalsamples[i]+64)]
                X_array_128.append(X)
                y_128.append(np.ones((128,)))
        if totalsamples[i]>= 750:
            if (totalsamples[i] in NonAF_samples_array):
                X  = p_signal[(totalsamples[i]-64):(totalsamples[i]+64)]
                X_array_128.append(X)
                y_128.append(np.zeros((128,)))
p_signal.shape
```

```
print(np.array(X_array_128).shape)

print(np.array(y_128).shape)

plt.figure(figsize=(20, 9))

plt.grid()

plt.plot(X_array_128[50] ,color='r')

plt.plot(y_128[50],color='b')

plt.show()

X_array_128 =
np.array(X_array_128).reshape((np.array(X_array_128).shape[0],128,1))

y_128 = np.array(y_128).reshape((np.array(y_128).shape[0],128,1))

X_train, X_valid, y_train, y_valid = train_test_split(X_array_128, y_128,
test_size=0.4, random_state=42)

X_valid, X_test, y_valid, y_test = train_test_split(X_valid, y_valid,
test_size=0.5, random_state=42)


print(X_train.shape)

print(X_valid.shape)

print(X_test.shape)


print(y_train.shape)

print(y_valid.shape)

print(y_test.shape)
```

```
def build_unet_model():

    inputs = tf.keras.layers.Input((128, 1))

    c1 = tf.keras.layers.Conv1D(16, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(inputs)
    c1 = tf.keras.layers.Dropout(0.1)(c1)
    c1 = tf.keras.layers.Conv1D(16, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c1)
    p1 = tf.keras.layers.MaxPooling1D(pool_size=(2))(c1)

    c2 = tf.keras.layers.Conv1D(32, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(p1)
    c2 = tf.keras.layers.Dropout(0.1)(c2)
    c2 = tf.keras.layers.Conv1D(32, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c2)
    p2 = tf.keras.layers.MaxPooling1D(pool_size=(2))(c2)

    c3 = tf.keras.layers.Conv1D(64, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(p2)
    c3 = tf.keras.layers.Dropout(0.2)(c3)
    c3 = tf.keras.layers.Conv1D(64, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c3)
```

```
    p3 = tf.keras.layers.MaxPooling1D(pool_size=(2))(c3)


    c4 = tf.keras.layers.Conv1D(128, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(p3)
    c4 = tf.keras.layers.Dropout(0.2)(c4)
    c4 = tf.keras.layers.Conv1D(128, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c4)
    p4 = tf.keras.layers.MaxPooling1D(pool_size=(2))(c4)
    c5 = tf.keras.layers.Conv1D(128, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(p4)
    c5 = tf.keras.layers.Dropout(0.3)(c5)
    c5 = tf.keras.layers.Conv1D(128, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c5)


    u6 = tf.keras.layers.Conv1DTranspose(128, (2), strides=(2),
padding='same')(c5)
    u6 = tf.keras.layers.concatenate([u6, c4])
    c6 = tf.keras.layers.Conv1D(128, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(u6)
    c6 = tf.keras.layers.Dropout(0.2)(c6)
    c6 = tf.keras.layers.Conv1D(128, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c6)
```

```
    u7 = tf.keras.layers.Conv1DTranspose(64, (2), strides=(2),
padding='same')(c6)

    u7 = tf.keras.layers.concatenate([u7, c3])

    c7 = tf.keras.layers.Conv1D(64, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(u7)

    c7 = tf.keras.layers.Dropout(0.2)(c7)

    c7 = tf.keras.layers.Conv1D(64, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c7)


    u8 = tf.keras.layers.Conv1DTranspose(32, (2), strides=(2),
padding='same')(c7)

    u8 = tf.keras.layers.concatenate([u8, c2])

    c8 = tf.keras.layers.Conv1D(32, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(u8)

    c8 = tf.keras.layers.Dropout(0.1)(c8)

    c8 = tf.keras.layers.Conv1D(32, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c8)

    u9 = tf.keras.layers.Conv1DTranspose(16, (2), strides=(2),
padding='same')(c8)

    u9 = tf.keras.layers.concatenate([u9, c1])

    c9 = tf.keras.layers.Conv1D(16, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(u9)

    c9 = tf.keras.layers.Dropout(0.1)(c9)
```

```
    c9 = tf.keras.layers.Conv1D(16, (3), activation='relu',
kernel_initializer='he_normal', padding='same')(c9)
    outputs = tf.keras.layers.Conv1D(1, (1), activation='sigmoid')(c9)
    unet_model = tf.keras.Model(inputs, outputs, name="U-Net")


