VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI – 590 018

A PROJECT REPORT ON

"Detection of Cardiac Arrhythmia Using Machine Learning"

Submitted in partial fulfillment of the requirements of the Award of degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE & ENGINEERING

Submitted By:

AVANI H S	**4VV16CS016**
MADHU S	**4VV16CS051**
MANOJ ATHREYA A	**4VV16CS055**
POOJA	**4VV14CS064**

UNDER THE GUIDANCE OF

Dr. K Paramesha
Professor
Department of Computer Science & Engineering
VVCE, Mysuru

2019-2020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
VIDYAVARDHAKA COLLEGE OF ENGINEERING
MYSURU – 570 002

Vidyavardhaka College of Engineering
Gokulam III Stage, Mysuru-570 002

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project report entitled **"Detection of Cardiac Arrhythmia Using Machine Learning"** is a bona fide work carried out by **Avani H S** (4VV16CS016), **Madhu S** (4VV16CS051), **Manoj Athreya A** (4VV16CS055), **Pooja** (4VV14CS064) students of VIII semester Computer Science and Engineering, **Vidyavardhaka College of Engineering, Mysuru** in partial fulfillment for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering** of the **Visvesvaraya Technological University, Belagavi**, during the academic year **2019-2020**. It is certified that all the suggestions and corrections indicated for the internal assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the requirements in respect of project work prescribed for the said degree.

Signature of the Guide	**Signature of the HOD**	**Signature of the Principal**
(Dr. K Paramesha)	**(Dr. Ravi Kumar V)**	**(Dr. B Sadashive Gowda)**

Name of the Examiners **Signature with Date**

1)

2)

ACKNOWLEDGEMENT

If words are considered as the tokens of acknowledges, then the words play the heralding role of expressing our gratitude.

With proud gratitude we thank God Almighty for all the blessings showered on us and for completing our project successfully.

We owe our gratitude to The Principal, **Dr. B Sadashive Gowda** for his wholehearted support and for his kind permission to undergo the project.

We wish to express our deepest gratitude to **Dr. Ravi Kumar V,** Head of Department, Computer Science and Engineering, VVCE, for his profound alacrity in our project and his valuable suggestions.

We wish to enunciate our special thanks to our paradigmatic and relevant Project coordinators **Janhavi V** Associate Professor, **Usha C S** Assistant Professor, **Nithin Kumar** Assistant Professor, **Nagashree Nagaraj** Assistant Professor and internal guide **Dr. K Paramesha**, Professor in Computer Science and Engineering, VVCE who gave us throughout our project period with their valuable suggestions and for devoting their precious time in making this project a success.

In the end, we are anxious to offer our sincere thanks to our family members and friends for their valuable suggestions, encouragement and unwavering support.

Avani H S (4VV16CS016)

Madhu S (4VV16CS051)

Manoj Athreya A (4VV16CS055)

Pooja (4VV14CS064)

ABSTRACT

Cardiac Arrhythmia is a condition where a person suffers from an abnormal heart rhythm. It is due to the malfunction in the electrical impulses within the heart that coordinate how it beats. As a result, the heart beats too fast or too slow. The rhythm of the heart is controlled by a node at the top of the heart, called the sinus node, which triggers an electrical signal that travels through the heart causing it to beat and pump blood around the body. Excess electrical activity in the top or bottom of the heart means that the heart does not pump efficiently. The most common symptoms of Arrhythmia include shortness of breath, fainting, an unexpected loss of heart function and unconsciousness that leads to death within minutes unless the person receives emergency medical treatment to restart the heart. So, it's vital to know about and understand the condition, what danger signs to look out for and how to diagnose it early. To diagnose Cardiac Arrhythmia early, doctors need to carefully evaluate heartbeats from different locations of the body accurately. Reviewing these fundamental heart sounds (FHSs) for every patient manually is very time consuming for medicines. A potential solution to this is to provide automated diagnosis through the use of machine learning models. Hence classification of heart sound recordings using Machine Learning techniques could help overcome this problem. The aim of this project is to build a convolution neural network that differentiates between normal and abnormal heart sounds.

CONTENTS

CHAPTERS **Page No.**

LIST OF FIGURES

FIGURES **Page No.**

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Cardiac Arrhythmia is a condition where a person suffers from an abnormal heart rhythm. It is due to the malfunction in the electrical impulses within the heart that coordinate how it beats. As a result, the heart beats too fast or too slow. The rhythm of the heart is controlled by a node at the top of the heart, called the sinus node, which triggers an electrical signal that travels through the heart causing the heart to beat, pumping blood around the body. Excess electrical activity in the top or bottom of the heart means that the heart doesn't pump efficiently. [1]

The most common symptoms of Arrhythmia include shortness of breath, fainting, an unexpected loss of heart function and unconsciousness that leads to death within minutes unless the person receives emergency medical treatment to restart the heart.[2] So, it's vital to know about and understand the condition, what danger signs to look out for and how to diagnose it early. To diagnose Cardiac Arrhythmia early, doctors need to carefully evaluate heartbeats from different locations of the body accurately.

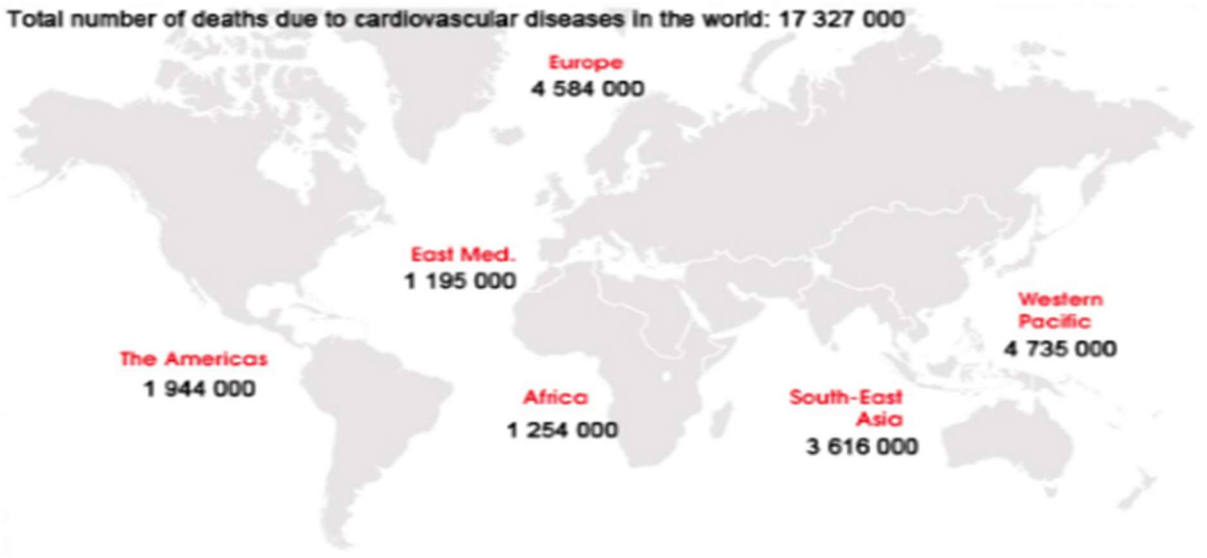

Figure 1.1: Total number of death cases found around the world

The above figure gives the illustration of the number of arrhythmia related cases found all over the world and their data set of heart related disease which can be used to differentiate between the kind of arrhythmia and classify the results between the normal and abnormal heartbeat. As of the report it shows 17,327,000 cases reported all over the world till 2019. With Western Pacific and European nations having a greater number of cases reported.[3]

1.1 SCOPE OF WORK

Health is very important to each and every person in this world. Reviewing these fundamental heart sounds (FHSs) for every patient manually is very time consuming for doctors. A potential solution to this is to provide automated diagnosis using machine learning models. Hence classification of heart sound recordings using Machine Learning techniques could help overcome this problem.[16]

This report illustrates the heart sound classification process using machine learning techniques. It starts with an introduction, about the datasets, followed by an exploratory data analysis that shows some summary statistics about the data sets, and finally the best prediction algorithm.

1.2 MOTIVATION

A disease is a big reason for morbidity and mortality in the current living style. Identification of such disease is an important but a complex task that needs to be performed very minutely, efficiently and the correct automation would be desirable. Every human being cannot be equally skillful and so as doctors. All the doctors cannot be equally skilled in all the specialties and at many remote places we don't have skilled and specialist doctors available easily. An automated system in medical diagnosis would enhance medical care and it can also reduce costs as well as time in the process of diagnoses. In this study, we have designed a model that can efficiently discover the normal and abnormal heartbeats to predict the risk levels of patients based on the given parameters of the symptoms of some cardiac arrhythmia. The performance of the system will be evaluated in terms of classification accuracy and the results will show that the system has the great potential in predicting the disease risk level more accurately.[18]

1.3 MACHINE LEARNING

Machine Learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access the data and use it learn for themselves. The name machine learning was coined in 1959 by Arthur Samuel. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks

where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders, and computer vision. Machine learning is closely related to and often overlaps with computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning. Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to produce reliable, repeatable decisions and results and uncover hidden insights through learning from historical relationships and trends in the data.

Machine Learning Categories

- **Supervised learning**: The computer is presented with example inputs and their desired outputs, given by a teacher, and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback.
- **Semi-supervised learning**: The computer is given only an incomplete training signal: a training set with some of the target output missing. Semi-supervised learning is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training.
- **Active learning**: The computer can only obtain training labels for a limited set of instances and also has to optimize its choice of objects to acquire labels. When used interactively, these can be presented to the user for labeling.
- **Unsupervised learning**: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself in discovering hidden patterns in the data or a means towards an end feature learning.
- **Reinforcement learning**: Data in form of rewards and punishments are given only as feedback to the program's actions in a dynamic environment. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

Machine learning Applications

- In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more multi-label classification of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email or other messages and the classes are spam and not spam.
- In regression, also a supervised problem, the outputs are continuous rather than discrete.
- In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- Density estimation finds the distribution of inputs in some space.

Dimensionality reduction simplifies inputs by mapping them into a lower- dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.

1.4 OBJECTIVES

- The objective of the project is to quickly identify patients who are suffering from cardiac arrhythmia who would require further diagnosis.
- Conventional method of identifying arrhythmia consumes more time compared to automated diagnosis using machine learning techniques.
- We make use of various machine learning techniques to check which is suitable for the given problem and proceed for the prediction.
- The outcome of the project is to classify the PCG recordings to normal or abnormal which would help doctors to proceed for further treatment.

CHAPTER 2

LITERATURE SURVEY

Electrocardiogram (ECG) electrical technique which is used to detect the activity of the heart. ECG helps in the instinctive classification of cardiac arrhythmia which saves the time of cardiologists through quick diagnostic results [17]. The explanation of the electrocardiogram signal is an application of pattern recognition, which instinctively groups it to a system to one among a number of different classes [4].

