PROJECT REPORT

CLASSIFICATION OF ARRHYTHMIA BY USING DEEP LEARNING WITH 2D ECG SPECTRAL IMAGE REPRESENTATION

TEAM ID:PNT2022TMID34730

INTRODUCTION

Project Overview

This project is the design and implementation of the detection of arrhythmia detection using deep learning models Electrocardiogram (ECG) is a simple non-invasive measure to identify heart-related issues such as irregular heartbeats known as arrhythmias Deep CNNbased algorithm is implemented for training the model, artificial intelligence and machine learning are being utilized in a wide range of healthcare-related applications and datasets, many arrhythmia classifiers using deep learning methods have been proposed in recent years. However, the sizes of the available datasets from which to build and assess machine learning models are often very small and the lack of well-annotated public ECG datasets is evident. In this paper, we propose a deep transfer learning framework that is aimed toto perform classification on a small. The proposed method is to fine-tune general-purpose image classifier ResNet-18 with the MIT-BIH arrhythmia dataset in an accurate MIT-BIH arrhythmia dataset per further investigation many existing deep learning modes failed to avoid data leakage against AAMI recommendations.

Purpose

The purpose of the project is to design and implement of deep learning model deployed for the detection of heart disease and the prediction.

LITERATURE SURVEY

The electrocardiogram (ECG) is one of the most extensively employed signals used in the diagnosis and prediction of cardiovascular diseases (CVDs). The ECG signals can capture the heart's rhythmic irregularities, commonly known as arrhythmias. A careful study of ECG signals is crucial for precise diagnoses of patients' acute and chronic heart conditions. Cardiovascular diseases (CVDs) are the leading cause of human death, with over 17 million people known to lose their lives annually due to CVDs.

According to the World Heart Federation, three-fourths of the total CVD deaths are among the middle and low-income segments of the society. A classification model to identify CVDs at their early stage could effectively reduce the mortality rate by providing a timely treatment . One of the common sources of CVDs is cardiac arrhythmia, where heartbeats are known to deviate from their regular beating pattern. A normal heartbeat varies with age, body size, activity, and emotions. In cases where the heartbeat feels too fast or slow, the condition is known as palpitations. An arrhythmia does not necessarily mean that the heart is beating too fast or slow, it indicates that the heart is following an irregular beating pattern. It could mean that the heart is beating too fast—tachycardia (more than 100 beats per minute (bpm)), or slow—bradycardia(less than 60 bpm), skipping a beat, or in extreme cases, cardiac arrest. Some other common types of abnormal heart rhythms include atrialfibrillation, atrial flutter, and ventricular fibrillation. These deviations could be classified into various subclasses and represent different types of cardiac arrhythmia. An accurate classification of these types could help in diagnosing and treatment of heart disease patients. Arrhythmia could either mean a slow or fast beating of heart, or patterns that are not attributed to a normal heartbeat. An automated detection of such patterns is of great significance in clinical practice. There are certain known characteristics of cardiac arrhythmia, where the detection requires expert clinical knowledge. The electrocardiogram (ECG) recordings are widely used for diagnosing and predicting cardiac arrhythmia for diagnosing heart diseases. Towards this end, clinical experts might need to look at ECG recordings over a longer period of time for detecting cardiac arrhythmia. The ECG is a one-dimensional(1-D) signal representing a time series, which can be analyzed using machine learning techniques for automated detection of certain abnormalities. Recently, deep learning techniques have been developed, which provide significant performance in radiological image analysis.

EXISTING METHODS

Convolutional neural networks (CNNs) have recently been shown to work for multi-dimensional (1-D, 2-D, and in certain cases, 3-D) inputs but were initially developed for problems dealing with images represented as two-dimensional inputs. For time series data, 1-D CNNs are proposed but are less versatile when compared to 2-D CNNs. Hence, representing the time series data in a 2-D format could benefit certain machine learning tasks . Hence, for ECG signals, a 2-D transformation has to be applied to make the time series suitable for deep learning methods that require 2-D images as input. The short-time Fourier transform (STFT) can convert a 1-D signal into a 2-D spectrogram and encapsulate the time and frequency information within a single matrix. The 2-D spectrogram is similar to hyper-spectral and multi-spectral images (MSI), which have diverse applications in remote sensing and clinical diagnosis, including spectral un-mixing, ground cover classification and matching, mineral exploration, medical image classification, change detection, synthetic material identification, target detection, activity recognition, and surveillance . The 2-D matrix of spectrogram coefficients could be useful for extracting robust features for representation of acardiac ECG signal.

This representation could allow the application of CNN architectures (designed to operate on 2-D inputs) for development of automated systems related to CVDs.1.1. Related Works. The ECG signal detects abnormal conditions and malfunctions by recording the potential bio-electric variation of the human heart. Accurately detecting the clinical condition presented by an ECG signal is a challenging task. Therefore, cardiologists need to accurately predict and identify the right kind of abnormal heartbeat ECG wave before recommending a particular treatment. This might require observing and analyzing ECG recordings that might continue for hours (patients in critical care). To overcome this challenge for the visual and physical explanation of the ECG signal, computer-aided diagnostic systems have been developed to automatically identify such signals automatically. Most of the research in this field has been conducted by incorporating different approaches of machine learning (ML) techniques for the efficient identification and accurate examination of ECG signals. ECG signal classification based on different approaches has been presented in the literature including frequency analysis , artificial neural networks (ANNs) [22],heuristic-based methods, statistical methods , support vector machines (SVMs) , wavelet transform , filter banks hidden Markov models , and mixture-of-expert methods. An artificial neural network based method obtained an average accuracy of 90.6% for the classification of ECG wave into six classes. Meanwhile, a feed-forward neural network was used as a classifier for the detection of four types of arrhythmia classes and achieved an average accuracy of 96.95% .

DEEP LEARNING

Machine learning is a subset of artificial intelligence used with high-end diagnostic tools for the prediction and diagnosis of different types of illnesses. Deep learning, as a subset of ML, has many applications in the prediction and prevention of fatal sicknesses, particularly CVDs. Different techniques of deep learning used for the analysis of bioinformatics signals have been presented in . A recurrent neural network (RNN) was used for feature extraction and achieved an average accuracy of 98.06% for detecting four types of arrhythmia. For the classification and extraction of features from a 1-D ECG signal, a 1-D convolutional neural network model was proposed and yielded a classification accuracy of 96.72%. Another deeper 1-D CNN model was proposed for the classification of the ECG dataset and obtained an average accuracy of 97.03%. In both instances, a large ECG dataset was used, but the ECG signals were represented as a 1-D time series. A nine-layer2-D CNN model was applied for an automatic classification of five different heartbeat arrhythmia types achieving an accuracy of 94.03%. Deep Learning (DL) has recently become a topic of study in different applications including healthcare, in which timely detection of anomalies on Electrocardiogram (ECG) can play a vital role in patient monitoring.

PROPOSED SOLUTIONS

A previously proposed paper presents a comprehensive review study on the recent DL methods applied to the ECG signal for the classification purposes. This study considers various types of the DL methods such as Convolutional Neural Network (CNN), Deep Belief Network (DBN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU). From the 75 studies reported within 2017 and 2018, CNN is dominantly observed as the suitable technique for feature extraction, seen in 52% of the studies. DL methods showed high accuracy in correct classification of Atrial Fibrillation (AF) (100%), Supra ventricular Ectopic Beats (SVEB) (99.8%), and Ventricular Ectopic Beats (VEB) (99.7%) using the GRU/LSTM, CNN, and LSTM, respectively. High-risk patients of cardiovascular disease can be provided with computerized electrocardiogram (ECG) devices to detect Arrhythmia. These require long segments of quality ECG which however can lead to missing the episode. To overcome this, it has been proposed a deep-learning approach, where the scalogram obtained by continuous wavelet transform (CWT) is classified by the network based on the signature corresponding to arrhythmia. The CWT of the recordings is obtained and used to train the 2D convolutional neural network (CNN) for automatic arrhythmia detection. Here, the proposed model is trained and tested to identify five types of heartbeats such as normal, left bundle branch block, right bundle branch block, atrial premature, and premature ventricular contraction. Themodel shows an average sensitivity, specificity, and accuracy to be 98.87%, 99.85%, and 99.65%, respectively. The result shows that the proposed model can detect arrhythmia effectively from short segments of ECG and has the potential for being used for personalised and digital healthcare. The early diagnosis of cardiovascular infection is focused on exploration and distinction signs of arrhythmia. Throughout this analysis it is proposed the interaction between CNN-LSTM and FCL to improve the preparedness influence, limiting the effects on the model training of an enormous amount of basic specific ECG beat information. The proposed architecture utilizes CNNs to decrease each spectral variation in the input feature but instead moves it on to LSTM layers while providing outputs to DNN layers, which have a more effective feature representation. The findings indicate that CNN-LSTM and FCL have obtained 99.33%, 96.06%, 94.36%, and 92.65%, individually, with the results being accuracy, F1 score, precision, and recall. The adequacy and intensity of the proposed architecture were seen by the MIT-BIH arrhythmic test results. The methodology proposed could be used to help cardiologists in diagnosing ECGs with a better level of accuracy and impartiality in telemedicine scenarios. In future examinations, various kinds and specific beats will be included. In addition, to analyze the appearance of the CNN LSTM using the FCL pattern, it is intend also to introduce specific rates of noise to ECG signals. The ECG image from ECG signal is processed by some image processing techniques.

To optimize the identification and categorization process, this research presents a hybrid deep learning-based technique. This paper contributes in two ways. Automating noise reduction and extraction of features, 1D ECG data are first converted into 2D pictures. Then, based on experimental evidence, a hybrid model called CNNLSTM is presented, which combines CNN and LSTM models. A comprehensive research has been conducted using the broadly used MIT_BIH arrhythmia dataset to assess the efficacy of the proposed CNN-LSTM technique. The results reveal that the proposed method has a 99.10 percent accuracy rate. Furthermore, the proposed model has an average sensitivity of 98.35 percent and a specificity of 98.38 percent. These outcomes are superior to those produced using other procedures, and they will significantly reduce the amount of involvement necessary by physicians. Arrhythmia categorization is the most important topic in medicine. The heart rate irregularity is known as an arrhythmia. This study developed an approach for computerized cardiac arrhythmia monitoring using the CNN-LSTM model. This technique employs convolutional neural network for feature engineering and LSTM for categorization, and it uses the CWT to transform 1D ECG signals into 2D ECG image plots, making them a suitable raw input for this network. Investigations on three ECG cross data bases showed that they can outperform other classification methods whenused correctly.The MIT–BIH arrhythmia database information was divided into pathologic and normal categories depending on the ECG beat types shown in it. The confusion matrix for the testing dataset revealed that “regular sinus rhythm” had 99 percent validation accuracy, “cardiac arrhythmias” had 98.7% validation accuracy, and “congestive heart attacks” had 99 percent validation accuracy. Furthermore, ARR has 0.98 percent sensitivity and 0.98 percent specificity, while CHF has 0.96 percent sensitivity and 0.99 percent specificity, and NSR has 0.97 percent sensitivity and 0.99 percent specificity. Our methodology beats earlier methods in terms of overall efficiency. Furthermore, CWT's large computational load is a negative. Although it would considerably reduce the amount of intervention required by physicians, it could not ever achieve a comprehensive inter subject state. It would be an excellent next research topic. To address these challenges, a reliable arrhythmia classification system is required.

