ECG- Image based Heartbeat classification for Arrhythmia Detection Using IBM Watson Studio

Team Members

Aswin S

Arunkumaar T S

Krithika Sree T

Anitha Lakshmi S

INTRODUCTION

1.1 Overview

An electrocardiogram (ECG) is a complete representation of the electrical activity of the heart on the surface of the human body, and it is extensively applied in the clinical diagnosis of heart diseases, it can be reliably used as a measure to monitor the functionality of the cardiovascular system. ECG signals have been widely used for detecting heart diseases due to its simplicity and non-invasive nature. Features of ECG signals can be computed from ECG samples and extracted using some softwares.

Arrhythmia is a representative type of Cardio Vascular Disease that refers to any irregular change from the normal heart rhythms. There are several types of arrhythmia including atrial fibrillation, premature contraction, ventricular fibrillation, and tachycardia. Although a single arrhythmia heartbeat may not have a serious impact on life, continuous arrhythmia beats can result in fatal circumstances.

1.2 Purpose

To detect ECG image based Heartbeat Classification for classifying several types of Arrhythmia using Convolution Neural Networks(CNN). The target of the classification process is to obtain an intelligent model, that is capable to class any heartbeat signal to specific type of Arrhythmia clasification. The final results shows that the project is more efficient in terms of accuracy and also competitive in terms of sensitivity and specificity.

Literature Survey
2.1 Existing Problem
Cardiologists use mostly the raw ECG to diagnose. The simplest and fastest method of feature extraction is then to extract sampled points from an ECG signal curve. However, one should be aware of the fact that the amount of the extracted features used to characterize the heartbeat can be a burden for the classification algorithm. For this reason, most of the works that use the raw signal perform a down sampling of the waveform or some feature selection in order to reduce the computation time. In order to circumvent this issue, a simple machine learning method is chosen to classify the arrhythmias.

2.2 Proposed Solution
In this project, we built an effective electrocardiogram (ECG) arrhythmia classification method 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. We have created a web application where the user selects the image which is to be classified. The image is fed into the model that is trained and the cited class will be displayed on the webpage. This simple machine learning method also allows a fast retraining of the classifier if new ECG data become available.

Theoritical Analysis
3.1 Block Diagram
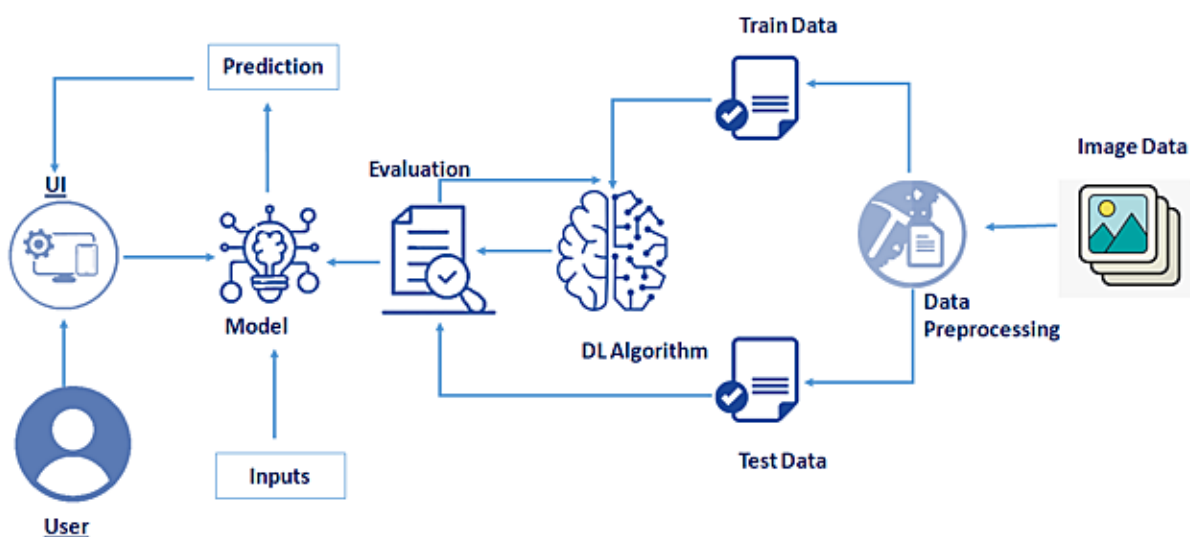
3.2 Hardware/Software Designing

The Softwares that is required for the project are Anaconda Navigator, Jupyter Notebook, Spyder and Notepad. The Skills that are Required for the project are Python, Python Web Frame Works, ANN, CNN, Tensorflow, Keras, Open CV. One should have prior knowledge on Supervised and unsupervised learning, Regression Classification and Clustering, Artificial Neural Networks, Convolution Neural Networks and Flask. To build a Deep learning models it required the following packages Tensorflow, Keras, Open CV and Flask.