    return unet_model
unet_model = build_unet_model()
unet_model.summary()
unet_model.compile(loss = 'mean_squared_error',optimizer =
'adam',metrics='binary_accuracy')
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
history = unet_model.fit(X_train, y_train,
          epochs=10,
          batch_size=64,
          validation_data=(X_valid,y_valid),
          shuffle=True,verbose=1,
          callbacks=[callback])
unet_model.save("atrialfib.h5")
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

folder_id = None
file_list = drive.ListFile({'q': "'root' in parents and trashed=false"}).GetList()
for file in file_list:
    if file['title'] == 'unet_model' and file['mimeType'] ==
'application/vnd.google-apps.folder':
        folder_id = file['id']
        break

if folder_id:
    model_file = drive.CreateFile({'title': 'atrialfib.h5', 'parents': [{'id':
folder_id}]})
    model_file.SetContentFile('atrialfib.h5')
    model_file.Upload()
    print("Model saved to 'unet_model' folder in Google Drive.")
else:
```

```
    print("Folder 'unet_model' not found in Google Drive.")
plt.plot(history.history['binary_accuracy'], 'r', label='Training accuracy')
plt.plot(history.history['val_binary_accuracy'], 'b', label='Training accuracy')


plt.title('Training Vs Validation Accuracy')
plt.xlabel('No. of Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
plt.plot(history.history['loss'], 'r', label='Training loss')
plt.plot(history.history['val_loss'], 'b', label='Validation loss')


plt.title('Training vs Validation Loss')
plt.xlabel('No. of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
pred = unet_model.predict(X_test)
pred = (pred > 0.5).astype(np.uint8).astype(int)
preds = []
test = []
for i in pred:
```

```
    if 0 in i:
        preds.append(np.zeros((1)))
    else:
        preds.append(np.ones((1)))

for i in y_test:
    if 0 in i:
        test.append(np.zeros((1)))
    else:
        test.append(np.ones((1)))
TN = sum(1 for pred, actual in zip(preds, test) if pred == 0 and actual == 0)
FP = sum(1 for pred, actual in zip(preds, test) if pred == 1 and actual == 0)

specificity = TN / (TN + FP)

print("Specificity:", specificity)
from sklearn import metrics
from sklearn.metrics import confusion_matrix
CLASS_LABELS = ['Normal', 'Atrial Fibrillation']
cm_data = confusion_matrix(test, preds)
cm = pd.DataFrame(cm_data, columns=CLASS_LABELS, index =
CLASS_LABELS)
```

```
cm.index.name = 'Actual'

cm.columns.name = 'Predicted'

plt.figure(figsize = (5,5))

plt.title('Confusion Matrix', fontsize = 20)

sns.set(font_scale=1.2)

ax = sns.heatmap(cm, cbar=False, cmap="Blues", annot=True,
annot_kws={"size": 16}, fmt='g')

print('Accuracy:', np.round(metrics.accuracy_score(test, preds),3)*100,'%')

print('Precision:', np.round(metrics.precision_score(test, preds,
average='weighted'),3))

print('Recall:', np.round(metrics.recall_score(test, preds, average='weighted'),3))

print('F1 Score:', np.round(metrics.f1_score(test, preds, average='weighted'),3))
```

Code for Detection

```
!pip install wfdb

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

import tensorflow.keras

import pandas as pd

import numpy as np
```

```
import wfdb
import random
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from wfdb import processing
random.seed(42)
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import find_peaks
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from google.colab import drive
drive.mount('/content/drive')
data = '/content/drive/My Drive/pred/files/'
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
```

```
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
file_id = '1bDGZ75f5HcPR-382TtxZ0MloE8P5GbQZ'
import os
downloaded_file = drive.CreateFile({'id': file_id})
downloaded_file.GetContentFile('atrialfib.h5')
from keras.models import load_model
model = load_model('atrialfib.h5')
def load_ecg(file,filetype):
    record = wfdb.rdrecord(file)
    annotation = wfdb.rdann(file, filetype)
    p_signal = record.p_signal
    atr_sym = annotation.symbol
    atr_sample = annotation.sample
    return p_signal, atr_sym, atr_sample
def af_data(qrs_sample,atr_sample):
    AF_samples_array = []
```