In another study, it proposes a two-dimensional (2-D) convolutional neural network (CNN) model for the classification of ECG signals into eight classes; namely, normal beat, premature ventricular contraction beat, paced beat, right bundle branch block beat, left bundle branch block beat, atrial premature contraction beat, ventricular flutter wave beat, and ventricular escape beat. The one-dimensional ECG time series signals are transformed into 2-D spectrograms through short-time Fourier transform. The 2-D CNN model consisting of four convolutional layers and four pooling layers is designed for extracting robust features from the input spectrograms. The proposed methodology is evaluated on a publicly available MIT-BIH arrhythmia dataset. They achieved a state-of-the art average classification accuracy of 99.11%, which is better than those of recently reported results in classifying similar types of arrhythmias. The performance is significant in other indices as well, including sensitivity and specificity, which indicates the success of the proposed method to some extent. Although there are many previously proposed models, the accuracy rate still needs an improvement. A solution for this problem needs to be devised ignorer to accurately find the type and extent of the abnormality.

REFERENCES

1. Sorriento D., Iaccarino G. Inflammation and cardiovascular diseases: the most recent findings. International Journal of Molecular Sciences. 2019;20(16, article 3879) doi:10.3390/ijms20163879. [PMC free article] [PubMed] [CrossRef] [Google Scholar]

2. Sangeetha D., Selvi S., Ram M. S. A. A CNN based similarity learning for cardiac arrhythmia prediction. 2019 11th International Conference on Advanced Computing (ICoAC); December 2019; Chennai, India. pp. 244–248. [CrossRef] [Google Scholar]

3. Almalchy M. T., Ciobanu V., Popescu N. Noise removal from ECG signal based on filtering techniques. 2019 22nd International Conference on Control Systems and Computer Science (CSCS); May 2019; Bucharest, Romania. pp. 176–181. [CrossRef] [Google Scholar]

4. Yao Q., Wang R., Fan X., Liu J., Li Y. Multi-class arrhythmia detection from 12-lead varied- length ECG using attention-based time-incremental convolutional neural network. Information Fusion . 2020;53:174–182. Doi: 10.1016/j.inffus.2019.06.024. [CrossRef] [Google Scholar]

5. Ince T., Kiranyaz S., Gabbouj M. A generic and robust system for automated patient-specific classification of ECG signals. IEEE Transactions on Biomedical Engineering. 2009;56(5):1415–1426. doi: 10.1109/TBME.2009.2013934. [PubMed] [CrossRef] [Google Scholar]

6. Li H., Yuan D., Wang Y., Cui D., Cao L. Arrhythmia classification based on multi-domain feature extraction for an ECG recognition system. Sensors. 2016;16(10): p. 1744. doi: 10.3390/s16101744. [PMC free article] [PubMed] [CrossRef] [Google Scholar]

7. Mc Namara, K.; Alzubaidi, H.; Jackson, J.K. Cardiovascular disease as a leading cause of death: How are pharmacists getting involved? Integr. Pharm. Res. Pract. 2019, 8, 1. [CrossRef] [PubMed]

8. Lackland, D.T.; Weber, S.M.A. Global burden of cardiovascular disease and stroke: hypertension at the core. Can. J. Cardiol. 2015, 31, 569–571. [CrossRef] [PubMed]

9. Mustaqeem, A.; Anwar, S.M.; Majid, M. A modular cluster based collaborative

recommendersystem for cardiac patients. Artif. Intell. Med. 2020, 102, 101761. [CrossRef] [PubMed]

10. Irmakci, I.; Anwar, S.M.; Torigian, D.A.; Bagci, U. Deep Learning for Musculoskeletal Image Analysis. arXiv 2020, arXiv:2003.00541.

11. Anwar, S.M.; Majid, M.; Qayyum, A.; Awais, M.; Alnowami, M.; Khan, M.K. Medical image analysis using convolutional neural networks: A review. J. Med. Syst. 2018, 42, 226. [CrossRef]

12. Gu, J.; Wang, Z.; Kuen, J.; Ma, L.; Shahroudy, A.; Shuai, B.; Liu, T.; Wang, X.; Wang, G.; Cai, J.; et al. Recent advances in convolutional neural networks. Pattern Recognit. 2018, 77, 354–377. [CrossRef]

13. Wu, Y.; Yang, F.; Liu, Y.; Zha, X.; Yuan, S. A comparison of 1-D and 2-D deep convolutional neural networks in ECG classification. arXiv 2018, arXiv:1810.07088.

14.https://www.researchgate.net/publication/341623436_Classification_of_Arrhythmia_by_Using_Deep_earning_with_2-D_ECG_Spectral_Image_Representation

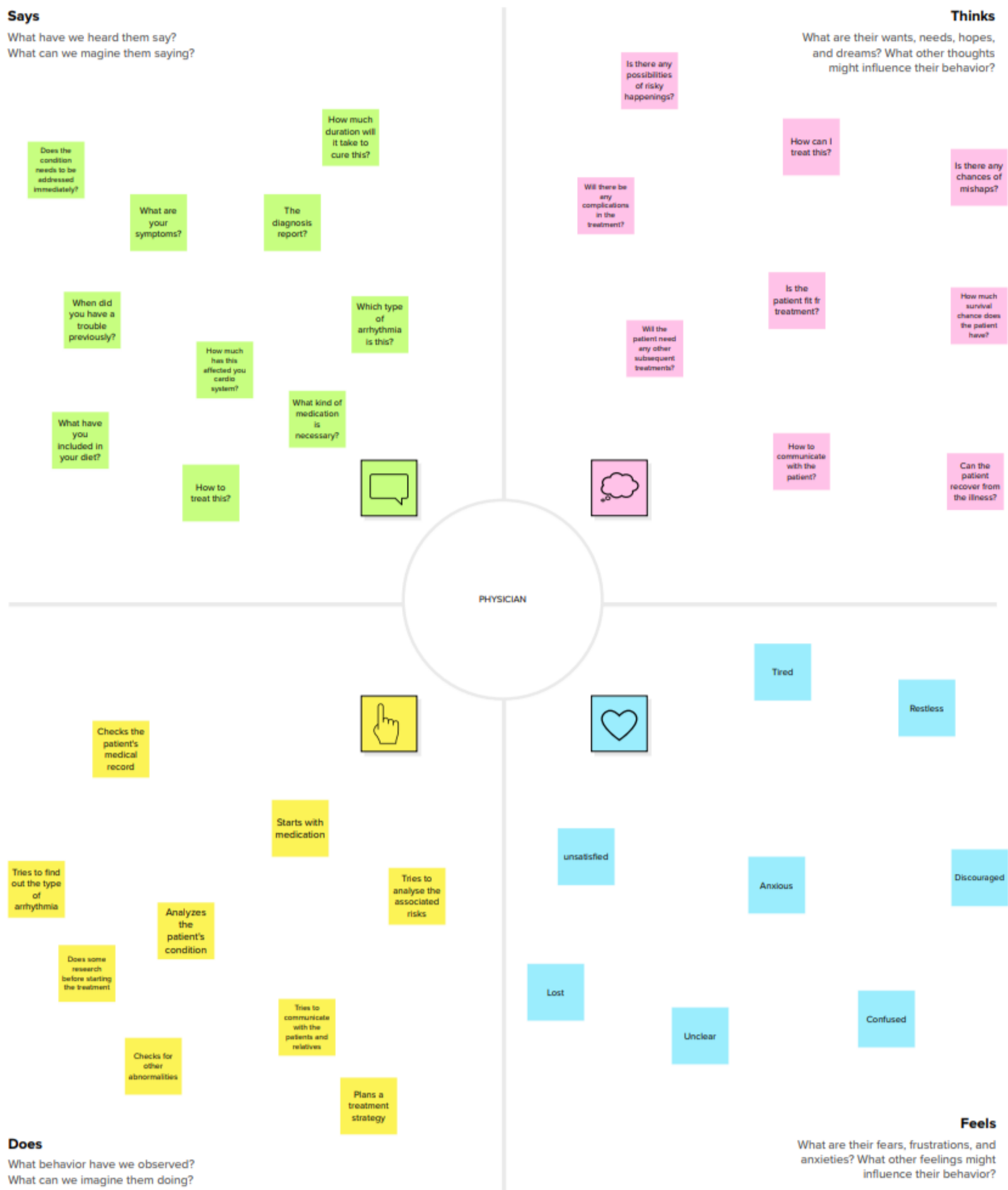
15. http://doi.org/10.11591/ijeecs.v25.i2.pp931-940

16. https://www.sciencedirect.com/science/article/pii/S2590188520300123

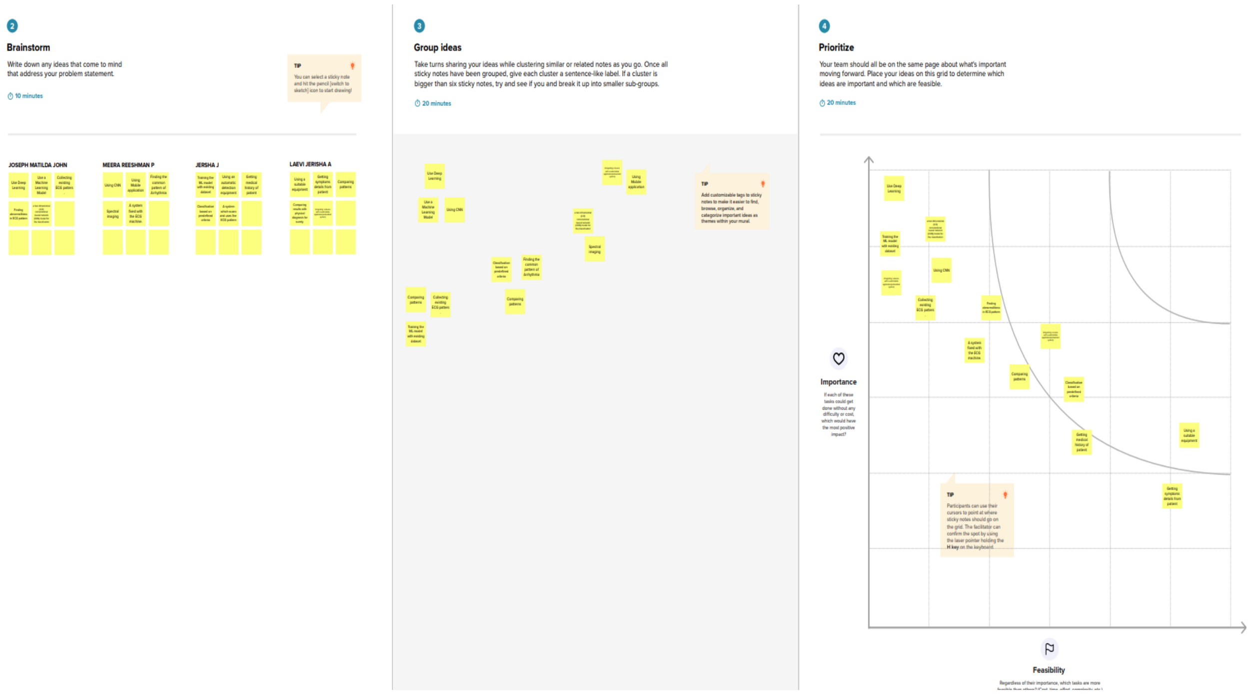
17. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4088025