The main motive of this paper is to evolve a step forward to the latest technology from pattern recognition to machine learning to predict arrhythmia which is easy to detect using latest methodology, called Phonocardiogram (PCG) recordings which is a plot of murmurs and sound recordings at high-fidelity produced by the heart recorded by a machine called phonocardiography.[15] Thus, the recording of all the sounds made by the heart during a cardiac cycle is known as phonocardiography.

The proposed system classifies and distinguishes the specific cardiac arrhythmias, which is known to be Right Bundle Branch Blocks (RBBB) from the normal and paced heartbeats. Where normal heartbeats are the beats of healthy adult humans and paced heart beats are artificial beats from the pacemaker device.[12]

RBBB is a type of arrhythmia that is most commonly associated with hypertensive, ischemic, rheumatic and pulmonary heart disease [10]. Hence classification of heart sound recordings using Machine Learning techniques could help to overcome the better accuracy detection problem and which is also time-efficient. Machine learning, an intelligent multi-disciplinary approach which is highly used in supervised learning to construct analytical models.[13].

This plays a vital role in a wide range of serious applications, such as data mining, the processing of natural languages, image recognition, and skilled systems.[14] This approach seems appropriate to the problem of arrhythmia detection, since this problem can be transformed into a typical classification task. [11]

The aim of this paper is to identify different contributions made by various authors in detection of Cardiac arrhythmia and to enhance the view of the problem in a broader perspective.

2.1 COMPARISON OF DIFFERENT METHODOLOGIES USED IN DETECTING ARRHYTHMIA

Table 2.1 gives us a picturesque idea of the methods used by various cardiac researchers with Machine Learning. It also gives the list of recommendations that we thought might in future have been implemented in the system.

Ref no.	Objective	Concept Used	Results / Outcome	Advantage	Disadvantage
[1]	This paper is based on classification of morphological Arrhythmias designed for accurate detection of heartbeat in the ECG and it is robust.	Convolutional neural network (CNN)	The proposed methodology classifies the morphological arrhythmia with an accuracy rate of 92.14%	This method can be used in reducing the rate of misdiagnosed computerized interpretations of ECG.	Training data set were low, in number of hundreds.
[2]	Designing an efficient ensemble (network) of Classifiers to classify cardiac arrhythmias based on segments of ECG signals automatically.	Deep learning Ensemble learning,	The proposed method classifies ECG into 17 classes and 15 types of arrhythmias with the accuracy99.37%	The proposed model can evaluate the cardiac health immediately by implementing mobile devices or also can be applied this in cloud computing	ECG datasets of a particular database was taken. Not implemented for different cases.
[3]	This paper is based on the classification of arrhythmia into supra-ventricular arrhythmia (S), premature ventricular contraction (V), normal (N), Atrial Fibrillation (AF)	Maximum Mutual Information Estimation (MMIE) theory and Hidden Markov Models (HMM)	The proposed method effectively classifies the ECG into respective arrhythmia conditions.	There may be Confusion between the classes of beats which are morphologically similar can be reduced strongly by using MMIE training.	Difference between normal and arrhythmia datasets were not said.

[4]	This paper is based on the method, by Using Empirical Model Decomposition, Arrhythmia can be detected by using ECG signals.	Empirical Mode Decomposition (EMD), N-Bayes algorithms and linear discriminant analysis (LDA)	The proposed method classifies the normal signal with accuracy 92% and accuracy of LBB signal classification is81%	Proposed methodology provides an algorithm for accurate detection of normal and arrhythmic ECG signals.	Lacked the explanation of model. Explanation of signal classification not clear.
[5]	This paper is based on the detection of cardiac arrhythmia with the ECG signals using deep learning	deep learning, biomedical signal processing, convolutional neural networks and transfer learning	Different cardiac conditions like, normal, paced or right bundle branch block can be classified effectively	Recognition rate of the proposed method is 98.51% and testing accuracy around 92%.	ECG signals taken and result calculation method was not proper.
[6]	Objective of this paper is classifying cardiac arrhythmias and to studying the performance of the algorithms of machine learning.	OneR, J48 and Naïve Bayes algorithms.	Naïve Bayes and OneR shows the most stable rate of accuracy	The comparison between the accuracy of different algorithms helps to choose the best algorithm.	Only algorithms were considered for comparison and their working implementation is not clearly seen.
[7]	This paper is based on the classification and detection of four cardiac diseases	feature extraction algorithm (Pan Tompkins algorithm)	The proposed methodology classified four cardiac diseases like, Long Term Atrial Fibrillation (AF), Supraventricular Arrhythmia and Sleep Apnea, Arrhythmia	Forward net provides the best solution in most cases of arrhythmia	Feature extraction method employed is not appropriate in this approach.
[8]	Objective of this paper is to detect Robust heartbeat using ECG and BP signals	The comparison between the accuracy of different algorithms.	The proposed method successfully detects Robust Heartbeat.	The proposed method can be generalizable and accuracy is improved	Earlier methods were used, no new techniques were incorporated.
[9]	This paper is based on Classification of heartbeat for the detection of cardiac arrhythmia	Spiking neural network (SNN)	The proposed method classifies the incoming ECG heartbeats into one of 23 different classes and detects arrhythmia with an accuracy of 95.7%	Improvement in accuracy of 22% over state-of-the-art approaches.	Classification of Arrhythmia into different stages is not clearly explained.

Table 2.1: Various approach used in detecting arrhythmia by cardiac researcher's

CHAPTER 3

SOFTWARE REQUIREMENTS AND SPECIFICATIONS

3.1 SOFTWARE REQUIREMENTS AND SPECIFICATION

Software Requirements Specification (SRS) provides an overview of the entire SRS with purpose, scope, definitions, acronyms, abbreviations, references and overview of the SRS. A software requirements specification is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform the various gestures and determine its accuracy. The SRS is a requirements specification for a software system, is a description of the behavior of a system to be developed and may include a set of use cases that describe interactions the users will have with the software. In addition, it also contains non-functional requirements. Nonfunctional requirements impose constraints on the design or implementation. Software requirements specification establishes the basis for agreement between customers and contractors or suppliers on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

3.2 FUNCTIONAL REQUIREMENTS

A functional requirement defines a function of a software system or its components. A function is described as a set of inputs, the behavior, and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases. Functional requirements are supported by non-functional requirements also known as quality requirements, which impose constraints on the design or implementation such as

performance requirements, security. In some cases, a requirements analyst generates use cases after gathering and validating a set of functional requirements. The hierarchy of functional requirements is: user/stakeholder request → feature → use case → business rule. Each use case illustrates behavioral scenarios through one or more functional requirements. Often, though, an analyst will begin by eliciting a set of use cases, from which the analyst can derive the functional requirements that must be implemented to allow a user to perform each use case.

The Functional Requirements Specification documents the operations and activities that our system must be able to perform

- Take the symptoms as the input.
- Compare the symptoms with the disease symptoms matrix.
- Output the possible disease based on brute force algorithms.
- Questionnaires based on the particular symptoms to be asked.
- Take the intensity of symptoms numerically.
- Build the machine learning models with the actual datasets.
- Classify whether the user is suffering from the disease based on the intensities given by the user taking as the test case and to predict the chances of disease that the user is suffering from.

3.3 NON-FUNCTIONAL REQUIREMENTS

In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions. Non-functional requirements define how a system is supposed to be. Non-functional requirements are in the form of system shall be, an overall property of the system as a whole or of a particular aspect and not a specific function. The system's overall properties commonly mark the difference between whether the development project has succeeded or failed. Non-functional requirements are often called quality attributes of a system. Other terms for non-functional requirements are qualities, quality goals, quality of service requirements, constraints, non-behavioral requirements or technical requirements. Informally these are sometimes called ilities, form attributes like stability and portability. The plan for implementing non-functional requirements is detailed in the system architecture, because they are usually architecturally significant requirements.

3.4 OTHER NON-FUNCTIONAL REQUIREMENTS

Following factors are used to measure software development quality. Each attribute can be used to measure the product performance. These attributes can be used for quality assurance as well as quality control. Quality assurance activities are oriented towards prevention of introduction of defects and quality control activities are aimed at detecting defects in products and services.

- **Reliability:** Measure if the product is reliable enough to sustain in any condition and give consistently correct results. Product reliability is measured in terms of working on a project under different working environments and different conditions.
- **Maintainability:** Different versions of the product should be easy to maintain. For development it should be easy to add code to existing systems and should be easy to upgrade for new features and new technologies time to time. Maintenance should be cost effective and easy. System be easy to maintain and corrects defects or makes a change in the software.
- **Usability:** This can be measured in terms of ease of use. Application should be user friendly. The system must be easy to use for input preparation, operation, and interpretation of output, and provide consistent user interface standards or conventions with our other frequently used systems. They should be easy for new or infrequent users to learn to use the system.
- **Portability:** This can be measured in terms of costing issues related to porting, technical issues related to porting, behavioral issues related to porting.
- **Correctness:** Application should be correct in terms of its functionality, calculations used internally and the navigation should be correct. This means applications should adhere to functional requirements.
- **Efficiency:** It is a quality attribute. Measured in terms of time required to complete any task given to the system. For example, systems should utilize processor capacity, disk space and memory efficiently. If the system is using all the available resources then the user will get degraded performance failing the system for efficiency. If the system is not efficient then it cannot be used in real time applications.

3.5 HARDWARE REQUIREMENTS

- **Processor:** Intel CORE i5 or higher
- **RAM:** 4GB RAM
- **Monitor:** EGVGA Compatible
- **Keyboard:** Normal keyboard (QWERTY)
- **Hard Disk:** Atleast 2GB

3.6 SOFTWARE REQUIREMENTS

- **Operating system:** Windows/Linux.
- **Language:** Python.
- **Libraries:** Python Libraries, Keras, Numpy, Matplotlib.
- **IDE:** Anaconda, Jupyter Notebook, iPython.
- **Editor:** Google Colab.

CHAPTER 4

SYSTEM ANALYSIS

Cardiac Arrhythmia prediction has been regarded as a critical topic. Artificial intelligence and machine learning techniques have already been developed to solve this type of medical care problem. Recently, neural network ensembles have been successfully utilized in a variety of applications including to assist in medical diagnosis. ML ensembles can significantly improve the generalization ability of learning systems through training a finite number of neural networks and then combining their results. However, the performance of multiple classifiers in arrhythmia prediction is not fully understood. The major purpose of this study is to investigate the performance of different models among the various available models and to check which models predict the given sample accurately. In addition, we use various evaluation criteria like accuracy, loss, precision, recall and f-beta score to examine the performance of these models with real-life datasets.

4.1 PROPOSED SYSTEM

The proposed model consists of a convolution neural network built using different keras activation function model and transfer learning model. The training data consist of about 3700 spectrogram images for training and validation. The training is done using a GPU. The model is implemented considering the functionalities of ReLU, ELU and Tanh activation functions and the transfer learning method. We consider accuracy, precision, recall, f-beta score and loss as parameters in evaluation of the models developed and based on the scores obtained from the above parameters the robustness and efficiency of that model was predicted and further used to predict the normal and abnormal spectrograms.[20]

In the first model we used the ReLU activation function to develop a CNN model consisting of numerous nodes in the network to train and validate the given dataset. This function provides us to use gradient descent with backpropagation of errors to train the deep neural network.