```
    NonAF_samples_array = []
    for i in range (len(atr_sample)):
        if  (i % 2 == 0) & (i < len(atr_sample)-1) :
            NonAF_samples = qrs_sample[( qrs_sample >= atr_sample[i] ) &
(qrs_sample < atr_sample[i+1])  ]
            NonAF_samples_array.append(NonAF_samples)
        if  (i % 2 == 1) & (i < len(atr_sample)-1) :
            AF_samples = qrs_sample[(qrs_sample >= atr_sample[i] ) &
(qrs_sample < atr_sample[i+1])  ]
            AF_samples_array.append(AF_samples)
        if  (i % 2 == 0)  & (i == len(atr_sample)-1) :
            NonAF_samples = qrs_sample [ qrs_sample > atr_sample[i] ]
            NonAF_samples_array.append(NonAF_samples)
        if  (i % 2 == 1)  & (i == len(atr_sample)-1) :
            AF_samples = qrs_sample [ qrs_sample > atr_sample[i] ]
            AF_samples_array.append(AF_samples)
  return
np.array(AF_samples_array,dtype='object'),np.array(NonAF_samples_array,dty
pe='object')
patients = ['00']
y_128 = []
X_array_128 = []
for pt in patients:
```

```
    file = data + pt
    p_signal, atr_sym, atr_sample = load_ecg(file, 'atr')
    p_signal, qrs_sym, qrs_sample = load_ecg(file, 'qrs')
    AF_samples_array, NonAF_samples_array = af_data(qrs_sample,
atr_sample)
    AF_samples_array = np.concatenate(AF_samples_array, axis=0)
    NonAF_samples_array = np.concatenate(NonAF_samples_array, axis=0)
    totalsamples = AF_samples_array
    p_signal = p_signal[:, 0]
    for i in range(1, np.size(totalsamples) - 3, 1):
        if totalsamples[i] in AF_samples_array:
            X = p_signal[(totalsamples[i] - 64):(totalsamples[i] + 64)]
            X_array_128.append(X)
            y_128.append(np.ones((128,)))
        if totalsamples[i] in NonAF_samples_array:
            X = p_signal[(totalsamples[i] - 64):(totalsamples[i] + 64)]
            X_array_128.append(X)
            y_128.append(np.zeros((128,)))
    X_array_128 =
np.array(X_array_128).reshape((np.array(X_array_128).shape[0], 128, 1))
    y_128 = np.array(y_128).reshape((np.array(y_128).shape[0], 128, 1))
    pred = model.predict(X_array_128)
```

```
    pred = (pred > 0.5).astype(np.uint8).astype(int)
    plt.figure(figsize=(20, 9))
    plt.grid()
    plt.plot(X_array_128.reshape(-1)[:600], color='r', label='X_array_128')
    plt.plot(pred.reshape(-1)[:600], color='b', label='Prediction')
    plt.plot(y_128.reshape(-1)[:600], color='g', label='True Label')
    peaks, _ = find_peaks(X_array_128.reshape(-1)[:600], height=0.1)
    plt.scatter(peaks, X_array_128.reshape(-1)[:600][peaks], color='green',
label='Peaks (Height > 0.1)')
    plt.legend()
    plt.show()
def is_array_of_zeros(arr):
  return np.all(arr == 1)
from IPython.display import HTML, display
html_code1 = """
<div class="alert">
  <p>AF DETECTED.</p>
</div>
<style>
.alert {
  padding: 20px;
  align= center;
```

```
  background-color: #f44336;
  color: white;
  margin-bottom: 15px;
}
</style>
"""
html_code2 = """
<div class="alert">
  <p>AF NOT DETECTED.</p>
</div>
<style>
.alert {
  padding: 20px;
  align= center;
  background-color: #48D426;
  color: white;
  margin-bottom: 15px;
}
</style>
"""
if is_array_of_zeros(y_128):
    display(HTML(html_code1))
```

```
else:
    display(HTML(html_code2))
```

Code for Extracting Real-Time Signal

```
#include <ThingSpeak.h>
#include <WiFi.h>
const char *ssid = "Wifi Name"; // Our Wifi name is to be given
const char *password = "Password";
const char *thingSpeakAddress = "api.thingspeak.com";
const String apiKey = "O2JUIN8J6CPPX1AW";
void setup() {
  Serial.begin(9600);
  pinMode(10, INPUT);
  pinMode(11, INPUT);
  connectToWiFi();
}
void loop() {
  if ((digitalRead(10) == 1) || (digitalRead(11) == 1)) {
    Serial.println("!");
  } else {
    int ecgValue = analogRead(A1);
    Serial.println(ecgValue);
```