18. https://www.techscience.com/iasc/v31n2/44530/html

EMPATHYMAP


Says
What have we heard them say?
What can we magine them saying?
Does the condition needs to be addressed immediately?
How much duration will it take to cure this?
What are your symptoms?
The diagnosis report?
When did you have a trouble previously?
Which type of arrhythmia is this?
How much has this affected you cardio system?
What kind of medication is necessary?
What have you included in your diet?
How to treat this?
Thinks
What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?
Is there any possibilities of risky happenings?
How can I treat this?
Is there any chances of mishaps?
Will there be any complications in the treatment?
Is the patient fit fr treatment?
How much survival chance does the patient have?
Will the patient need any other subsequent treatments?
How to communicate with the patient?
Can the patient recover from the illness?
PHYSICIAN
Checks the patient's medical record
Starts with medication
Tries to find out the type of arrhythmia
Tries to analyse the associated risks
Analyzes the patient's condition
Does some research before starting the treatment
Tries to communicate with the patients and relatives
Checks for other abnormalities
Plans a treatment strategy
Does
What behavior have we observed?
What can we imagine them doing?
Tired
Restless
unsatisfied
Anxious
Discouraged
Lost
Unclear
Confused
Feels
What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

BRAINSTOMING



Proposed Solutions

S.No.	Parameter	Description
1	Problem Statement (Problem to be solved)	To classify the given ECG pattern on the basis of different types of Arrhythmia.
2	Idea / Solution description	We use a web application through which the user selects the image which is to be classified into the arrhythmia type by an effective electrocardiogram (ECG) arrhythmia classification method. It can be done by using a convolutional neural network (CNN), in which we classify ECG into seven categories, one being normal and the other six being different types of arrhythmia using deep two-dimensional CNN with grayscale ECG images The image is fed into the model that is trained and the cited class will be displayed on the webpage.
3	Novelty / Uniqueness	The proposed solution considers other physical abnormalities which contribute to the disease thereby helps in exact classification of arrhythmia and to make people awareness on their general health.
4	Social Impact / Customer Satisfaction	Easy and a quick method of classification of arrhythmia which replaces the traditional method, thereby allowing a one click solution to its users.
5	Business Model (Revenue Model)	● Can collaborate with diagnosis centres and hospitals. ● Can collaborate with government for health awareness camps.
6	Scalability of the Solution	Can be used by any individual throughout the world who has a minimal knowledge of web application usage.

Problem Solution Fit

1. CUSTOMER SEGMENT(S)

CS

Patients with Arrhythmia

6. CUSTOMER CONSTRAINTS

Spending time, budget, no cash, transport issues, incorrect diagnosis and takes time for result analysis.

5. AVAILABLE SOLUTIONS

Many ECG apps which can record your heartbeat and rhythm using the electrical heart sensor and then check for signs of atrial fibrillation (AFib), a form of irregular rhythm. They have tried to classify arrhythmia with existing ECG patterns but have some drawbacks like inability to classify with high precision due to lack of optimization.

2. JOBS-TO-BE-DONE / PROBLEMS

Customers find it difficult to diagnose their disease at diagnostic centers. Web applications available for this purpose are not trustworthy due to its inaccuracies.

9. PROBLEM ROOT CAUSE

RC

Lack of adequate data to train the models and and unavailability of accurate data set. There is alo a lack of generalized data available for the disease.

7. BEHAVIOUR

BE

3. TRIGGERS

TR

What triggers customers to act?

Customers finding out that there are new and sophisticated technologies being used in different developed countries, and that the time for taking action to cure Arrhythmia is comparatively less. Using Web Applications for diagnosis is rapidly increasing among the common people.

10. YOUR SOLUTION

SL

Using an automated diagnosis model to classify the fed ECG pattern with arrhythmia into an appropriate group. This includes a Convolutional Neural Network and enhancing the accuracy by introducing many such patterns for training the AI model. This ensures easy recognition of the disease even when the given pattern has a minimal trace of Arrhythmia in its pattern.

8. CHANNELS of BEHAVIOR

CH

8.1 ONLINE

Online consultation.

8.2 OFFLINE

Visiting Laboratories to test and give it for analysis to classify the type of Arrhythmia.

4. EMOTIONS: BEFORE / AFTER

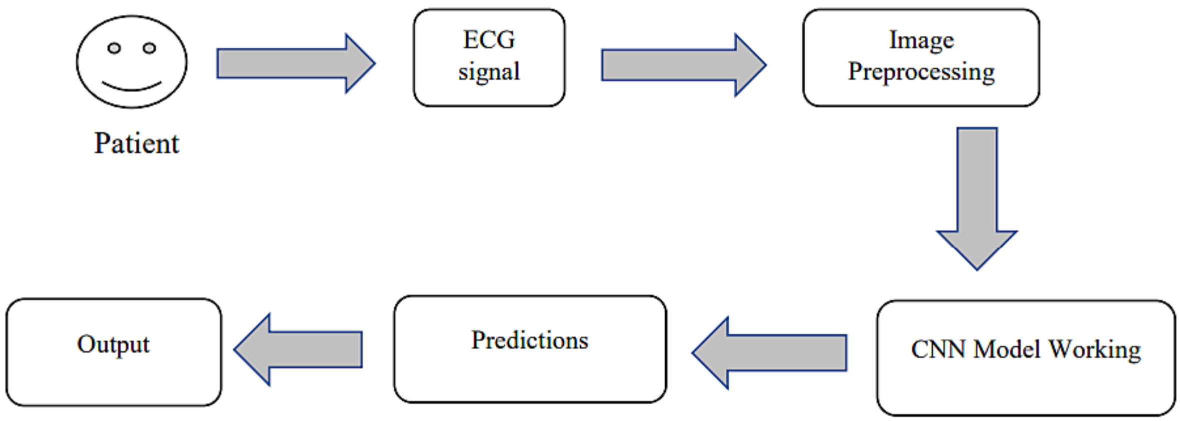
EM

How do customers feel when they face a problem or a job and afterwards?

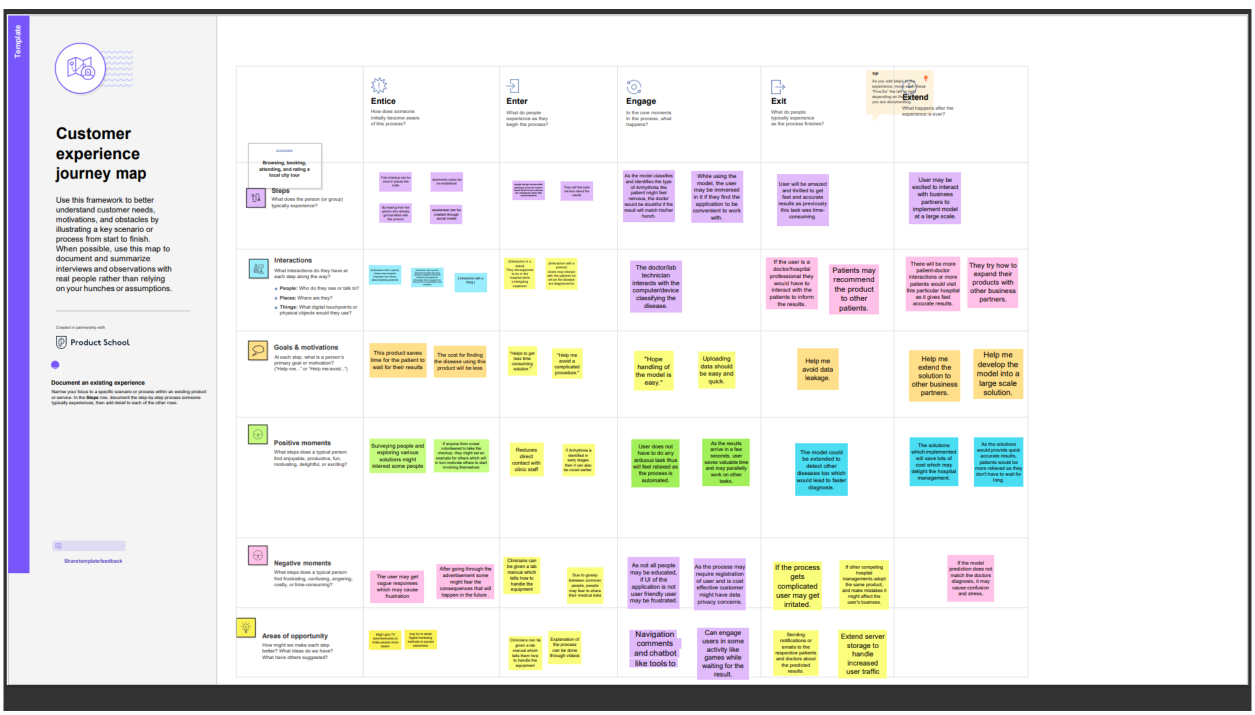
Customers feel lost, scared due to their disease's effects, have their own insecurities and are affected by anxiety after giving samples for the required diagnosis.

They will feel more confident, be aware of their abnormality and be more confident.

Solution Architecture



CUSTOMER JOURNEY



SOLUTION REQUIREMENT

FUNCTIONAL REQUIREMENT

Following are the functional requirements of the proposed solution

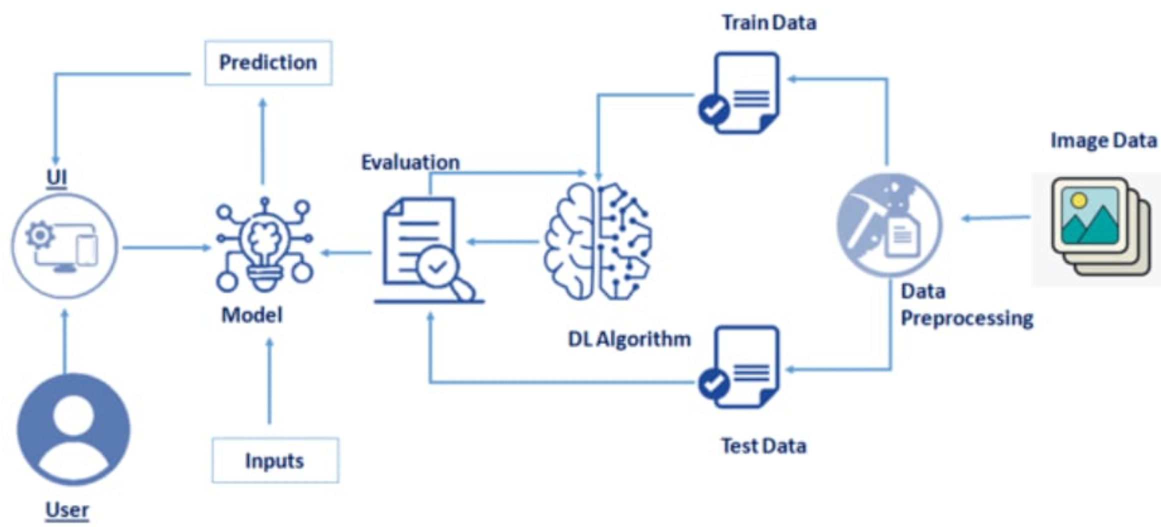
FR No.	**Functional Requirement (Epic)**	**Sub Requirement (Story / Sub-Task)**
FR-1	Artificial Intelligence model	scanning the ECG graph Comparing it with the trained data
FR-2	Web Application	Upload the ECG pattern View the results and information Creting the UI
FR-3	Training the model	Data to train the model
FR-4	Backend using Python	building the GET, POST methods