In the second model we used the ELU activation function to develop a CNN model which provides us the convergence and with the extra alpha constant provides a positive number to differentiate among the dataset.

In the third model we used Tanh activation function to develop a CNN model which provides non-linear functionality and with a range of -1 to 1. It has a steeper derivative compared to others and hence we used this model to compare the results.[20]

In the fourth model to implement using a different approach we used Transfer Learning model to see if there is any change exists and to compare among other models. So, we decided to implement a pre trained model VGG16. This process allows us to use the pre-trained models trained on datasets with millions of images such as COCO, Imagenet etc.

4.2 OPERATIONS AND ACTIVITY

Operations and activities that the system is able to perform:-

- Convert the heart recordings into spectrogram images.
- Evaluate model on accuracy, precision, recall, f-beta score and loss
- Implement a Convolution Neural Network and Transfer Learning model to train and validate the given dataset.
- Evaluate the model on accuracy and predict the new spectrogram image given as normal or abnormal.

4.3 ADVANTAGES OF PROPOSED SYSTEM

- Automatic heart sound classification has promising potential to accurately detect heart diseases such as cardiac.
- It could be used in non-clinical environments such as a patient residence by medical personnel as a quick heart pathology screening technique or at places with poor financial conditions to support healthcare.
- The focus of this project is to classify whether the patient has normal or abnormal heart sound from the Phonocardiogram (PCG) or heartbeat recordings to quickly identify patients who would require further diagnosis.
- This is a supervised learning problem since we already know if the heart sound in the training dataset is normal or abnormal.
- The basic idea is to convert each heart sound recording (wav file) to a spectrogram image and train a Convolutional Neural Network over those images. Then given a new PCG recording, we will be able to classify it as normal or abnormal.

CHAPTER 5

SYSTEM DESIGN

The purpose of the design is to obtain the solution of a problem specified by the system requirements. This phase is the first step in moving from problem to solution domain. In other words, starting with what is needed to how long it takes to design it, defines how to satisfy the needs. The design of the system is perhaps the most critical factor affecting the quality of the software and has a major impact on the later phases, particularly testing and maintenance. Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development.

5.1 SYSTEM ARCHITECTURE

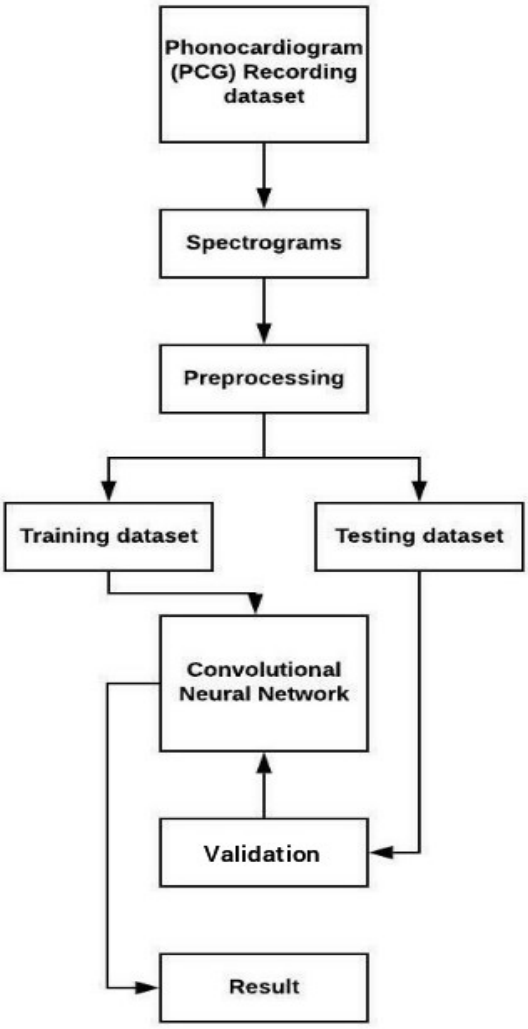

Figure 5.1: Flowchart of system development

The focus of this project is to classify whether the patient has normal or abnormal heart sound from the Phonocardiogram (PCG) or heartbeat recordings to quickly identify patients who would require further diagnosis. This is a supervised learning problem since we already know if the heart sound in the training dataset is normal or abnormal. The basic idea is to convert each heart sound recording (wav file) to a spectrogram image and train a convolutional neural network over those images. Then given a new PCG recording, we will be able to classify it as normal or abnormal. The figure 5.1 depicts the workflow as mentioned.

5.2 METRICS

The metrics to determine how well the model performs on the entire dataset is logarithmic loss which is also named as categorical cross entropy. The formulae is shown below.

$$\text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\ log\ (p_{i,j}) \qquad \text{……..(i)}$$

This can be described as negative; the log likelihood of the model given each observation is chosen independently from a distribution that places the predicted probability mass on the corresponding class, for each observation. In our case, each recording is already labeled with one true class and for each one, a set of predicted probabilities is considered an outcome. Here N is the number of images in the test set, M is the number of image class labels i.e. 2, log is the natural logarithm, $y_{i,j}$ is 1 if observation belongs to class and 0 otherwise, and $p_{i,j}$ is the predicted probability that observation belongs to a given class.

Metric	Formula
True positive rate, recall	TP / TP+FN
False positive rate	FP / FP+TN
Precision	TP / TP+FP
Accuracy	TP+TN / TP+FN+TN+FP
F-measure	2*precision*recall / precision + recall

Table 5.2: Formulae to calculate the metrics

Since the dataset is imbalanced i.e. number of normal samples is greater than number of abnormal samples in the training dataset, accuracy is not only the metric considered. Hence

the model was evaluated on precision which can be described as the ratio of correct positive predictions made out of the total positive predictions made, recall which is the ratio of correct positive predictions made out of the actual total that were positive. We also calculated another metric known as f-beta score which is given by the weighted average of precision. The formulae to calculate these metrics are given in table 5.2.

The Development Phases in Brief

- **Planning Phase:** The Planning phase began with figuring out a way to develop a framework with organized machine learning models working at an environment agnostic level. Thus, to implement this model we planned to build an environment which will learn from dataset and from our observations and learning acquired by training the model we will be able to implement machine learning algorithms and to predict the input accurately.
- **Analysis Phase:** This phase began with the analysis of the requirements needed to build the working model of our system by preparing an abstract and carrying out a literature survey and realizing about the different approaches to go about and develop the system. The analysis revealed that the primary difficulty arises due to insufficient explorations of data.
- **Design Phase:** We built our design by partitioning the modules of the whole system into two levels, the high-level design and the low-level design. In the high level we present a system that represents in its most abstract level which consisted of the methodologies and necessary approaches about designing the environment. In the lower layer of the design we have focused mainly on the build and detailed design of the ML models and how it could perhaps be used to train the model using the available dataset using keras libraries.

High Level Design

High-level design (HLD) explains the architecture that would be used for developing a software product. The architecture diagram provides an overview of an entire system, identifying the main components that would be developed for the product and their interfaces. The HLD uses possibly nontechnical to mildly technical terms that should be understandable to the administrators of the system.

CHAPTER 6

ALGORITHMS AND TECHNIQUES

Recent algorithms applied to cardiology challenges include heart sound segmentation, transformation of one-dimensional waveforms into two-dimensional time frequency heat map representations using mel-frequency coefficients and classification of MFCC heat maps. But given the recent success of deep neural networks in computer vision, we decided to use convolutional neural networks for this problem. CNNs are the modern-day state-of-the-art algorithm for image classification and visual recognition tasks and have now improved to the point where they now outperform humans on computer vision challenges such as ImageNet and COCO.

CNN are like normal neural networks. Every neuron in each layer receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network expresses a single differentiable score function. But instead they take images as input which allows us to encode certain properties into the architecture and outputs classes. Ex: dog, cat, tree, etc. according to our dataset. A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume. They are composed of many layers like convolution, pooling, fully connected as shown in the figure 6.1.

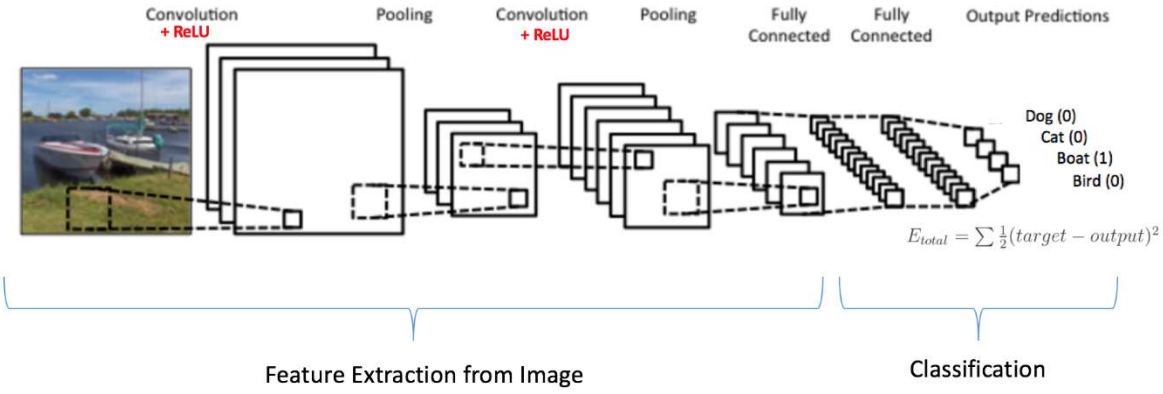

Figure 6.1: CNN architecture is formed by a stack of distinct layers

Implementing a CNN from scratch

Consider our input image to be 32x32x3. On each 2D array of data, we train a whole bunch of N x N kernels. These kernels are nothing but filters that we run across the 2D array. For each position (x,y), we compute the dot product summation between this kernel and the

image values around that point. This process is called convolution, hence the name Convolutional Neural Networks. In this way, we are able to detect edges, lines, blobs of colors and other visual elements. Then we apply a nonlinear layer or activation layer immediately afterward to introduce nonlinearity to a system that basically has just been computing linear operations during the convolution layers. We will then apply a pooling layer which outputs the maximum number in every subregion that the filter convolves around. This reduces the spatial dimension the length and the width but not the depth of the input volume from 32x32 to 16x16 and the number of parameters or weights drastically. In the end, we take all these features and apply what is called a fully connected layer to it. In this layer, each neuron in this layer will be connected to all the neutrons in the previous one just like in a normal neural network. It looks at the high-level features which correlate to a certain class so that when we compute the products between the weights and the previous layer, we get the correct probabilities for the different classes. In a nutshell, we decided to use CNN for this problem because contrary to normal neural networks where all the neurons in one layer are connected to all the neurons in the previous layer and the operation becomes computationally very intensive, convolutional layers are technically locally connected layers and all the pixel positions share the same filter weights. This significantly reduced the number of parameters and gives better results.