```
    uploadToThingSpeak(ecgValue);
  }
  delay(100);
}
void connectToWiFi() {
  Serial.print("Connecting to Wi-Fi");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");
}
void uploadToThingSpeak(int value) {
  WiFiClient client;
  if (client.connect(thingSpeakAddress, 80)) {
    String postStr = apiKey;
    postStr += "&field1=";
    postStr += String(value);
    postStr += "\r\n\r\n";
    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
```

```
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(postStr.length());
    client.print("\n\n");
    client.print(postStr);
  }
  client.stop();
}
```

REFERENCE

[1] A. Rizwan et al., "A Review on the State of the Art in Atrial Fibrillation Detection Enabled by Machine Learning," in IEEE Reviews in Biomedical Engineering, vol. 14, pp. 219-239, 2021, doi: 10.1109/RBME.2020.2976507.

[2] A. Vadillo-Valderrama, R. Goya-Esteban, R. P. Caulier-Cisterna, A. García-Alberola and J. L. Rojo-Álvarez, "Differential Beat Accuracy for ECG Family Classification Using Machine Learning," in IEEE Access, vol. 10, pp. 129362-129381, 2022, doi: 10.1109/ACCESS.2022.3227219.

[3] DDCNN: A Deep Learning Model for AF Detection From a Single-Lead Short ECG Signal, Zhaocheng Yu, Junxin Chen, yu Iu, Yongypong Chen, Tingting Wang, Robert Nowak, Zhihan Lv, October 2022.

[4] E. Prabhakararao and S. Dandapat, "Atrial Fibrillation Burden Estimation Using Multi-Task Deep Convolutional Neural Network," in IEEE Journal of Biomedical and Health Informatics, vol. 26, no. 12, pp. 5992-6002, Dec. 2022, doi: 10.1109/JBHI.2022.3191682.

[5] K. Gupta, V. Bajaj and I. A. Ansari, "Atrial Fibrillation Detection Using Electrocardiogram Signal Input to LMD and Ensemble Classifier," in IEEE Sensors Letters, vol. 7, no. 6, pp. 1-4, June 2023, Art no. 7002904, doi: 10.1109/LSENS.2023.3281129.

[6] P. -Y. Hsu and C. -K. Cheng, "Arrhythmia Classification using Deep Learning and Machine Learning with Features Extracted from Waveform-based Signal Processing," 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), Montreal, QC, Canada, 2020, pp. 292-295, doi: 10.1109/EMBC44109.2020.9176679.

[7] P. Zhang, Y. Chen, F. Lin, S. Wu, X. Yang and Q. Li, "Semi-Supervised Learning for Automatic Atrial Fibrillation Detection in 24-Hour Holter

Monitoring," in IEEE Journal of Biomedical and Health Informatics, vol. 26, no. 8, pp. 3791-3801, Aug. 2022, doi: 10.1109/JBHI.2022.3173655.

[8] S. J. M. Yazdi, H. -J. Park, C. -S. Son and J. -H. Lee, "A Novel Machine Learning Approach to Classify and Detect Atrial Fibrillation Using Optimized Implantable Electrocardiogram Sensor," in IEEE Access, vol. 9, pp. 149250-149265, 2021, doi: 10.1109/ACCESS.2021.3123367.

[9] S. K. Bashar et al., "Novel Density Poincaré Plot Based Machine Learning Method to Detect Atrial Fibrillation From Premature Atrial/Ventricular Contractions," in IEEE Transactions on Biomedical Engineering, vol. 68, no. 2, pp. 448-460, Feb. 2021, doi: 10.1109/TBME.2020.3004310.

[10] S. M S, V. K. K, S. J R, S. K. Patnasetty, S. Bajpai and S. Ojha, "Classification of Cardiac Arrhythmia Using Machine Learning Algorithms," 2023 International Conference on Advances in Electronics, Communication, Computing and Intelligent Information Systems (ICAECIS), Bangalore, India, 2023, pp. 390-394, doi: 10.1109/ICAECIS58353.2023.10170082.

[11] X. Zhang, M. Jiang, K. Polat, A. Alhudhaif, J. Hemanth and W. Wu, "Detection of Atrial Fibrillation From Variable-Duration ECG Signal Based on Time-Adaptive Densely Network and Feature Enhancement Strategy," in IEEE Journal of Biomedical and Health Informatics, vol. 27, no. 2, pp. 944-955, Feb. 2023, doi: 10.1109/JBHI.2022.3221464.