NON FUNCTIONAL REQUIREMENT

Following are the non-functional requirements of the proposed solution

FR No.	**Non-Functional Requirement** **Description**
NFR-1	**Usability**
NFR-2	**Security**
NFR-3	**Reliability**
NFR-4	**Performance**
NFR-5	**Availability**
NFR-6	**Scalability** **Scalability**

DATA FLOW DIAGRAM



USER STORIES

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Dashboard	USN-1	User can upload image of the ECG graph.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive the diagnosis as to whether I have Arrhythmia or not	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I receive the type of arrhythmia	I can register & access the dashboard with Facebook Login	High	Sprint-2
		USN-4	As a user, I can receive the suggested remedy		Medium	Sprint-1

TECHNOLOGY STACK

Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	Web UI	HTML, CSS,Python.
2.	Application Logic-1	Data Preprocessing	Keras, Tensorflow, Numpy - (Importing Essentisl Libraries)
3.	Application Logic-2	CNN Model Creating	Keras, Tensorflow, Numpy - (Importing Essentisl Libraries)
4.	Application Logic-3	Web Application (UI)	Flask
5.	Database	Images (Jpeg, PNG, Jpg, etc..)	Uploads Folder
6.	File Storage	File storage requirements (only if necessary)	IBM Block Storage / Google Drive (Depends On Preference)
7.	External API-1	Keras	Image Processing API.
8.	Deep Learning Model	Inception v3 architecture	Object Recognition Model, etc.
9.	Infrastructure (Server / Cloud)	Application Deployment on web server	Flask—a Python WSGI HTTP server

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Flask	Technology of Open source framework
2.	Security Implementations	CSRF protection, cookies protection, jinja templating and user input.	Jinja2
3.	Scalable Architecture	Micro Services	Micro web application framework by Flask
4.	Availability	1. built-in development server and fast debugger 2. integrated support for unit testing 3. RESTful request dispatching Jinja2 templating Unicode based	Jinja2
5.	Performance	ORM-agnostic, web framework, WSGI 1.0 compliant, HTTP request handling functionality High Flexibility	SQLAlchemy, extensions, Werkzeug, Jinja2, Sinatra Ruby framework.

PROJECT PLANNING AND SCHEDULING

Product Backlog, Sprint Schedule, and Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, and password, and confirming my password.	10	High
Sprint-1	E-mail confirmation	USN-2	As a user, I will receive a confirmation email once I have registered for the application	10	Medium
Sprint-2	Login	USN-3	As a user, I can log into the application by entering my email & password	5	High
Sprint-2	Upload Images	USN-4	As a user,I should be able to upload the image of ECG.	10	High
Sprint-2	Dashboard	USN-5	As a user, based on my requirement I can navigate through the dashboard.	5	Medium

Sprint-3	Train the model	Task 1	As a developer, the dataset will be uploaded and trained by developed algorithm.	20	High
Sprint-4	Testing & Evaluation	Task 2	As a developer, we tested the trained model using the provided dataset and model will be evaluated for accurate results.	10	High
Sprint-4	Display predicted result	USN-6	As a user, I can view the predicted result in the dashboard.	10	High

Project Tracker, Velocity & Burndown Chart:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Sprint Release Date (Actual)	Story Points Completed (as on Planned End Date)
Sprint-1	20	6 Days	1 Nov 2022	5 Nov 2022	29 Oct 2022	20
Sprint-2	20	6 Days	6 Nov 2022	10 Nov 2022	05 Nov 2022	20
Sprint-3	20	6 Days	11 Nov 2022	17 Nov 2022	12 Nov 2022	20
Sprint-4	20	6 Days	18 Nov 2022	25 Nov 2022	19 Nov 2022	20

CODING

PYTHON

```
import
os
          import numpy as np
          from flask import Flask, request, render_template
          from tensorflow.keras.models import load_model
          from tensorflow.keras.preprocessing import image
          app=Flask(_name_)
          model=load_model('ECG.h5')
```

```
@app.route("/")
def about():
  return render_template("about.html")
@app.route("/about")
def home():
  return render_template("about.html")
@app.route("/info")
def information():
  return render_template("info.html")
@app.route("/upload")
def test():
  return render_template("index6.html")
@app.route("/predict",methods-["GET", "POST"])
def upload():
if request.method== 'POST':
   f=request.files['file']
   basepath=os.path.dirname('_file_')
   filepath=os.path.join(basepath, "uploads", f.filename)
   f.save(filepath)
   img=image.load_img(filepath, target_size=(64,64))
   x=image.img_to_array(img)
   x=np.expand_dims(x,axis=0)
   pred=model.predict_classes (x)
   print("prediction", pred)
   index=['Left Bundle Branch Block', 'Normal', 'Premature Atrial Contraction", 'Premature
Fibrillation']
   result=str(index[pred[0]])
   return result

return None

if_name_=="_main_":
     app.run(debug=false)
```

sql

```
CREATE DATABASE IF NOT EXISTS 'geeklogin' DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci;
USE 'geeklogin';
CREATE TABLE IF NOT EXISTS 'accounts'(
    'id' int(11)NOT NULL AUTO_INCREMENT,
    'username' varchar(50) NOT NULL,
    'password' varchar(255) NOT NULL,
```

```
    'email' varchar(100) NOT NULL,
    PRIMARY KEY ('id')
    ) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

HTML Files

about.html

```
<htm
l>
            <head>
                <meta charset="UTF-8">
                <title> ABOUT </title>
                <link rel="stylesheet" href="style.css">
            </head>
            <body background="bg1.jpeg"></br></br></br></br></br>
                <div align="center">
                  <div align="center" class="border1">
                     <div class="header1">
                        <h1 class="word">ECG Arrhythmia Classification Using CNN</h1>
                     </div></br></br></br>
                        <h1 class="bottom">
                            Cardiovascular diseases (CVDs) are the leading cause of death globally,
        under 70 years of age.

        The most important behavioural risk factors of heart disease and stroke are unhealthy diet,
        stroke, heart failure and other complications.

        Cessation of tobacco use, reduction of salt in the diet, eating more fruit and vegetables, r

        Identifying those at highest risk of CVDs and ensuring they receive appropriate treatment ca
                        </h1></br></br></br>
                        <a href="Project Development Phase\SPRINT 2\info.html" class="btn">info</a>
                  </div>
                </div>
            </body>
        </html>
```

base.html

```
<!DOCTYPE
html>
```

```
<html lang="en">
<head>
  <title>ARRHYTHMIA CLASSIFICATION</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    #f{
      align-content: center;
    }
    .container {
  margin: auto;
  width: 20%;
  border: 3px solid white;
  background-color: #b6e2ff;
  border-radius: 25px;
  padding: 80px;
  align-content: center;
  text-align: center;
}

.b1 {
  background-color:black;
  border: none;
  border-radius: 25px;
  color: rgb(255, 255, 255);
  padding: 10px 22px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;

}
body{
  background-color:white;
}
h1{

  font-style: Arial;
  color: white

}
```

```
          </style>
        </head>
        <body background="bg1.jpeg">
          <script>
            var loadFile = function(event) {
              var image = document.getElementById('output');
              image.src = URL.createObjectURL(event.target.files[0]);
            };
            </script>
          <br><br>
          <h1  ><center>CLASSIFICATION OF ARRHYTHMIA</center></h1><br>
        <div class="container">

          <br><br>
          <form  action="{{ url_for('login') }}" method="post" enctype="multipart/form-data"


            <center><label><b>Upload Your  Image :</b></label><br><br></center>

            <center><input type="file"  accept="image/*" name="my_image" id="my_image" oncha
        ></center><br><br>
            <center><p><img id="output" width="200" /></p></center>

              <center><button type="submit" class="b1" onclick="alert('uploaded successfully
        </center>


          </form>

        {% if type %}

        <h2> TYPE   : <i> {{type}} </i></h2>

        {% endif %}
           </div>
          </center>
        </body>
```

index.html

```
<htm
l>
            <head>
                <meta charset="UTF-8">
```

```
            <title> Index </title>
            <link rel="stylesheet" href="style.css">
        </head>
        <body background="bg1.jpeg"></br></br></br></br></br>
            <div align="center">
              <div align="center" class="border1">
                 <div class="header1">
                    <h1 class="word">ECG Arrhythmia Classification Using CNN</h1>
                 </div></br></br></br>
                    <p class="bottom">
                        Cardiovascular diseases (CVDs) are the leading cause of death globally,
    under 70 years of age.

    The most important behavioural risk factors of heart disease and stroke are unhealthy diet,
    stroke, heart failure and other complications.

    Cessation of tobacco use, reduction of salt in the diet, eating more fruit and vegetables, r

    Identifying those at highest risk of CVDs and ensuring they receive appropriate treatment ca
                    </p></br></br></br>
                    <a href="info.html" class="btn">info</a>
                    <a href="base.html" class="btn">Predict</a>
              </div>
            </div>
        </body>
    </html>
```

info.html

```
<htm
l>
            <head>
                <meta charset="UTF-8">
                <title> Info </title>
                <link rel="stylesheet" href="style.css">
            </head>
            <body background="bg1.jpeg"></br></br></br></br></br>
                <div align="center">
                  <div align="center" class="border1">
                     <div class="header1">
                        <h1 class="word">ECG Arrhythmia Classification Using CNN</h1>
                     </div></br>
                        <h1 class="bottom">
                            Normal</br>
```

```
                </h1></br>
                <p>Your heart rate is the number of times each minute that your heart beats,
to your health.</p></br></br></br>
                <a href="index.html" class="btn">Home</a>
            </div>
          </div>
      </body>
</html>
```

style.css

```
.heade
r{
              padding: 5px 120px;
              width: 150px;
              height: 70px;
              background-color: black;
            transition: 0.3s;



          }
          .header1{
              padding: 5px 120px;
              width: 300px;
              height: 100px;
              background-color: black;
            transition: 0.3s;



          }

          .border{
              padding: 80px 50px;
              width: 400px;
              height: 450px;
              border-radius: 5px;
              background-color: white;
              transition: 0.3s;
              opacity: 0.8

          }

          .border1{
```

```
    padding: 80px 50px;
    width: 700px;
    height: 300px;
    border-radius: 5px;
    background-color: white;
    transition: 0.3s;
    opacity: 0.8

}


.btn {
    padding: 10px 40px;
    background-color: black;
    color: #FFFFFF;
    font-style: oblique;
    font-weight: bold;
    border-radius: 10px;
    box-shadow: 0 4px 8px 0
rgba(0,0,0,0.2);
    transition: 0.3s;
}

.textbox{
    padding: 10px 40px;
    background-color: black;
    text-color: #FFFFFF;
    border-radius: 10px;
}

::placeholder {
    color: #FFFFFF;
    opacity: 1;
    font-style: oblique;
    font-weight: bold;
  transition: 0.3s;

}

.word{
    color: #FFFFFF;
    font-style:normal;
    font-weight: bold;
```

```
        transition: 0.3s;

    }

    .bottom{
        color: black;
      transition: 0.3s;
      font-size: 12px;