6.1 ACTIVATION FUNCTIONS

RELU: A recent invention which stands for Rectified Linear Units. The formula is deceptively simple: max (0, z) and min (0, z). Despite its name and appearance, it's not linear and provides the same benefits as sigmoid, with better performance as shown in figure 6.1.1.

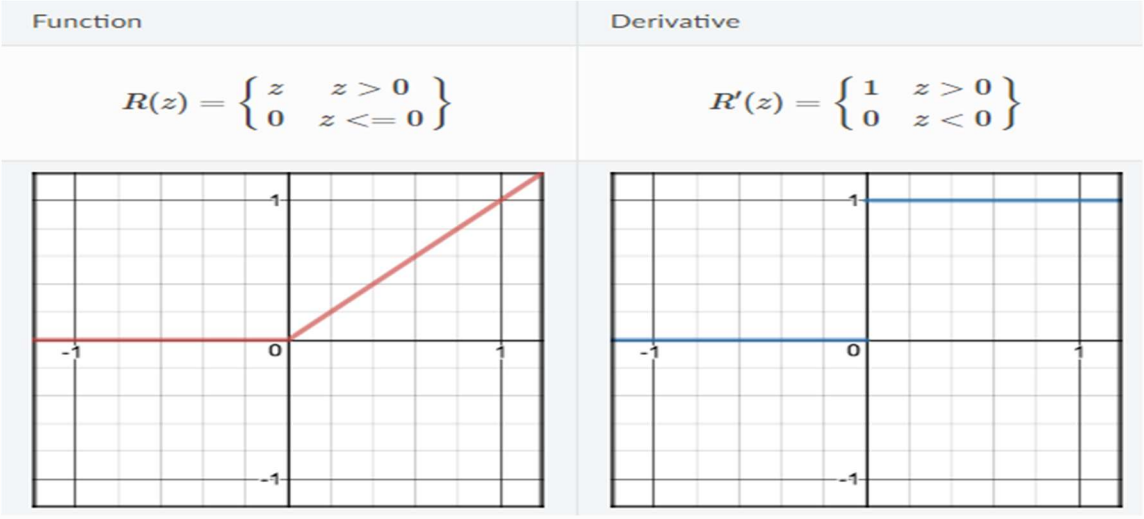

Figure 6.1.1: Shows the function and derivative plot of ReLU function

Pros

- It avoids and rectifies the vanishing gradient problem.
- ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.

Cons

- One of its limitations is that it should only be used within hidden layers of a neural network model.
- Some gradients can be fragile during training and can die. It can cause a weight update which will make it never activate on any data point again. Simply saying that ReLU could result in dead neurons.
- In other words, for activations in the region (x<0) of ReLU, gradient will be 0 because of which the weights will not get adjusted during descent. This is called the dying ReLU problem.
- The range of ReLU is [0, inf). This means it can blow up the activation.

ELU: Exponential Linear Unit or its widely known name ELU is a function that tends to converge cost to zero faster and produce more accurate results. Different to other activation functions, ELU has an extra alpha constant which should be a positive number. ELU is very similar to ReLU except negative inputs. On the other hand, ELU becomes smooth slowly until its output equals -α whereas ReLU sharply smoothens as shown in figure 6.1.2.

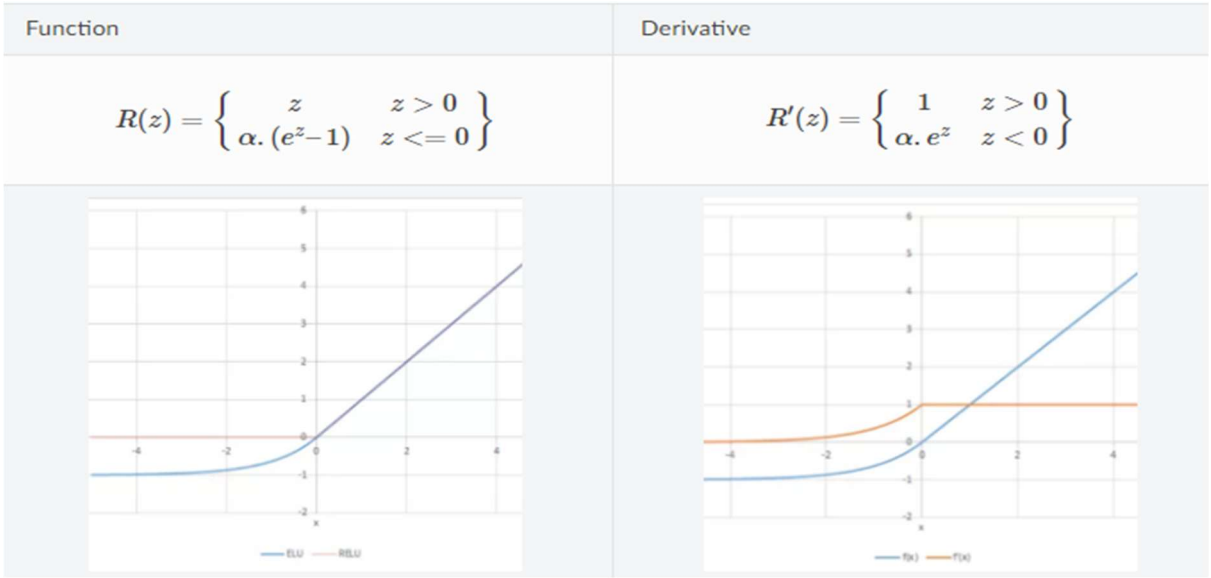

Figure 6.1.2: Shows the function and derivative plot of ELU function

Pros

- ELU becomes smooth slowly until its output equals -α whereas RELU sharply smoothens.
- ELU is a strong alternative to ReLU.
- Unlike ReLU, ELU can produce negative outputs.

Cons

- For x > 0, it can blow up the activation with the output range of [0, inf].

Tanh: Tanh squashes a real-valued number to the range [-1, 1]. It's non-linear. But unlike sigmoid, its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity as shown in figure 6.1.3.

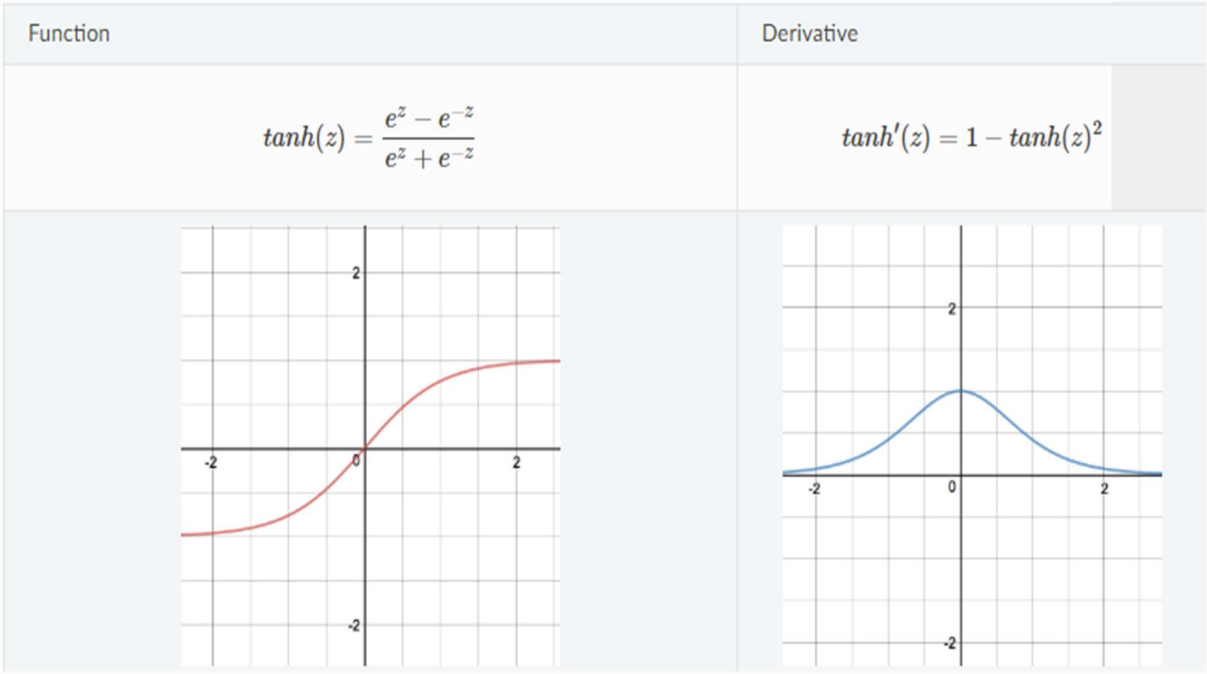

Figure 6.1.3 shows the function and derivative plot of Tanh function

Pros

- The gradient is stronger for tanh than sigmoid as derivatives are steeper.

Cons

- Tanh also has the vanishing gradient problem.

SIGMOID: Sigmoid takes a real value as input and outputs another value between 0 and 1. It's easy to work with and has all the nice properties of activation functions: it's non-linear, continuously differentiable, monotonic, and has a fixed output range as shown in figure 6.1.4.

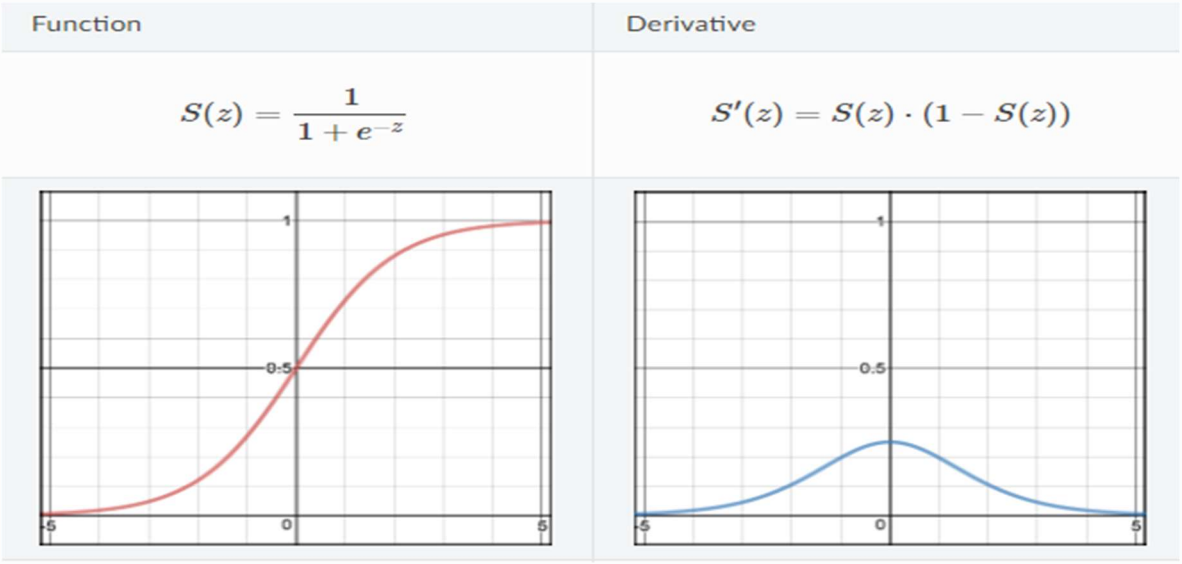

Figure 6.1.4 shows the function and derivative plot of Sigmoid function

Pros

- It is nonlinear in nature. Combinations of this function are also nonlinear.
- It will give an analog activation unlike step function.
- It has a smooth gradient too.
- It's good for a classifier.
- The output of the activation function is always going to be in range (0,1) compared to (-inf, inf) of linear function. So, we have our activations bound in a range. Then, it won't blow up the activations then.

Cons

- Towards either end of the sigmoid function, the Y values tend to respond very less to changes in X.
- It gives rise to a problem of vanishing gradients.
- Its output isn't zero centred. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.
- The network refuses to learn further or is drastically slow depending on use case and until gradient /computation gets hit by floating point value limits.

6.2 TRANSFER LEARNING

Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks.

We implemented a pre trained model VGG16. This process allows us to use the pre-trained models trained on datasets with millions of images such as COCO, Imagenet etc. This means instead of building a model from scratch to solve a similar dataset, we can use the model trained on some other dataset as a starting point. The last layer of the network was removed and replaced with custom classifiers i.e. classifying between normal and abnormal. This means we used the network only upto the fully connected layers and removed all subsequent layers. Since the structure of our model was not exactly the same as the one used when training weights.

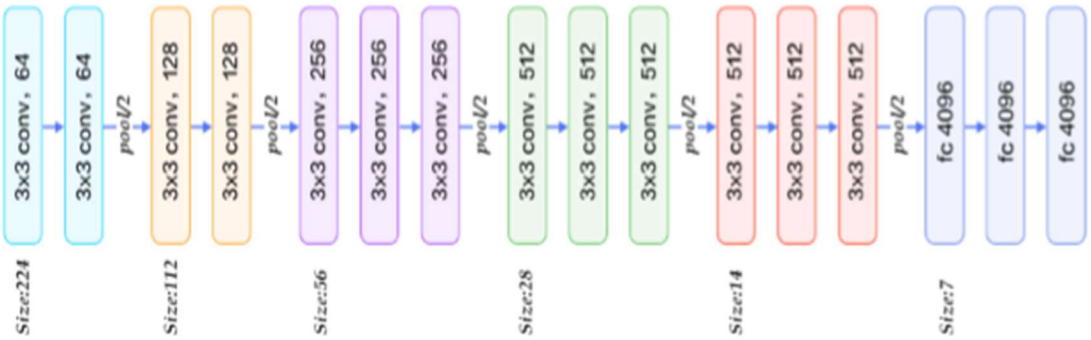

Figure 6.2: The architecture for VGG16 model

The above figure 6.2 shows the architecture for the VGG16 model. It is important to note that input dimensions for this model were 150x150 instead of 224x224 as shown in the figure. The architecture for this model is as follows: Each block is stacked with convolution layers and at the end of every block a pooling layer to reduce spatial dimension is applied. The output dimensions are increased by a factor of 2 every block. It should also be noted that zero padding is applied before every convolution layer in each block since it is not shown in the figure. The last three fully connection layers were removed as discussed above. For experiment purpose, we also decided to implement the VGG16 model architecture from scratch including the fully connected layer to see how it performed.

CHAPTER 7

IMPLEMENTATION

Implementation of any software is always preceded by important decisions regarding selection of platform, the language used, etc. These decisions are often influenced by several factors such as the real environment in which the system works, the speed that is required, the security concerns, other implementation specific details etc. An implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through computer programming and deployment. Many implementations may exist for a given specification or standard. For example, web browsers contain implementations of World Wide Web consortium recommended specifications, and software development tools contain implementations of programming languages.

The design carried out in order to develop the model for being able to train and test and imply machine learning was translated into a working model by implementing our design process which was broken down into three segments of programmed source code. Each of the segments were responsible for handling an aspect of the working model. We have described the working of each of the segments along with the code snippets and implementation in the following subsections.

7.1 ABOUT TOOLS

The tools which were used for implementation of the project are as follows

- Python 3 and above
- Keras, Python libraries
- Anaconda3
- iPython and Jupyter notebook
- Google Colab

PYTHON

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido Van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped

down as the leader in the language community after 30 years. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems. C-Python, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and C-Python are managed by the non-profit Python Software Foundation. Python uses dynamic typing, and a combination of reference counting and a cycle detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

ANACONDA

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. The open- source Anaconda Distribution is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 11 million users worldwide, it is the industry standard for developing, testing, and training on a single machine. The Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. Jupyter has developed and supported the interactive computing products Jupyter Notebook, Jupyter Hub, and Jupyter Lab, the next-generation version of Jupyter Notebook. The Jupyter Notebook application allows to create and edit documents that display the input and output of a Python or R language script. JupyterLab is the next-generation user interface for Project Jupyter. It offers all the familiar building blocks of the classic Jupyter Notebook, terminal, text editor, file browser, rich outputs, etc. in a flexible and powerful user interface.

NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on the arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with

extensive modifications. NumPy is opensource software and has many contributors. NumPy provides a high-performance multidimensional array and basic tools to compute with and manipulate these arrays. SciPy builds on this and provides a large number of functions that operate on NumPy arrays and are useful for different types of scientific and engineering applications.

PANDAS

Pandas is a software library written for the python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term panel data, an econometrics term for data sets that include observations over multiple time periods for the same individuals. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with pandas is used in a wide range of fields including academic and commercial domains including finance, economics, statistics, analytics, etc.

JUPYTER AND iPYTHON

Project Jupyter is a non-profit, open-source project, born out of the IPython project in 2014 as it evolved to support interactive data science and scientific computing across all programming languages. Jupyter will always be 100% open-source software, free for all to use and released under the liberal terms of the modified BSD license.

Jupyter is developed in the open on GitHub, through the consensus of the Jupyter community. All online and in-person interactions and communications directly related to the project are covered by the Jupyter Code of Conduct. This Code of Conduct sets expectations to enable a diverse community of users and contributors to participate in the project with respect and safety.

iPython Notebook: Interactive Python is a command shell for interactive computing in multiple programming languages, originally developed for the Python programming language, that offers introspection, rich media, shell syntax, tab completion, and history.

Colab Notebook: Colaboratory is a free Jupyter notebook environment that runs in the cloud and stores its notebooks on google drive. All notebooks are stored in Jupyter notebook format (.ipynb).

The enhanced interactive Python shells and kernel have the following main features:

- Comprehensive object introspection.
- Input history, persistent across sessions.
- Caching of output results during a session with automatically generated references.
- Extensible tab completion, with support by default for completion of python variables and keywords, filenames and function keywords.
- Extensible system of magic commands for controlling the environment and performing many tasks related to IPython or the operating system.
- A rich configuration system with easy switching between different setups simpler than changing $PYTHONSTARTUP environment variables every time.
- Session logging and reloading.
- Extensible syntax processing for special purpose situations.
- Access to the system shell with user-extensible alias system.
- Easily embeddable in other Python programs and GUIs.

KERAS

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling. Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU) principally in conjunction with CUDA. Keras follows best practices for reducing cognitive load.

7.2 DATA EXPLORATION AND VISUALIZATION

The dataset used for this project is available freely as part of the Physio Net in Cardiology Challenge 2016 which focuses on automatic classification of normal or abnormal phonocardiogram (PCG) recording. The training data consists of PCG signals of varying length, between 5s to 120s sampled at 2000 Hz and are provided in .wav format. Therefore, we choose the training set to be about 2000 labelled recordings and the validation set to be 686 recordings. The dataset is based on time-frequency features, to capture the intensity and the pitch of the recordings. Due to its robustness and interpretability, spectrograms are a visual way of representing the signal strength or loudness of a signal. Therefore, raw wav files need parsed into spectrograms first and the class labels are extracted from header files. Splitting the dataset into training and testing sets is also automatically taken care of by the script.

To convert .wav recording to spectrogram, we use python code and a few library functions. We import the wav file to the program. We take the data in the form of samples with respect to time and frequency on the x and y axis respectively. We use the function plt.pcolormesh to indicate colour to spectrogram which helps to improve the classification, pt.imshow function to display the spectrogram with respect to each wav file provided as a sample.

The recordings in the dataset are based on time-frequency features of the raw signal. So, to capture the intensity and the pitch of the recordings, we decided to use spectrograms. Due to its robustness and interpretability, Spectrograms have been applied to a wide range of areas since it a visual way of representing the signal strength, or loudness, of a signal over time at various frequencies present in a waveform. Therefore, raw wav files need to be parsed into spectrograms first and the class labels are extracted from header files. To do this, we decided to write a python script separately from the project notebook to convert every recording into its subsequent spectrogram. Splitting the dataset into training and testing sets is also automatically taken care of by this script.

The script follows the following approach

1. Load the .wav files into memory.
2. Process all the recordings and header files and parse them for file names and class labels and store them in separate lists. Once we have all the class labels and file names, we use scipy's wave file class to get the sample rate and data of the wav file.

3. The length of the windowing segments and sampling frequency for the spectrograms will be 256. Matplotllib's spectrogram() function computes and plots a spectrogram. It takes minimum 3 parameters:-

   x: Array or sequence containing the data or data of wav file

   Fs: The sampling frequency or the samples per time unit which is used to calculate Fourier frequencies

   NFFT: The number of data points used in each block for the FFT.

4. The spectrogram is then plotted as a colormap.

Finally, using the list of class labels we separate the spectrograms into individual classes. This way we can easily feed the images directly into our model using keras inbuilt image data generator class. The directory of the dataset is organized into folders of training and validation. In turn, the folders are sub-classified as normal and abnormal to differentiate the given spectrograms to the desired folder. The whole part is taken care of by the python file written to convert the heart recordings into its spectrogram image and then used to construct the machine learning model to predict the same.