    }
```

ipynb

```python
from keras.preprocessing.image import ImageDataGenerator
```

In [7]:

```python
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=
0.2,horizontal_flip=True)
```

In [8]:

```python
test_datagen=ImageDataGenerator(rescale=1./255)
```

In [9]:

```python
x_train=train_datagen.flow_from_directory(directory=r'C:\Users\91913\project
_dev\data\train',target_size=(64,64),batch_size=32,class_mode='categorical
')
```

```
Found 15341 images belonging to 6 classes.
```

In [10]:

```python
x_test=test_datagen.flow_from_directory(directory=r'C:\Users\91913\project_d
ev\data\test',target_size=(64,64),batch_size=32,class_mode='categorical')
```

```
Found 6825 images belonging to 6 classes.
```

In [13]:

```python
import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras.layers import Conv2D,MaxPooling2D
```

In [15]:

```python
model=Sequential()
model.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
```

In [17]:

```python
model.add(MaxPooling2D(pool_size=(2,2)))
```

In [19]:

```python
model.add(Conv2D(32,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

In [20]:

```
model.add(Flatten())
```

In [21]:

```
model.add(Dense(32))
```

In [22]:

```
model.add(Dense(6,activation='softmax'))
```

In [23]:

```
model.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0         
 )                                                               
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 15, 15, 32)       0         
 2D)                                                             
                                                                 
 conv2d_1 (Conv2D)           (None, 13, 13, 32)        9248      
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 6, 6, 32)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 1152)              0         
                                                                 
 dense (Dense)               (None, 32)                36896     
                                                                 
 dense_1 (Dense)             (None, 6)                 198       
                                                                 
=================================================================
Total params: 47,238
Trainable params: 47,238
Non-trainable params: 0
_________________________________________________________________
```

In [24]:

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['ac
curacy'])
```

In [26]:

```
model.fit_generator(generator=x_train,steps_per_epoch=len(x_train),epochs=10
,validation_data=x_test,validation_steps=len(x_test))
C:\Users\91913\AppData\Local\Temp\ipykernel_40184\1926459362.py:1:
```

```
UserWarning: `Model.fit_generator` is deprecated and will be removed in a
future version. Please use `Model.fit`, which supports generators.

model.fit_generator(generator=x_train,steps_per_epoch=len(x_train),epochs=10
,validation_data=x_test,validation_steps=len(x_test))
Epoch 1/10
480/480 [==============================] - 75s 154ms/step - loss: 0.8666 -
accuracy: 0.7056 - val_loss: 0.6230 - val_accuracy: 0.7796
Epoch 2/10
480/480 [==============================] - 59s 122ms/step - loss: 0.2768 -
accuracy: 0.9166 - val_loss: 0.4389 - val_accuracy: 0.8716
Epoch 3/10
480/480 [==============================] - 58s 122ms/step - loss: 0.2013 -
accuracy: 0.9391 - val_loss: 0.4706 - val_accuracy: 0.8675
Epoch 4/10
480/480 [==============================] - 59s 123ms/step - loss: 0.1729 -
accuracy: 0.9479 - val_loss: 0.3641 - val_accuracy: 0.9034
Epoch 5/10
480/480 [==============================] - 61s 127ms/step - loss: 0.1583 -
accuracy: 0.9519 - val_loss: 0.4433 - val_accuracy: 0.9010
Epoch 6/10
480/480 [==============================] - 61s 126ms/step - loss: 0.1429 -
accuracy: 0.9578 - val_loss: 0.4426 - val_accuracy: 0.8894
Epoch 7/10
480/480 [==============================] - 61s 127ms/step - loss: 0.1326 -
accuracy: 0.9598 - val_loss: 0.4584 - val_accuracy: 0.8869
Epoch 8/10
480/480 [==============================] - 60s 126ms/step - loss: 0.1246 -
accuracy: 0.9629 - val_loss: 0.4406 - val_accuracy: 0.8949
Epoch 9/10
480/480 [==============================] - 62s 129ms/step - loss: 0.1139 -
accuracy: 0.9643 - val_loss: 0.3692 - val_accuracy: 0.9140
Epoch 10/10
480/480 [==============================] - 60s 125ms/step - loss: 0.1084 -
accuracy: 0.9681 - val_loss: 0.3529 - val_accuracy: 0.9216
```

Out[26]:

In [27]:

```
model.save('ECG.h5')
```

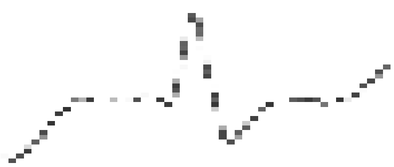
In [35]:

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
model=load_model("ECG.h5")
```

In [60]:

```
img=image.load_img('uploads\PAC.png',target_size=(64,64))
img
```

Out[60]:



In [61]:

```
x=image.img_to_array(img)
```

In [62]:

```
x=np.expand_dims(x,axis=0)
```

In [65]:

```
pred=model.predict(x)
1/1 [==============================] - 0s 221ms/step
```

In [66]:

```
pred
```

Out[66]:

```
array([[0.000000e+00, 1.000000e+00, 0.000000e+00, 2.204437e-11,
        0.000000e+00, 0.000000e+00]], dtype=float32)
```

In [71]:

```
index=['Left Bundle Branch Block','Normal','Premature Atrial Contraction','Premature ventricular Contractions','Right bundle branch block','Ventricular fibrillation']
pred_id=pred.argmax(axis=1)[0]
```

In [73]:

```
result=str(index[pred_id])
result
```

Out[73]:

```
'Normal'
```

final ipynb

```
pwd
```

Out[1]:

```
'/home/wsuser/work'
```

In [2]:

```
imageSize = [299, 299]
trainPath = "C:\\Users\91913\project_dev\data\train"
testPath ="C:\\Users\91913\project_dev\data\test"
```

In [3]:

```
import tensorflow as tf
```

```
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.models import Model,load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img
from tensorflow.keras.applications.xception import Xception,
preprocess_input
from glob import glob
import numpy as np
import matplotlib.pyplot as plt
import h5py
```

In [4]:

```
train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2,
zoom_range = 0.2, horizontal_flip = True, vertical_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
type(train_datagen)
```

Out[4]:

```
keras.preprocessing.image.ImageDataGenerator
```

In [5]:

```
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

cos_client = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='RzL20fskxy-Y1gjml2csOOZat7w24Bdi8TNvronExyp7',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.private.us.cloud-object-
storage.appdomain.cloud')

bucket = 'arrhythmiafinal-donotdelete-pr-f5nzyd891ylukm'
object_key = 'preprocessed dataset.zip'

streaming_body_1 = cos_client.get_object(Bucket=bucket,
Key=object_key)['Body']
```

In [7]:

```
from io import BytesIO
import zipfile
unzip=zipfile.ZipFile(BytesIO(streaming_body_1.read()),'r')
file_paths=unzip.namelist()
```

```
for path in file_paths:
    unzip.extract(path)
```

In [8]:

```
pwd
```

Out[8]:

```
'/home/wsuser/work'
```

In [9]:

```
import os
filenames=os.listdir('/home/wsuser/work/preprocessed dataset/preprocessed
dataset/training')
```

In [11]:

```
training_set =train_datagen.flow_from_directory(trainPath, target_size =
(299,299),batch_size = 4, class_mode = 'categorical')
test_set = test_datagen.flow_from_directory(testPath, target_size =
(299,299),batch_size = 4, class_mode = 'categorical')
Found 137 images belonging to 5 classes.
Found 117 images belonging to 5 classes.
```

In [12]:

```
xception = Xception(input_shape = imageSize + [3], weights = 'imagenet',
include_top = False)
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 0s 0us/step
83697664/83683744 [==============================] - 0s 0us/step
```