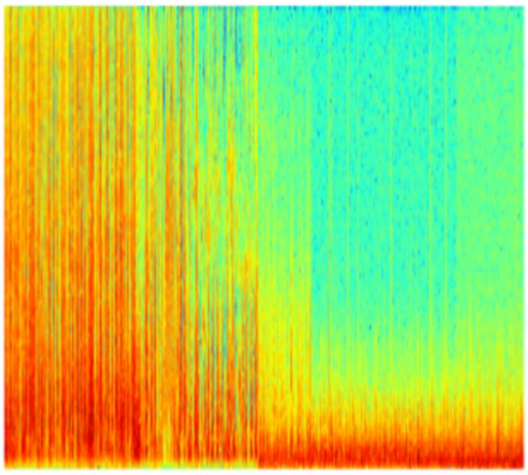

Figure 7.1.1: An abnormal heartbeat

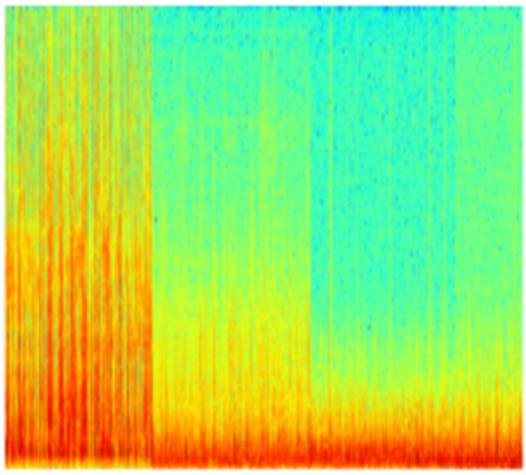

Figure: 7.1.2 A normal heartbeat

The figure 7.1.1 and figure 7.1.2 shows the spectrogram image of an abnormal and normal heartbeat which is obtained after the heartbeat recording is passed through the python program to convert it to spectrogram images.

Let's take a look at what the spectrogram does. At the most basic level, the spectrogram is

dividing the sound sample into multiple blocks in time domain and plotting the Fast Fourier Transform (FFT) of each block and displaying them in the same graph. The x-axis is time and the y axis is frequency and the analysis is done only on one of the blocks of the time domain. The amplitude of FFT for each time domain block is represented by color and intensity. This is used to show how energy levels vary over time.

The image below shows the number of samples for each class in training and validation set.

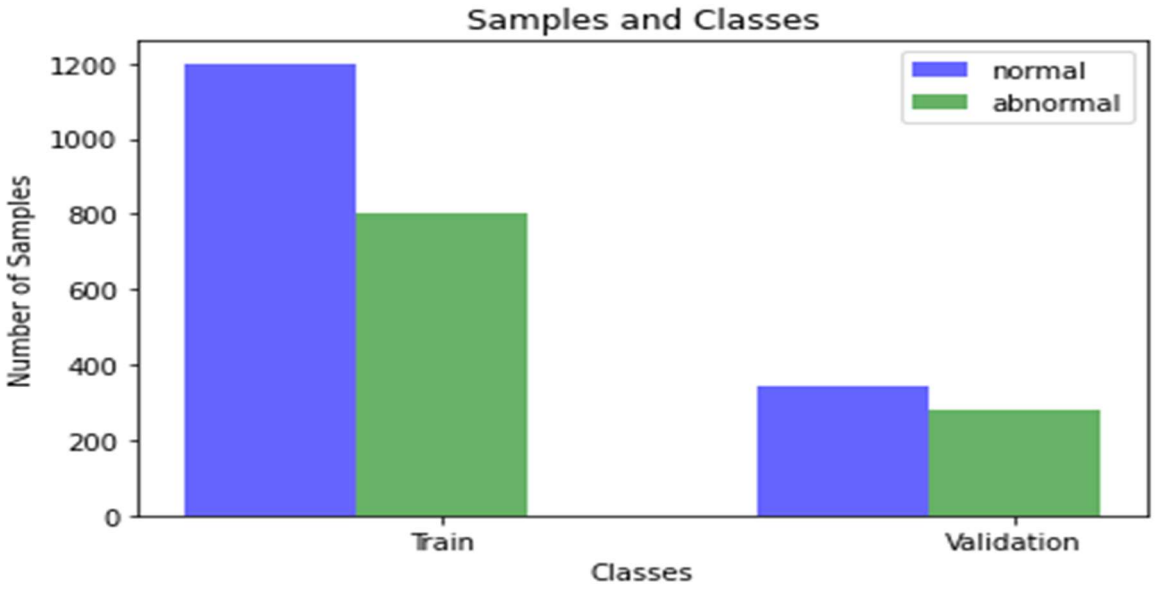

Figure 7.1.3: The number of samples for each class in the training and validation set

The above figure 7.1.3 depicts the bar graph of training and validation samples in the dataset, which are used for training and prediction of the model.

Data Preprocessing

Data Augmentation will be important for our model so that it never sees the exact same picture twice since different spectrograms look similar. This helps prevent overfitting and the model generalizes better. For images, this could be done by rotating the original image, changing lighting conditions, cropping it differently, so for one image we can generate different sub-samples.

For this capstone, the following data augmentation techniques were applied:

• **Resizing**: All the images in the dataset both training and testing were resized to 150x150x3 where width = height = 150 and colour channel = 3 before feeding into the CNN.

• **Normalize**: Pixel values for all images were normalized between 0 and 1. This was done by subtracting the minimum pixel (i.e. 0) and dividing by maximum pixel value (i.e. 255). In Keras, this was done by setting the rescale attribute to 1/.255.

• **Shear Transformation**: Shear Transformation was applied to control the shear intensity of the input images. It was set to 0.2 using the shear-range attribute.

• **Zooming**: Randomly zooming inside images by setting zoom-range attribute to 0.2

• **Flipping**: Half of the training inputs were randomly flipped horizontally using the horizontal-flip attribute.

All the augmentation techniques were implemented using keras image data generator class which takes in the above-mentioned attributes and provides us with real-time data augmentation. Apart from these, the image data generator class has the method flow from directory in which it takes the path to a directory and generates batches of augmented data. The data was looped over in batches indefinitely. It further provides us with the number of images in each set. Splitting the dataset into a training and testing set was done manually before applying the data preprocessing steps.

7.3 MACHINE LEARNING MODELS

The CNN models were implemented to check the accuracy obtained from the machine learning models obtained after the training and validation set is complete. We used ReLU, ELU and Tanh activation functions of keras to appropriately adjust weights in the neural network and to get correct prediction of results.

Model Architecture for initial network

We first implemented a small covnet with a simple stack of 3 convolution layers with a ReLU activation and followed by a max-pooling layer. This is then followed by two fully-connected layers. In the end we are left with a single unit and hence use sigmoid activation since this is a binary classification. Each convolution block creates a convolution kernel which is convolved with the input layer over a single spatial followed by a nonlinear activation function. The pooling layer downscales the input in both spatial dimensions. The number of filters, filter size and downscale pooling size for 1st, 2nd and 3rd layers are [32, (3,3), (2,2)], [32, (3,3), (2,2)] and [64, (3,3), (2,2)] respectively. After flattening the image i.e. converting our 3D feature maps to 1D feature vectors we apply the first fully connected layer which has 64 units and activation function. Finally, we apply sigmoid activation to the last unit to classify into normal or abnormal. The model is then saved as a file in (.h5) format and imported to the prediction function. The prediction function takes this model and a new spectrogram image to classify it as normal or abnormal.

Model Architecture for updated deeper network

After the initial model, we decided to implement a deeper covnet. The stack for each block is as follows:- We first applied zero padding i.e. adding rows and columns of zeros at the top, bottom, left and right side of an image. Then convolutions and activation function are followed by the batch normalization layer. This way the activations of the previous layer at each batch are normalized i.e. mean activation is almost 0 and the activation standard deviation close to 1. After this we applied a max-pooling layer. This architecture has 5 such blocks. This is then followed by two fully-connected layers. We also used dropout to avoid overfitting. In the end we are left with a single unit and hence use sigmoid activation since this is a binary classification. Kernel size for zero padding, filter size, downscale size in each block is the same and is (1,1), (3,3) and (2,2) respectively. The number of filters for 1st, 2nd, 3rd, 4th and 5th layers are 64, 128, 256, 512, 512 respectively. After flattening the image i.e. converting our 3D feature maps to 1D feature vectors we apply the first fully connected layer which has 64 units and activation function. Finally, we apply sigmoid activation to the last unit to classify into normal or abnormal.

7.4 IMPLEMENTED MODEL

Models in Keras are defined as a sequence of layers. We create a Sequential model and add layers one at a time to our network architecture. The first thing to get right is to ensure the input layer has the right number of input features. This can be specified when creating the first layer with the input_dim argument and setting it to 8 for the 8 input variables. Fully connected layers are defined using the Dense class. We can specify the number of neurons or nodes in the layer as the first argument, and specify the activation function using the activation argument.

We will be using Rectified Linear Unit (ReLU), Exponential Linear Unit (ELU) and Tangent Hyperbolic (Tanh) for the first, second and third model respectively with Sigmoid function at the end of the layers. To improve the original model, we decided to implement a pre-trained model VGG-16. This process is known as Transfer Learning.

Compiling the model uses the efficient numerical libraries under the backend such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware, such as CPU or GPU or even distributed. When compiling, we must specify some additional properties required when

training the network. The training of a network means finding the best set of weights to map inputs to outputs in our dataset. We must specify the loss function to use to evaluate a set of weights, the optimizer is used to search through different weights for the network and any optional metrics we would like to collect and report during training.

We will define the optimizer as the efficient stochastic gradient descent algorithm adam. This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems. Finally, because it is a classification problem, we will collect and report the classification accuracy, defined via the metrics argument.

We have defined our model and compiled it ready for efficient computation. We can train or fit our model on our loaded data by calling the fit**()** function on the model. Training occurs over epochs and each epoch is split into batches. Epoch is one pass through all of the rows in the training dataset and batch is one or more samples considered by the model within an epoch before weights are updated. One epoch is composed of one or more batches, based on the chosen batch size and the model is fit for many epochs. The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the epochs argument. We must also set the number of dataset rows that are considered before the model weights are updated within each epoch, called the batch size and set using the batch_size argument. These configurations can be chosen experimentally by trial and error. We want to train the model enough so that it learns a better mapping of rows of input data to the output classification. The model will always have some error, but the amount of error will level out after some point for a given model configuration. This is called model convergence.

We have trained our neural network on the entire dataset and we can evaluate the performance of the network on the same dataset. You can evaluate your model on your training dataset using the evaluate**()** function on your model and pass it the same input and output used to train the model. This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics you have configured, such as accuracy. The evaluate**()** function will return a list with two values. The first will be the loss of the model on the dataset and the second will be the accuracy of the model on the dataset. We are only interested in reporting the accuracy, so we will ignore the loss value. On computation of different machine learning models, the model which has the highest accuracy is taken for validation.

CHAPTER 8

TESTING

System testing is the stage before system implementation where the system is made an error tree and all the needed modifications are made. The system was tested with test data and necessary corrections to the system were carried out. All the reports were checked by the admin and approved.

8.1 SOFTWARE TESTING

A test plan is a general document for the entire project, which defines the scope, approach to be taken, and schedule of testing, as well as identifying the test item for the entire testing process, and the personnel responsible for the different activities of testing. This document describes the plan for testing, the knowledge management tool.

Major testing activities are:

- Test units
- Features to be tested
- Approach for testing
- Test deliverables
- Schedule
- Personal allocation

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort traditionally occurs after the requirements have been defined and the coding process has been completed having been shown that fixing a bug is less expensive when effort traditionally occurs after the requirements have been found earlier in the development process. Different software development models will focus the test effort at different points in the development process. Different software development models will focus the test effort at different points in the development process. Newer development models such as agile, often employ test driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed. The obtained results show the correctness of the model and where the execution has gone wrong for various given inputs.

8.2 TESTING PROCESS

An important point is that software testing should be distinguished from the separate discipline of Software Quality Assurance (SQA), which encompasses all business process areas, not just testing. Software Testing may be compared to tasting the food before being served to the guests. The best purpose of both is to ensure a defect free delivery.

Testing Objectives

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as-yet-undiscovered error.
- A successful test case is one that uncovers an as-yet-undiscovered error.

These objectives imply a dramatic change in the viewpoint. They move opposite to the commonly held view that a successful test is one in which no errors are found. Our objective is to design test cases such that they systematically uncover different classes of errors with minimum time and effort. Testing is mainly used to determine the quality of the product according to the user needs.

8.3 TYPES OF TESTING

- **Unit Testing**: Here we test each module individually and integrate the overall system. Unit testing focuses verification efforts even in the smallest unit of software design in each module. This is also known as module testing. The modules of the system are tested separately. This testing is carried out in the programming style itself. In this testing each module is focused to work satisfactorily as regard to expected output from the module there are some validation checks for the fields.
- **Integration Testing**: Data can be lost across an interface, one module can have an adverse effect on the other sub functions, when combined may not produce the desired functions. Integrated testing is the systematic testing to uncover the errors with an interface. This testing is done with simple data and the developed system has run successfully with this simple data. The need for an integrated system is to find the overall system performance.
- **System Testing**: Taking various kinds of data plays a vital role in system testing. After preparing the test data, the system under steady is tested using the test data.

While testing errors are again uncovered and corrected by using the above steps and corrections are also noted for future use.

CHAPTER 9

RESULT AND SNAPSHOT

We evaluate our models based on the measures taken like accuracy, precision, f-beta score and recall. We try to minimize the loss function in all the models to achieve better accuracy. Overall, we get a whole simulated model to find the abnormal and normal spectrograms.

- **Accuracy:** Accuracy is the proximity of measurement results to the true value. In the fields of science and engineering, the accuracy of a measurement system is the degree of closeness of measurements of a quantity to that quantity.
- **Precision**: Precision is a description of random errors, a measure of statistical variability. It is the degree to which repeated measurements under unchanged conditions show the same results.
- **Recall:** Examples of measures that are a combination of precision and recall are the F-measure the weighted harmonic means of precision and recall, or the Matthews correlation coefficient, which is a geometric mean of the chance-corrected variants.
- **F-beta score**: The F-beta score is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0. The beta parameter determines the weight of recall in the combined score. beta < 1 lends more weight to precision, while beta > 1 favours recall where beta -> 0 considers only precision, beta -> +inf only recall.
- **Loss:** A loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some cost associated with the event. An optimization problem seeks to minimize a loss function.

We ran the tests for the machine learning model built using ReLU, ELU, Tanh activation function and the transfer learning model and obtained results for the above measures. The values in table 9.1 are obtained after tests are run for the dataset obtained of about three thousand heart recordings. These values show the average of 100 epochs that are run on every model. Considering the parameters mentioned in the metrics the accuracy of the model is calculated and is displayed in the form of table. Based on the performance of each model on training and validation dataset graph is plotted.

ML Model	Accuracy	Precision	Recall	F-beta Score	Loss
Model 1 (ReLU)	0.8006	0.7862	0.9175	0.8418	0.5545
Model 2 (ELU)	0.7326	0.6852	0.8875	0.7814	0.6328
Model 3 (Tanh)	0.7689	0.7112	0.8913	0.8018	0.6113
Model 4 (Transfer Learning)	0.8119	0.8012	0.9375	0.8651	0.5118

Table 9.1: Table shows the values obtained for measures of different model

The graph is plotted for accuracy and loss shown by each model. The accuracy v/s epoch graph has accuracy values on y-axis and epoch values on x-axis. Similarly, for loss v/s epoch graph, loss is plotted on y-axis and epoch on x-axis. It provides an overview of the model behavior for each epoch run on the model and their accuracy rate. The orange line represents the result of the training set and blue line represents the result of validation set.

Graph of ReLU Activation Function

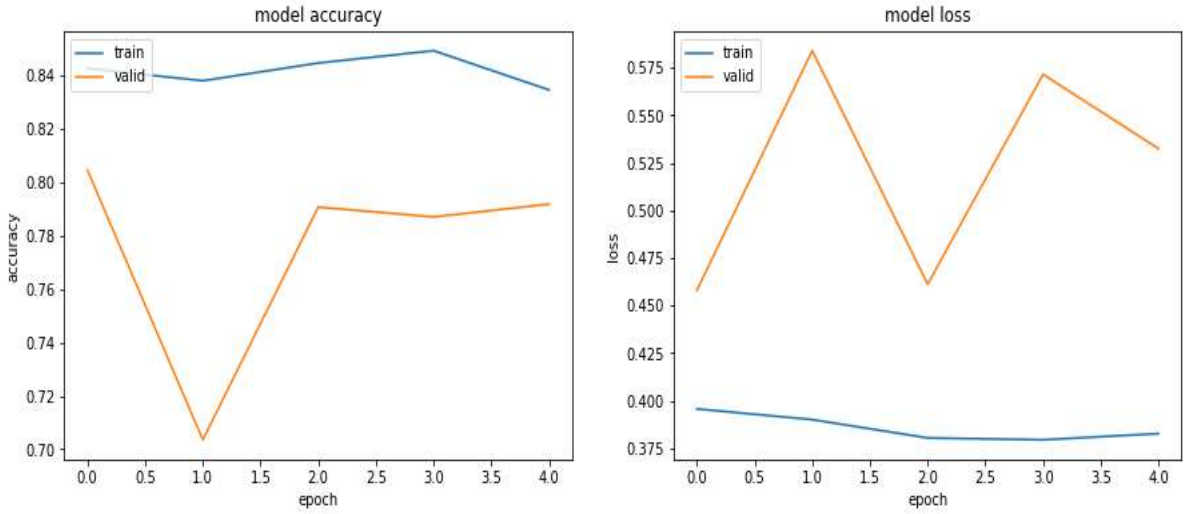

Figure 9.1.1: Accuracy and Loss graph of ReLU Activation Function

As shown in the above figure 9.1.1 we can observe that the training accuracy is maintained almost constant between 0.83 and 0.85 with a slight variation. The validation accuracy first decreases and later rises and remains constant throughout. In training loss, we can see a constant decrease at 0.375 and the validation loss increase and decrease but it is less than 0.575 which is a good sign. Overall, when compared to other activation functions this provided best results.

Graph of ELU Activation Function

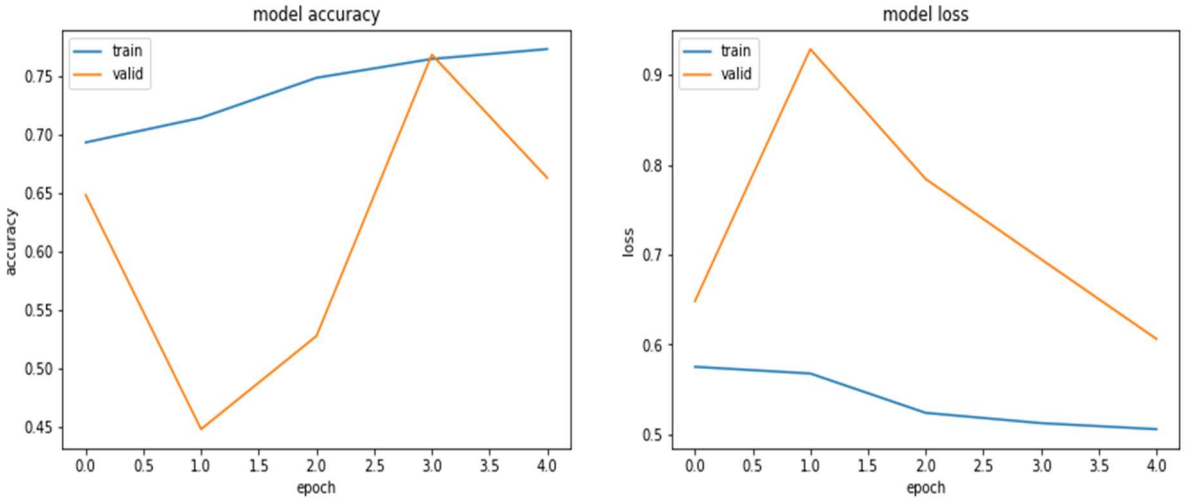

Figure 9.1.2: Accuracy and Loss graph of ELU Activation Function

As shown in the above figure 9.1.2 we can observe that the training accuracy increases steadily and validation accuracy has a variation of increase and decrease. The loss in training accuracy is on a decreasing trend and about validation loss it shows a drastic change of increase and then decrease.

Graph of Tanh Activation Function

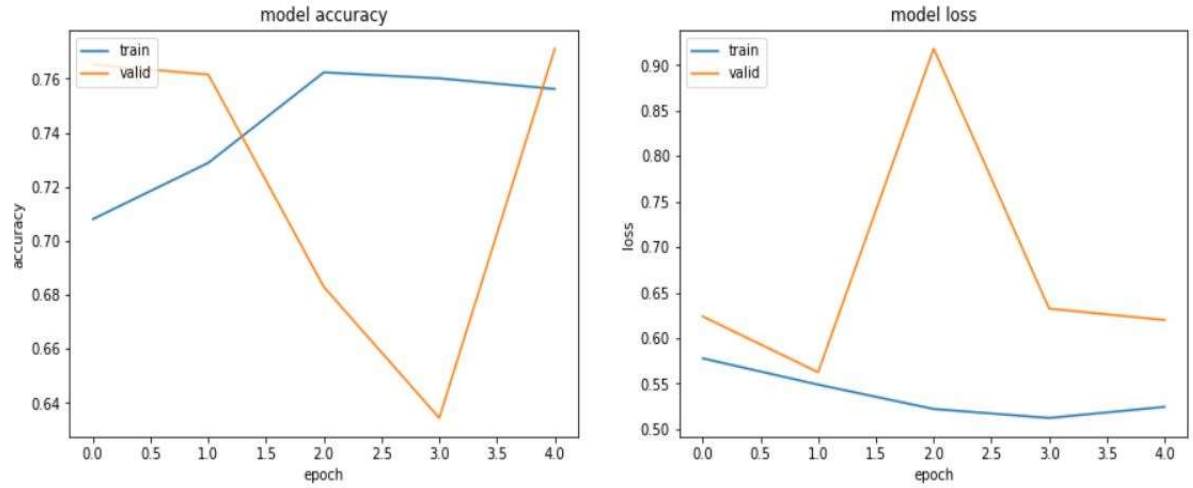

Figure 9.1.3: Accuracy and Loss graph of Tanh Activation Function

As shown in the figure 9.1.3 the training accuracy increases upto a point and becomes constant as the validation accuracy has a radical decrease and increase is observed. The training loss decreases upto 0.50 while the validation loss increases to a peak of 0.90 and later decreases.