In [13]:

```
for layer in xception.layers:
  layer.trainable = False

x = Flatten()(xception.output)

prediction = Dense(5, activation = 'softmax')(x)

model = Model(inputs = xception.input, outputs = prediction)

model.summary()
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected
to
==================================================================================================
 input_1 (InputLayer)           [(None, 299, 299, 3  0           []
```

```
                                )]

 block1_conv1 (Conv2D)          (None, 149, 149, 32  864
['input_1[0][0]']
                                )

 block1_conv1_bn (BatchNormaliz  (None, 149, 149, 32  128
['block1_conv1[0][0]']
 ation)                         )

 block1_conv1_act (Activation)  (None, 149, 149, 32  0
['block1_conv1_bn[0][0]']
                                )

 block1_conv2 (Conv2D)          (None, 147, 147, 64  18432
['block1_conv1_act[0][0]']
                                )

 block1_conv2_bn (BatchNormaliz  (None, 147, 147, 64  256
['block1_conv2[0][0]']
 ation)                         )

 block1_conv2_act (Activation)  (None, 147, 147, 64  0
['block1_conv2_bn[0][0]']
                                )

 block2_sepconv1 (SeparableConv  (None, 147, 147, 12  8768
['block1_conv2_act[0][0]']
 2D)                            8)

 block2_sepconv1_bn (BatchNorma  (None, 147, 147, 12  512
['block2_sepconv1[0][0]']
 lization)                      8)

 block2_sepconv2_act (Activatio  (None, 147, 147, 12  0
['block2_sepconv1_bn[0][0]']
 n)                             8)

 block2_sepconv2 (SeparableConv  (None, 147, 147, 12  17536
['block2_sepconv2_act[0][0]']
 2D)                            8)

 block2_sepconv2_bn (BatchNorma  (None, 147, 147, 12  512
```

```
['block2_sepconv2[0][0]']
 lization)                      8)

 conv2d (Conv2D)                (None, 74, 74, 128)  8192
['block1_conv2_act[0][0]']

 block2_pool (MaxPooling2D)     (None, 74, 74, 128)  0
['block2_sepconv2_bn[0][0]']

 batch_normalization (BatchNorm  (None, 74, 74, 128)  512
['conv2d[0][0]']
 alization)

 add (Add)                      (None, 74, 74, 128)  0
['block2_pool[0][0]',

'batch_normalization[0][0]']

 block3_sepconv1_act (Activatio  (None, 74, 74, 128)  0
['add[0][0]']
 n)

 block3_sepconv1 (SeparableConv  (None, 74, 74, 256)  33920
['block3_sepconv1_act[0][0]']
 2D)

 block3_sepconv1_bn (BatchNorma  (None, 74, 74, 256)  1024
['block3_sepconv1[0][0]']
 lization)

 block3_sepconv2_act (Activatio  (None, 74, 74, 256)  0
['block3_sepconv1_bn[0][0]']
 n)

 block3_sepconv2 (SeparableConv  (None, 74, 74, 256)  67840
['block3_sepconv2_act[0][0]']
 2D)

 block3_sepconv2_bn (BatchNorma  (None, 74, 74, 256)  1024
['block3_sepconv2[0][0]']
 lization)

 conv2d_1 (Conv2D)              (None, 37, 37, 256)  32768
```

```
['add[0][0]']

 block3_pool (MaxPooling2D)     (None, 37, 37, 256)  0
['block3_sepconv2_bn[0][0]']

 batch_normalization_1 (BatchNo  (None, 37, 37, 256)  1024
['conv2d_1[0][0]']
 rmalization)

 add_1 (Add)                    (None, 37, 37, 256)  0
['block3_pool[0][0]',

'batch_normalization_1[0][0]']

 block4_sepconv1_act (Activatio  (None, 37, 37, 256)  0
['add_1[0][0]']
 n)

 block4_sepconv1 (SeparableConv  (None, 37, 37, 728)  188672
['block4_sepconv1_act[0][0]']
 2D)

 block4_sepconv1_bn (BatchNorma  (None, 37, 37, 728)  2912
['block4_sepconv1[0][0]']
 lization)

 block4_sepconv2_act (Activatio  (None, 37, 37, 728)  0
['block4_sepconv1_bn[0][0]']
 n)

 block4_sepconv2 (SeparableConv  (None, 37, 37, 728)  536536
['block4_sepconv2_act[0][0]']
 2D)

 block4_sepconv2_bn (BatchNorma  (None, 37, 37, 728)  2912
['block4_sepconv2[0][0]']
 lization)

 conv2d_2 (Conv2D)              (None, 19, 19, 728)  186368
['add_1[0][0]']

 block4_pool (MaxPooling2D)     (None, 19, 19, 728)  0
['block4_sepconv2_bn[0][0]']
```

```
 batch_normalization_2 (BatchNo  (None, 19, 19, 728)  2912
['conv2d_2[0][0]']
 rmalization)

 add_2 (Add)                    (None, 19, 19, 728)  0
['block4_pool[0][0]',

'batch_normalization_2[0][0]']

 block5_sepconv1_act (Activatio  (None, 19, 19, 728)  0
['add_2[0][0]']
 n)

 block5_sepconv1 (SeparableConv  (None, 19, 19, 728)  536536
['block5_sepconv1_act[0][0]']
 2D)

 block5_sepconv1_bn (BatchNorma  (None, 19, 19, 728)  2912
['block5_sepconv1[0][0]']
 lization)

 block5_sepconv2_act (Activatio  (None, 19, 19, 728)  0
['block5_sepconv1_bn[0][0]']
 n)

 block5_sepconv2 (SeparableConv  (None, 19, 19, 728)  536536
['block5_sepconv2_act[0][0]']
 2D)

 block5_sepconv2_bn (BatchNorma  (None, 19, 19, 728)  2912
['block5_sepconv2[0][0]']
 lization)

 block5_sepconv3_act (Activatio  (None, 19, 19, 728)  0
['block5_sepconv2_bn[0][0]']
 n)

 block5_sepconv3 (SeparableConv  (None, 19, 19, 728)  536536
['block5_sepconv3_act[0][0]']
 2D)

 block5_sepconv3_bn (BatchNorma  (None, 19, 19, 728)  2912
```

```
['block5_sepconv3[0][0]']
 lization)

 add_3 (Add)                    (None, 19, 19, 728)  0
['block5_sepconv3_bn[0][0]',

'add_2[0][0]']

 block6_sepconv1_act (Activatio  (None, 19, 19, 728)  0
['add_3[0][0]']
 n)

 block6_sepconv1 (SeparableConv  (None, 19, 19, 728)  536536
['block6_sepconv1_act[0][0]']
 2D)

 block6_sepconv1_bn (BatchNorma  (None, 19, 19, 728)  2912
['block6_sepconv1[0][0]']
 lization)

 block6_sepconv2_act (Activatio  (None, 19, 19, 728)  0
['block6_sepconv1_bn[0][0]']
 n)

 block6_sepconv2 (SeparableConv  (None, 19, 19, 728)  536536
['block6_sepconv2_act[0][0]']
 2D)

 block6_sepconv2_bn (BatchNorma  (None, 19, 19, 728)  2912
['block6_sepconv2[0][0]']
 lization)

 block6_sepconv3_act (Activatio  (None, 19, 19, 728)  0
['block6_sepconv2_bn[0][0]']
 n)

 block6_sepconv3 (SeparableConv  (None, 19, 19, 728)  536536
['block6_sepconv3_act[0][0]']
 2D)

 block6_sepconv3_bn (BatchNorma  (None, 19, 19, 728)  2912
['block6_sepconv3[0][0]']
 lization)
```

```
 add_4 (Add)                    (None, 19, 19, 728)  0
['block6_sepconv3_bn[0][0]',

'add_3[0][0]']

 block7_sepconv1_act (Activatio  (None, 19, 19, 728)  0
['add_4[0][0]']
 n)

 block7_sepconv1 (SeparableConv  (None, 19, 19, 728)  536536
['block7_sepconv1_act[0][0]']
 2D)

 block7_sepconv1_bn (BatchNorma  (None, 19, 19, 728)  2912
['block7_sepconv1[0][0]']
 lization)

 block7_sepconv2_act (Activatio  (None, 19, 19, 728)  0
['block7_sepconv1_bn[0][0]']
 n)

 block7_sepconv2 (SeparableConv  (None, 19, 19, 728)  536536
['block7_sepconv2_act[0][0]']
 2D)

 block7_sepconv2_bn (BatchNorma  (None, 19, 19, 728)  2912
['block7_sepconv2[0][0]']
 lization)

 block7_sepconv3_act (Activatio  (None, 19, 19, 728)  0
['block7_sepconv2_bn[0][0]']
 n)

 block7_sepconv3 (SeparableConv  (None, 19, 19, 728)  536536
['block7_sepconv3_act[0][0]']
 2D)

 block7_sepconv3_bn (BatchNorma  (None, 19, 19, 728)  2912
['block7_sepconv3[0][0]']
 lization)

 add_5 (Add)                    (None, 19, 19, 728)  0
```

```
['block7_sepconv3_bn[0][0]',

'add_4[0][0]']

 block8_sepconv1_act (Activatio  (None, 19, 19, 728)  0
['add_5[0][0]']
 n)

 block8_sepconv1 (SeparableConv  (None, 19, 19, 728)  536536
['block8_sepconv1_act[0][0]']
 2D)

 block8_sepconv1_bn (BatchNorma  (None, 19, 19, 728)  2912
['block8_sepconv1[0][0]']
 lization)

 block8_sepconv2_act (Activatio  (None, 19, 19, 728)  0
['block8_sepconv1_bn[0][0]']
 n)

 block8_sepconv2 (SeparableConv  (None, 19, 19, 728)  536536
['block8_sepconv2_act[0][0]']
 2D)

 block8_sepconv2_bn (BatchNorma  (None, 19, 19, 728)  2912
['block8_sepconv2[0][0]']
 lization)

 block8_sepconv3_act (Activatio  (None, 19, 19, 728)  0
['block8_sepconv2_bn[0][0]']
 n)

 block8_sepconv3 (SeparableConv  (None, 19, 19, 728)  536536
['block8_sepconv3_act[0][0]']
 2D)

 block8_sepconv3_bn (BatchNorma  (None, 19, 19, 728)  2912
['block8_sepconv3[0][0]']
 lization)

 add_6 (Add)                    (None, 19, 19, 728)  0
['block8_sepconv3_bn[0][0]',
```

```
'add_5[0][0]']

 block9_sepconv1_act (Activatio  (None, 19, 19, 728)  0
['add_6[0][0]']
 n)

 block9_sepconv1 (SeparableConv  (None, 19, 19, 728)  536536
['block9_sepconv1_act[0][0]']
 2D)

 block9_sepconv1_bn (BatchNorma  (None, 19, 19, 728)  2912
['block9_sepconv1[0][0]']
 lization)

 block9_sepconv2_act (Activatio  (None, 19, 19, 728)  0
['block9_sepconv1_bn[0][0]']
 n)

 block9_sepconv2 (SeparableConv  (None, 19, 19, 728)  536536
['block9_sepconv2_act[0][0]']
 2D)

 block9_sepconv2_bn (BatchNorma  (None, 19, 19, 728)  2912
['block9_sepconv2[0][0]']
 lization)

 block9_sepconv3_act (Activatio  (None, 19, 19, 728)  0
['block9_sepconv2_bn[0][0]']
 n)

 block9_sepconv3 (SeparableConv  (None, 19, 19, 728)  536536
['block9_sepconv3_act[0][0]']
 2D)

 block9_sepconv3_bn (BatchNorma  (None, 19, 19, 728)  2912
['block9_sepconv3[0][0]']
 lization)

 add_7 (Add)                    (None, 19, 19, 728)  0
['block9_sepconv3_bn[0][0]',

'add_6[0][0]']
```

```
 block10_sepconv1_act (Activati  (None, 19, 19, 728)  0
['add_7[0][0]']
 on)

 block10_sepconv1 (SeparableCon  (None, 19, 19, 728)  536536
['block10_sepconv1_act[0][0]']
 v2D)

 block10_sepconv1_bn (BatchNorm  (None, 19, 19, 728)  2912
['block10_sepconv1[0][0]']
 alization)

 block10_sepconv2_act (Activati  (None, 19, 19, 728)  0
['block10_sepconv1_bn[0][0]']
 on)

 block10_sepconv2 (SeparableCon  (None, 19, 19, 728)  536536
['block10_sepconv2_act[0][0]']
 v2D)

 block10_sepconv2_bn (BatchNorm  (None, 19, 19, 728)  2912
['block10_sepconv2[0][0]']
 alization)

 block10_sepconv3_act (Activati  (None, 19, 19, 728)  0
['block10_sepconv2_bn[0][0]']
 on)

 block10_sepconv3 (SeparableCon  (None, 19, 19, 728)  536536
['block10_sepconv3_act[0][0]']
 v2D)

 block10_sepconv3_bn (BatchNorm  (None, 19, 19, 728)  2912
['block10_sepconv3[0][0]']
 alization)

 add_8 (Add)                    (None, 19, 19, 728)  0
['block10_sepconv3_bn[0][0]',

'add_7[0][0]']

 block11_sepconv1_act (Activati  (None, 19, 19, 728)  0
['add_8[0][0]']
```

```
 on)

 block11_sepconv1 (SeparableCon  (None, 19, 19, 728)  536536
['block11_sepconv1_act[0][0]']
 v2D)

 block11_sepconv1_bn (BatchNorm  (None, 19, 19, 728)  2912
['block11_sepconv1[0][0]']
 alization)

 block11_sepconv2_act (Activati  (None, 19, 19, 728)  0
['block11_sepconv1_bn[0][0]']
 on)

 block11_sepconv2 (SeparableCon  (None, 19, 19, 728)  536536
['block11_sepconv2_act[0][0]']
 v2D)

 block11_sepconv2_bn (BatchNorm  (None, 19, 19, 728)  2912
['block11_sepconv2[0][0]']
 alization)

 block11_sepconv3_act (Activati  (None, 19, 19, 728)  0
['block11_sepconv2_bn[0][0]']
 on)

 block11_sepconv3 (SeparableCon  (None, 19, 19, 728)  536536
['block11_sepconv3_act[0][0]']
 v2D)

 block11_sepconv3_bn (BatchNorm  (None, 19, 19, 728)  2912
['block11_sepconv3[0][0]']
 alization)

 add_9 (Add)                    (None, 19, 19, 728)  0
['block11_sepconv3_bn[0][0]',

'add_8[0][0]']

 block12_sepconv1_act (Activati  (None, 19, 19, 728)  0
['add_9[0][0]']
 on)
```

```
 block12_sepconv1 (SeparableCon  (None, 19, 19, 728)  536536
['block12_sepconv1_act[0][0]']
 v2D)

 block12_sepconv1_bn (BatchNorm  (None, 19, 19, 728)  2912
['block12_sepconv1[0][0]']
 alization)

 block12_sepconv2_act (Activati  (None, 19, 19, 728)  0
['block12_sepconv1_bn[0][0]']
 on)

 block12_sepconv2 (SeparableCon  (None, 19, 19, 728)  536536
['block12_sepconv2_act[0][0]']
 v2D)

 block12_sepconv2_bn (BatchNorm  (None, 19, 19, 728)  2912
['block12_sepconv2[0][0]']
 alization)

 block12_sepconv3_act (Activati  (None, 19, 19, 728)  0
['block12_sepconv2_bn[0][0]']
 on)

 block12_sepconv3 (SeparableCon  (None, 19, 19, 728)  536536
['block12_sepconv3_act[0][0]']
 v2D)

 block12_sepconv3_bn (BatchNorm  (None, 19, 19, 728)  2912
['block12_sepconv3[0][0]']
 alization)

 add_10 (Add)                   (None, 19, 19, 728)  0
['block12_sepconv3_bn[0][0]',

'add_9[0][0]']

 block13_sepconv1_act (Activati  (None, 19, 19, 728)  0
['add_10[0][0]']
 on)

 block13_sepconv1 (SeparableCon  (None, 19, 19, 728)  536536
['block13_sepconv1_act[0][0]']
```

```
 v2D)

 block13_sepconv1_bn (BatchNorm  (None, 19, 19, 728)  2912
['block13_sepconv1[0][0]']
 alization)

 block13_sepconv2_act (Activati  (None, 19, 19, 728)  0
['block13_sepconv1_bn[0][0]']
 on)

 block13_sepconv2 (SeparableCon  (None, 19, 19, 1024  752024
['block13_sepconv2_act[0][0]']
 v2D)                           )

 block13_sepconv2_bn (BatchNorm  (None, 19, 19, 1024  4096
['block13_sepconv2[0][0]']
 alization)                     )

 conv2d_3 (Conv2D)              (None, 10, 10, 1024  745472
['add_10[0][0]']
                                )

 block13_pool (MaxPooling2D)    (None, 10, 10, 1024  0
['block13_sepconv2_bn[0][0]']
                                )

 batch_normalization_3 (BatchNo  (None, 10, 10, 1024  4096
['conv2d_3[0][0]']
 rmalization)                   )

 add_11 (Add)                   (None, 10, 10, 1024  0
['block13_pool[0][0]',
                                )
'batch_normalization_3[0][0]']

 block14_sepconv1 (SeparableCon  (None, 10, 10, 1536  1582080
['add_11[0][0]']
 v2D)                           )

 block14_sepconv1_bn (BatchNorm  (None, 10, 10, 1536  6144
['block14_sepconv1[0][0]']
 alization)                     )
```

```
 block14_sepconv1_act (Activati  (None, 10, 10, 1536  0
['block14_sepconv1_bn[0][0]']
 on)                             )

 block14_sepconv2 (SeparableCon  (None, 10, 10, 2048  3159552
['block14_sepconv1_act[0][0]']
 v2D)                            )

 block14_sepconv2_bn (BatchNorm  (None, 10, 10, 2048  8192
['block14_sepconv2[0][0]']
 alization)                      )

 block14_sepconv2_act (Activati  (None, 10, 10, 2048  0
['block14_sepconv2_bn[0][0]']
 on)                             )

 flatten (Flatten)              (None, 204800)       0
['block14_sepconv2_act[0][0]']

 dense (Dense)                  (None, 5)            1024005
['flatten[0][0]']

============================================================================
=======================
Total params: 21,885,485
Trainable params: 1,024,005
Non-trainable params: 20,861,480
____________________________________________________________________________
_______________________
```