Graph of Transfer Learning Model

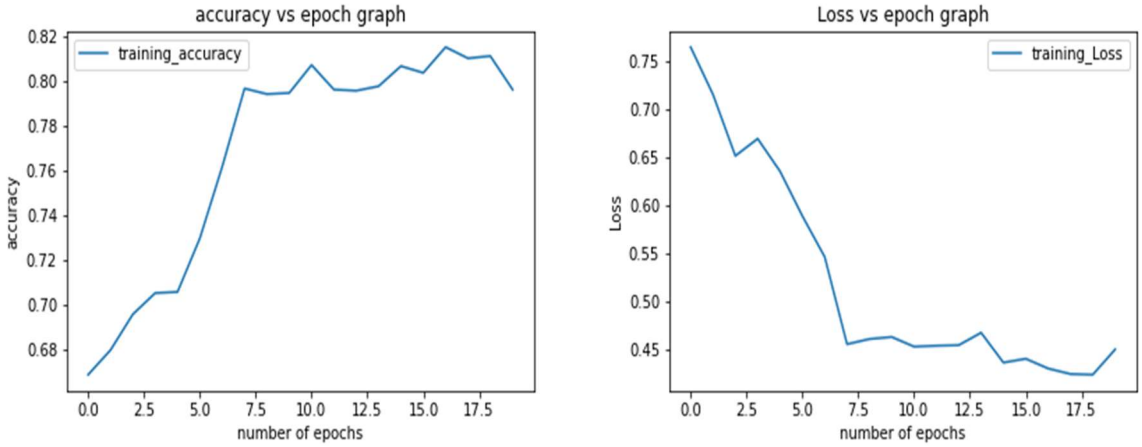

Figure 9.1.4: Training Accuracy and Loss graph of VGG-16 Model

As shown in the above figure 9.1.4 the training accuracy increases and is maintained at a constant level while the training loss also decreases as the epochs increase.

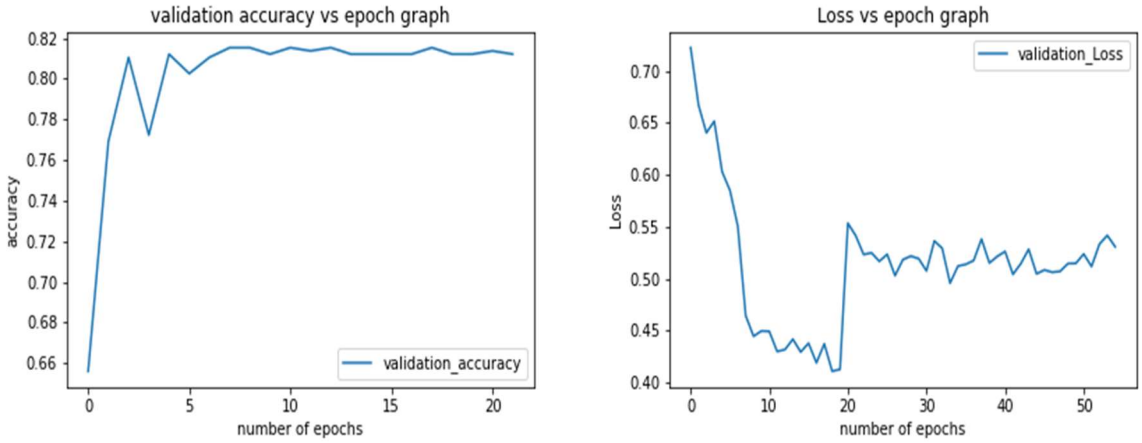

Figure 9.1.5: Validation Accuracy and Loss graph of VGG-16 Model

The figure 9.1.5 shows the validation accuracy where it increases and maintains a constant level at 0.81. Whereas, the validation loss decreases and has a slight increase later.

Based on the above evaluations, it is obvious that every model has its own merits and flaws. Even though no model could capture the complex features to detect the heartbeats as normal or abnormal. The validation accuracy of 80% and training accuracy of 88% could be considered reasonable. It came close enough to validate the approach that we took while building the model. It's likely that additional experimentation fine tuning the model and adjusting and tweaking the hyperparameters could lead to even better results.

CONCLUSION

It is a very well-known fact that Deep Convolutional Networks outperform every algorithm when it comes to computer vision and image classification tasks. However, as this was a non-trivial problem, we were not easily convinced that a single network would be able to classify the recordings with great accuracy. The results confirm that getting a better accuracy and more efficient solution is possible with few tweaks as mentioned in the improvements section. The approach for an end to end solution was to convert the recordings to spectrogram images and then train a deep convolutional network. We progressively implemented different architectures to classify the images and obtain a better result. We also applied various data augmentation techniques. Initially, we started with a simple CNN and then shifted to one with more layers. To further refine and improve results, we implemented a technique known as Transfer Learning. We used the pre-trained VGG16 model and extracted features from images and then ran the classifier over those features. We also trained the images on a custom model based on VGG16 model architecture from scratch to improve results.

The most difficult part was to extract the spectrogram images from the recordings and train the network over them on the local machine. Another challenging aspect of the project was to get a high recall and accuracy with lack of training samples and recordings filled with noise. It was very hard to achieve great results for this type of classification. Based on the observations made and results obtained we got an accuracy of 80.11% from ReLU network of Keras model which is the highest among them and an accuracy of 81.56% from transfer learning model implementing the VGG-16 architecture but when compared with loss VGG-16 performed well than the ReLU network.

FUTURE ENHANCEMENT

One of the simplest ways to improve upon the existing model is to use more data. Due to non-availability of data, high computational cost, time and memory constraints we were only able to use a small subset of the complete dataset consisting of 2000 samples for training and 700 for validation. This can be further overcome by training in the cloud platform. Another way that will be effective in improving the model would be to use a different model architecture such as InceptionV3 or RESNET.

The dataset also had a lot of noise since the heart recordings consisted of people talking and breathing heavily and due to high computational cost, it was not possible to eliminate them. To accurately separate signal from noise, the original signal can be filtered with a high pass Butterworth filter to remove noise above 4Hz or 240 beats per minute. The filtered signal then could be transformed to it's approximate frequency domain. Given the way that our model reached a relatively high accuracy rate and then plateaued on both the training and validation sets suggests that we may have hit a limit on the complexity of features our model was able to encode. We could attempt to address this by adding additional convolution layers expanding the number of filters in existing layers or even trying some novel new architectures. Microsoft’s winning ImageNet model, uses residual layers to make features detected at lower layers directly available to higher levels.

REFERENCES

[1] Weifang Sun, Nianyin Zeng and Yuchao He, ***"Morphological Arrhythmia Automated Diagnosis Method Using Gray-Level Co-Occurrence Matrix Enhanced Convolutional Neural Network"*** in IEEE Access, February 2019.

[2] V. Sai Krishna, A. Nithya Kalyani, ***"Prediction of Cardiac Arrhythmia using Artificial Neural Network"*** in International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8, Issue-1S4, June 2019.

[3] Carlos S. Lima, Manuel J. Cardoso, ***"Cardiac Arrhythmia Detection by Parameters Sharing and MMIE Training of Hidden Markov Models"*** in 29th Annual International Conference of the IEEE EMB Cité Internationale, Lyon, France, August 23-26, 2007.

[4] Elif Izci, Mehmet Akif Ozdemir, Reza Sadighzadeh, Aydin Akan, ***"Arrhythmia Detection on ECG Signals by Using Empirical Mode Decomposition"*** in Proceedings of the third international workshop on advanced issues of e-commerce and web-based information systems, IEEE Proceedings, June 2018.

[5] Ali Isina, Selen Ozdalilib, ***"Cardiac arrhythmia detection using deep learning"*** in 9th International Conference on Theory and Application of Soft Computing, Computing with Words and Perception, ICSCCW 2017, 24-25 August 2017, Budapest, Hungary.

[6] Thara Soman, Patrick O Bobbie, ***"Classification of Arrhythmia Using Machine Learning Techniques"*** in International journal of man-machine studies, October 2016,

[7] R Karthik, Dhruv Tyagi, Amogh Raut, Soumya Saxena, Rajesh Kumar M, ***"Implementation of Neural Network and feature extraction to classify ECG signals"***, EP Europace, Volume 21, Issue 8, August 2019.

[8] B. S. Chandra, C. S. Sastry and S. Jana, ***"Robust Heartbeat Detection from Multimodal Data via CNN-based Generalizable Information Fusion"*** in Journal of the American Society for Information Science and Technology, 29 June 2019

[9] Anup Das, Francky Catthoor, Siebren Schaafsma, ***"A rule-based method to model myocardial fiber orientation in cardiac biventricular geometries with outflow tracts"*** in Communications of the ACM, 13 August 2019

[10] Milad Salem, Shayan Taheri, Jiann-Shiun Yuan, ***"ECG Arrhythmia Classification Using Transfer Learning from 2-Dimensional Deep CNN Features"***, in Communications of the ACM, 21 May 2017.

[11] Sajad Mousavi, Fatemeh Afghah, ***"Inter- and Intra-Patient ECG Heartbeat Classification for Arrhythmia Detection: A Sequence to Sequence Deep Learning Approach"***, ACM arXiv:1812.07421v2, 12 March 2019.

[12] Asim Darwaish, Farid Naït-Abdesselam, ***"Detection and Prediction of Cardiac Anomalies using Wireless Body Sensors and Bayesian Belief Network",*** arXiv:1904.07976v1, ACM proceedings, 16 April 2019.

[13] Rajpurkar, Pranav & Hannun, Awni & Haghpanahi, Masoumeh & Bourn, Codie & Y. Ng, Andrew. ***"Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks"***, arXiv:1707.01836v1 [cs.CV], ACM proceedings, 6 July 2017.

[14] Kiranyaz S, Ince T, Gabbouj M, ***"Real-time patient-specific ECG classification by 1-D convolutional neural networks"*** IEEE Transactions on Biomedical Engineering, Volume 63, pp. 664-675, June 2016.

[15] S. Hong and M. Wu and Y. Zhou and Q. Wang and J. Shang and H. Li and J. Xie, ***"ENCASE: An Ensemble Classifier for ECG classification using expert features and deep neural networks"*** Computing in Cardiology, Rennes pp. 1-4, October 2017.

Web References

[16] https://physionet.org/challenge/2016/: Aims to encourage the development of algorithms to classify heart sound recordings.

[17] https://www.kaggle.com/kinguistics/heartbeat-sounds :Classifying heartbeat anomalies from stethoscope audio

[18]https://www.heart.org/HEARTORG/Conditions/Arrhythmia/AboutArrhythmia/AboutArrh%20ythmia_UCM_002010_Article.jsp#.WTeSZROGMU : symptoms, diagnosis, monitoring, prevention and treatment of Arrhythmia are detailed here.

[19] https://keras.io/api/layers/activations/ :Describing keras activation functions.

[20] https://www.physionet.org/challenge/2016/papers/challenge2016.pdf :Datasets required for classification are available here.