In [15]:

```
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])
```

In [17]:

```
r = model.fit_generator(training_set, validation_data = test_set, epochs =
50, steps_per_epoch = len(training_set)//32, validation_steps =
len(test_set)//32)
/tmp/wsuser/ipykernel_164/966271731.py:1: UserWarning: `Model.fit_generator`
is deprecated and will be removed in a future version. Please use
`Model.fit`, which supports generators.
  r = model.fit_generator(training_set, validation_data = test_set, epochs =
50, steps_per_epoch = len(training_set)//32, validation_steps =
len(test_set)//32)
Epoch 1/50
```

```
1/1 [==============================] - 2s 2s/step - loss: 20.8216 -
accuracy: 0.5000
Epoch 2/50
1/1 [==============================] - 2s 2s/step - loss: 13.3012 -
accuracy: 0.7500
Epoch 3/50
1/1 [==============================] - 2s 2s/step - loss: 10.3953 -
accuracy: 0.5000
Epoch 4/50
1/1 [==============================] - 2s 2s/step - loss: 16.1016 -
accuracy: 0.5000
Epoch 5/50
1/1 [==============================] - 2s 2s/step - loss: 16.1712 -
accuracy: 0.5000
Epoch 6/50
1/1 [==============================] - 2s 2s/step - loss: 22.0960 -
accuracy: 0.2500
Epoch 7/50
1/1 [==============================] - 2s 2s/step - loss: 13.6296 -
accuracy: 0.2500
Epoch 8/50
1/1 [==============================] - 2s 2s/step - loss: 21.5347 -
accuracy: 0.2500
Epoch 9/50
1/1 [==============================] - 2s 2s/step - loss: 0.0067 -
accuracy: 1.0000
Epoch 10/50
1/1 [==============================] - 2s 2s/step - loss: 1.9726 -
accuracy: 0.7500
Epoch 11/50
1/1 [==============================] - 2s 2s/step - loss: 4.1016 -
accuracy: 0.5000
Epoch 12/50
1/1 [==============================] - 2s 2s/step - loss: 3.5344 -
accuracy: 0.5000
Epoch 13/50
1/1 [==============================] - 2s 2s/step - loss: 27.7313 -
accuracy: 0.0000e+00
Epoch 14/50
1/1 [==============================] - 2s 2s/step - loss: 12.4525 -
accuracy: 0.5000
Epoch 15/50
1/1 [==============================] - 2s 2s/step - loss: 10.8806 -
```

```
accuracy: 0.5000
Epoch 16/50
1/1 [==============================] - 2s 2s/step - loss: 15.4410 -
accuracy: 0.5000
Epoch 17/50
1/1 [==============================] - 2s 2s/step - loss: 2.5684 -
accuracy: 0.7500
Epoch 18/50
1/1 [==============================] - 2s 2s/step - loss: 0.0921 -
accuracy: 1.0000
Epoch 19/50
1/1 [==============================] - 2s 2s/step - loss: 3.5925 -
accuracy: 0.7500
Epoch 20/50
1/1 [==============================] - 2s 2s/step - loss: 11.6361 -
accuracy: 0.5000
Epoch 21/50
1/1 [==============================] - 1s 1s/step - loss: 20.0808 -
accuracy: 0.5000
Epoch 22/50
1/1 [==============================] - 2s 2s/step - loss: 4.6868 -
accuracy: 0.7500
Epoch 23/50
1/1 [==============================] - 2s 2s/step - loss: 5.9605e-08 -
accuracy: 1.0000
Epoch 24/50
1/1 [==============================] - 2s 2s/step - loss: 17.2250 -
accuracy: 0.7500
Epoch 25/50
1/1 [==============================] - 2s 2s/step - loss: 22.6145 -
accuracy: 0.2500
Epoch 26/50
1/1 [==============================] - 2s 2s/step - loss: 6.1318 -
accuracy: 0.7500
Epoch 27/50
1/1 [==============================] - 2s 2s/step - loss: 21.4852 -
accuracy: 0.2500
Epoch 28/50
1/1 [==============================] - 2s 2s/step - loss: 15.6949 -
accuracy: 0.5000
Epoch 29/50
1/1 [==============================] - 2s 2s/step - loss: 0.7244 -
accuracy: 0.5000
```

```
Epoch 30/50
1/1 [==============================] - 1s 897ms/step - loss: 0.0000e+00 -
accuracy: 1.0000
Epoch 31/50
1/1 [==============================] - 2s 2s/step - loss: 0.0033 -
accuracy: 1.0000
Epoch 32/50
1/1 [==============================] - 2s 2s/step - loss: 20.9464 -
accuracy: 0.2500
Epoch 33/50
1/1 [==============================] - 2s 2s/step - loss: 0.0160 -
accuracy: 1.0000
Epoch 34/50
1/1 [==============================] - 2s 2s/step - loss: 7.6429 -
accuracy: 0.7500
Epoch 35/50
1/1 [==============================] - 2s 2s/step - loss: 0.2537 -
accuracy: 0.7500
Epoch 36/50
1/1 [==============================] - 2s 2s/step - loss: 8.1358 -
accuracy: 0.5000
Epoch 37/50
1/1 [==============================] - 2s 2s/step - loss: 5.3044 -
accuracy: 0.5000
Epoch 38/50
1/1 [==============================] - 2s 2s/step - loss: 0.8009 -
accuracy: 0.7500
Epoch 39/50
1/1 [==============================] - 2s 2s/step - loss: 3.2886 -
accuracy: 0.7500
Epoch 40/50
1/1 [==============================] - 2s 2s/step - loss: 11.5262 -
accuracy: 0.5000
Epoch 41/50
1/1 [==============================] - 2s 2s/step - loss: 27.1964 -
accuracy: 0.5000
Epoch 42/50
1/1 [==============================] - 2s 2s/step - loss: 6.3216 -
accuracy: 0.7500
Epoch 43/50
1/1 [==============================] - 2s 2s/step - loss: 5.2122 -
accuracy: 0.5000
Epoch 44/50
```

```
1/1 [==============================] - 1s 1s/step - loss: 9.0899 -
accuracy: 0.5000
Epoch 45/50
1/1 [==============================] - 2s 2s/step - loss: 2.9802e-08 -
accuracy: 1.0000
Epoch 46/50
1/1 [==============================] - 2s 2s/step - loss: 12.1451 -
accuracy: 0.5000
Epoch 47/50
1/1 [==============================] - 2s 2s/step - loss: 6.7772 -
accuracy: 0.5000
Epoch 48/50
1/1 [==============================] - 2s 2s/step - loss: 4.3266 -
accuracy: 0.5000
Epoch 49/50
1/1 [==============================] - 2s 2s/step - loss: 3.1950 -
accuracy: 0.7500
Epoch 50/50
1/1 [==============================] - 2s 2s/step - loss: 4.8329 -
accuracy: 0.7500
```

In [18]:

```
model.save('final.h5')
/opt/conda/envs/Python-3.9/lib/python3.9/site-
packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask
layers require a config and must override get_config. When loading, the
custom mask layer must be passed to the custom_objects argument.
  layer_config = serialize_layer_fn(layer)
```

In [21]:

```
!pip install watson-machine-learning-client --upgrade
Collecting watson-machine-learning-client
  Downloading watson_machine_learning_client-1.0.391-py3-none-any.whl (538
kB)
     |████████████████████████████████| 538 kB 14.2 MB/s eta 0:00:01
Requirement already satisfied: requests in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(2.26.0)
Requirement already satisfied: boto3 in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(1.18.21)
Requirement already satisfied: urllib3 in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(1.26.7)
Requirement already satisfied: tqdm in /opt/conda/envs/Python-
```

```
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(4.62.3)
Requirement already satisfied: ibm-cos-sdk in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(2.11.0)
Requirement already satisfied: lomond in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(0.3.3)
Requirement already satisfied: tabulate in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(0.8.9)
Requirement already satisfied: certifi in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(2022.9.24)
Requirement already satisfied: pandas in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from watson-machine-learning-client)
(1.3.4)
Requirement already satisfied: botocore<1.22.0,>=1.21.21 in
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from boto3->watson-
machine-learning-client) (1.21.41)
Requirement already satisfied: s3transfer<0.6.0,>=0.5.0 in
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from boto3->watson-
machine-learning-client) (0.5.0)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from boto3->watson-
machine-learning-client) (0.10.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from
botocore<1.22.0,>=1.21.21->boto3->watson-machine-learning-client) (2.8.2)
Requirement already satisfied: six>=1.5 in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1-
>botocore<1.22.0,>=1.21.21->boto3->watson-machine-learning-client) (1.15.0)
Requirement already satisfied: ibm-cos-sdk-core==2.11.0 in
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-cos-sdk-
>watson-machine-learning-client) (2.11.0)
Requirement already satisfied: ibm-cos-sdk-s3transfer==2.11.0 in
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-cos-sdk-
>watson-machine-learning-client) (2.11.0)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from requests-
>watson-machine-learning-client) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from requests->watson-machine-learning-
```

```
client) (3.3)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from pandas->watson-machine-learning-
client) (2021.3)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/envs/Python-
3.9/lib/python3.9/site-packages (from pandas->watson-machine-learning-
client) (1.20.3)
Installing collected packages: watson-machine-learning-client
Successfully installed watson-machine-learning-client-1.0.391
```

In [24]:

```
from ibm_watson_machine_learning import APIClient

wml_credentials = {
    'url':'https://us-south.ml.cloud.ibm.com',
    'apikey':'Nh5wxaW-n-x4wen51BeGt-gVRFumapUtikckfqs2T74u'
}

client = APIClient(wml_credentials)
```

In [25]:

```
def guid_from_space_name(client,space_name):
    space=client.spaces.get_details()
    return(next(item for item in space['resources'] if
item['entity']["name"]==space_name)['metadata']['id'])
```

In [26]:

```
space_uid=guid_from_space_name(client,'classify_arrhythmia')
print("Space UID="+space_uid)
Space UID=5eb3dfc2-35d5-44b5-9689-6f2dd7dadb95
```

In [27]:

```
client.set.default_space(space_uid)
```

Out[27]:

```
'SUCCESS'
```

In [28]:

```
client.software_specifications.list()
-----------------------------  ------------------------------------  ----
NAME                           ASSET_ID                              TYPE
default_py3.6                  0062b8c9-8b7d-44a0-a9b9-46c416adcbd9  base
kernel-spark3.2-scala2.12      020d69ce-7ac1-5e68-ac1a-31189867356a  base
pytorch-onnx_1.3-py3.7-edt     069ea134-3346-5748-b513-49120e15d288  base
scikit-learn_0.20-py3.6        09c5a1d0-9c1e-4473-a344-eb7b665ff687  base
spark-mllib_3.0-scala_2.12     09f4cff0-90a7-5899-b9ed-1ef348aebdee  base
pytorch-onnx_rt22.1-py3.9      0b848dd4-e681-5599-be41-b5f6fccc6471  base
ai-function_0.1-py3.6          0cdb0f1e-5376-4f4d-92dd-da3b69aa9bda  base
shiny-r3.6                     0e6e79df-875e-4f24-8ae9-62dcc2148306  base
```

```
tensorflow_2.4-py3.7-horovod   1092590a-307d-563d-9b62-4eb7d64b3f22  base
pytorch_1.1-py3.6               10ac12d6-6b30-4ccd-8392-3e922c096a92  base
tensorflow_1.15-py3.6-ddl      111e41b3-de2d-5422-a4d6-bf776828c4b7  base
autoai-kb_rt22.2-py3.10         125b6d9a-5b1f-5e8d-972a-b251688ccf40  base
runtime-22.1-py3.9              12b83a17-24d8-5082-900f-0ab31fbfd3cb  base
scikit-learn_0.22-py3.6         154010fa-5b3b-4ac1-82af-4d5ee5abbc85  base
default_r3.6                    1b70aec3-ab34-4b87-8aa0-a4a3c8296a36  base
pytorch-onnx_1.3-py3.6          1bc6029a-cc97-56da-b8e0-39c3880dbbe7  base
kernel-spark3.3-r3.6            1c9e5454-f216-59dd-a20e-474a5cdf5988  base
pytorch-onnx_rt22.1-py3.9-edt  1d362186-7ad5-5b59-8b6c-9d0880bde37f  base
tensorflow_2.1-py3.6           1eb25b84-d6ed-5dde-b6a5-3fbdf1665666  base
spark-mllib_3.2                 20047f72-0a98-58c7-9ff5-a77b012eb8f5  base
tensorflow_2.4-py3.8-horovod   217c16f6-178f-56bf-824a-b19f20564c49  base
runtime-22.1-py3.9-cuda         26215f05-08c3-5a41-a1b0-da66306ce658  base
do_py3.8                        295addb5-9ef9-547e-9bf4-92ae3563e720  base
autoai-ts_3.8-py3.8             2aa0c932-798f-5ae9-abd6-15e0c2402fb5  base
tensorflow_1.15-py3.6          2b73a275-7cbf-420b-a912-eae7f436e0bc  base
kernel-spark3.3-py3.9           2b7961e2-e3b1-5a8c-a491-482c8368839a  base
pytorch_1.2-py3.6               2c8ef57d-2687-4b7d-acce-01f94976dac1  base
spark-mllib_2.3                 2e51f700-bca0-4b0d-88dc-5c6791338875  base
pytorch-onnx_1.1-py3.6-edt      32983cea-3f32-4400-8965-dde874a8d67e  base
spark-mllib_3.0-py37            36507ebe-8770-55ba-ab2a-eafe787600e9  base
spark-mllib_2.4                 390d21f8-e58b-4fac-9c55-d7ceda621326  base
autoai-ts_rt22.2-py3.10         396b2e83-0953-5b86-9a55-7ce1628a406f  base
xgboost_0.82-py3.6              39e31acd-5f30-41dc-ae44-60233c80306e  base
pytorch-onnx_1.2-py3.6-edt      40589d0e-7019-4e28-8daa-fb03b6f4fe12  base
pytorch-onnx_rt22.2-py3.10      40e73f55-783a-5535-b3fa-0c8b94291431  base
default_r36py38                 41c247d3-45f8-5a71-b065-8580229facf0  base
autoai-ts_rt22.1-py3.9          4269d26e-07ba-5d40-8f66-2d495b0c71f7  base
autoai-obm_3.0                  42b92e18-d9ab-567f-988a-4240ba1ed5f7  base
pmml-3.0_4.3                    493bcb95-16f1-5bc5-bee8-81b8af80e9c7  base
spark-mllib_2.4-r_3.6           49403dff-92e9-4c87-a3d7-a42d0021c095  base
xgboost_0.90-py3.6              4ff8d6c2-1343-4c18-85e1-689c965304d3  base
pytorch-onnx_1.1-py3.6          50f95b2a-bc16-43bb-bc94-b0bed208c60b  base
autoai-ts_3.9-py3.8             52c57136-80fa-572e-8728-a5e7cbb42cde  base
spark-mllib_2.4-scala_2.11      55a70f99-7320-4be5-9fb9-9edb5a443af5  base
spark-mllib_3.0                 5c1b0ca2-4977-5c2e-9439-ffd44ea8ffe9  base
autoai-obm_2.0                  5c2e37fa-80b8-5e77-840f-d912469614ee  base
spss-modeler_18.1               5c3cad7e-507f-4b2a-a9a3-ab53a21dee8b  base
cuda-py3.8                      5d3232bf-c86b-5df4-a2cd-7bb870a1cd4e  base
runtime-22.2-py3.10-xc          5e8cddff-db4a-5a6a-b8aa-2d4af9864dab  base
autoai-kb_3.1-py3.7             632d4b22-10aa-5180-88f0-f52dfb6444d7  base
------------------------------  ------------------------------------  ----
```

```
Note: Only first 50 records were displayed. To display more use 'limit' parameter.
```

In [29]:

```
software_space_uid = client.software_specifications.get_uid_by_name("tensorflow_rt22.1-py3.9")
software_space_uid
```

Out[29]:

```
'acd9c798-6974-5d2f-a657-ce06e986df4d'
```

In [30]:

```
model_details = client.repository.store_model(model="classify-arrhythmia-model_new.tgz", meta_props={
    client.repository.ModelMetaNames.NAME:"Arrhytmia",
    client.repository.ModelMetaNames.TYPE:"tensorflow_2.7",
    client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_space_uid
})
```

In [ ]:

TESTING

PERFORMANCE TESTING

Model Performance Testing:

S.No	Parameter	Values	Screenshot
1.	Model Summary	Total params: 21,885,485 Trainable params: 1,024,005 Non-trainable params: 20,861,480	batch_normalization_3 (BatchNo rmalization) (None, 10, 10, 1024) 4096 ['conv2d_3[0][0]'] add_11 (Add) (None, 10, 10, 1024) 0 ['block13_pool[0][0]', 'batch_normalization_3[0][0]'] block14_sepconv1 (SeparableCon v2D) (None, 10, 10, 1536) 1582080 ['add_11[0][0]'] block14_sepconv1_bn (BatchNorm alization) (None, 10, 10, 1536) 6144 ['block14_sepconv1[0][0]'] block14_sepconv1_act (Activati on) (None, 10, 10, 1536) 0 ['block14_sepconv1_bn[0][0]'] block14_sepconv2 (SeparableCon v2D) (None, 10, 10, 2048) 3159552 ['block14_sepconv1_act[0][0]'] block14_sepconv2_bn (BatchNorm alization) (None, 10, 10, 2048) 8192 ['block14_sepconv2[0][0]'] block14_sepconv2_act (Activati on) (None, 10, 10, 2048) 0 ['block14_sepconv2_bn[0][0]'] flatten (Flatten) (None, 204800) 0 ['block14_sepconv2_act[0][0]'] dense (Dense) (None, 5) 1024005 ['flatten[0][0]'] ========== Total params: 21,885,485 Trainable params: 1,024,005 Non-trainable params: 20,861,480
2.	Accuracy	Training Accuracy-0.7500 Validation Accuracy0.8009	- loss: 0.8009 - accuracy: 0.7500
3.	Confidence Score-(Only Yolo Projects)	–	–

USER ACCEPTANCE

1. Purpose of Document:-

This document serves as a quick reference for the Classification of Arrhythmia by Using Deep Learning with 2-D ECG Spectral Image Representation project's test coverage and open issues as of the project's release for user acceptance testing.

2. Defect Analysis:-

This shows how many bugs were fixed or closed at each severity level and how they were fixed.

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	4	5	3	3	15
Duplicate	1	0	2	0	3
External	1	3	0	1	5
Fixed	9	2	4	13	28
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	15	15	13	19	62

3. Test-Case Analysis

This report shows the number of test cases that have passed, failed, and untested.

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	9	0	0	9
Client Application	40	0	0	40
Security	2	0	0	2
Out-source Shipping	3	0	0	3
Exception Reporting	9	0	0	9
Final Report Output	4	0	0	4
Version Control	2	0	0	2

DEMO LINK

https://youtu.be/MiNC-OzxhLg