3.2.1 Software Designing

Anaconda Navigator :

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and macOS. Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command-line interface (CLI).

Jupyter Notebook :

Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI). A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text mathematics, plots and rich media, usually ending with the ". ipynb" extension.

Tensorflow :

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML powered applications.

Keras :

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or Plaid ML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. 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.

Flask :

Flask is a microframework written in python. It is classified as a microframework because it does not require particular tools or libraries.It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object- relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools

3.2.2 Hardware designing

1. Processor: Intel core i5 or above.

2. 64-bit, quad-core, 2.5 GHz minimum per core

3. Ram: 8 GB or more

4. Hard disk: 10 GB of available space or more.

5. Display: Dual XGA (1024 x 768) or higher resolution monitors

6. Operating system: Window

Experimental Investigations

        The challenge in ECG classification is to handle the irregularities in the ECG signals which is very important to detect the patient status. Each heartbeat is a combination of action impulse waveforms produced by different specialized cardiac heart tissues. Heartbeats classification faces some difficulties because these waveforms differ from person to another, they are described by some features. These features are the inputs of machine learning algorithm.

        The project is designed in a way that first the User interacts with User interface to upload image, and then the Uploaded image is analyzed by the model which is integrated and finally Once the model analyses the uploaded image, the prediction is showcased on the UI.

To accomplish this, we have completed all the activities and tasks listed below:

- Data Collection.
    - Collect the dataset or Create the dataset
- Data Preprocessing.
    - Import the ImageDataGenerator library
    - Configure ImageDataGenerator class
    - Apply ImageDataGenerator functionality to Trainset and Testset
- Model Building
    - Import the model building Libraries
    - Initializing the model
    - Adding Input Layer
    - Adding Hidden Layer
    - Adding Output Layer
    - Configure the Learning Process
    - Training and testing the model
    - Optimize the Model
    - Save the Model
- Application Building
    - Create an HTML file
    - Build Python Code

4.1 Dataset collection :

The data set used has been downloaded from Kaggle which consists of nearly 250 images that are used to test and train the system.

4.2 Image pre-processing :

Image Pre-processing includes the following main tasks

Import ImageDataGenerator Library

The Pneumonia scanned image dataset has been downloaded from the Kaggle. The dataset consists of around 5000+ images, including infected and uninfected. These scanned images are taken as input to the primary step. The pre-processing is an essential and initial step in improving the quality of the scanned Image. The critical steps in preprocessing are the reduction of impulsive noises and image resizing. In the initial phase,ImageDataGenerator class is used to load the images with different modifications like considering the zoomed image, flipping the image and resizing the image to a range of 0 and 1.

Configure ImageDataGenerator Class.

In image processing, image acquisition is done by retrieving an image from a dataset for processing. It is the first step in the workflow sequence because, without an image no processing is possible. The image that is acquired is completely unprocessed. Here we process the image using the file path from the local device

Applying ImageDataGenerator functionality to the train set and test set. Specify the path of both the folders in the flow_from_directory method.

4.3 Model Building :

The neural network model is to be built by adding different network layers like convolution, pooling, flattening, dropout and neural layers.

For this step we need to import Keras and other packages that we're going to use in building the CNN. Import the following packages: Sequential is used to initialize the neural network. Adding a Convolution layer is used to make the convolutional network that deals with the images. Adding a Pooling layer is used to add the pooling layers. Flatten layer is the function that converts the pooled feature map to a single column that is passed to the fully connected layer. Adding a fully connected layer which includes a hidden layer to the neural network.

SEQUENTIAL:

To initialize the neural network, we create an object of the Sequential class. Ex; model=Sequential()

CONVOLUTION:

To add the convolution layer, we call the add function with the classifier object and pass in Convolution2D with parameters. The first argument is feature_detectors which is the number of feature detectors that we want to create. The second and third parameters are dimensions of the feature detector matrix. We used 256 feature detectors for CNNs. The next parameter is input shape which is the shape of the input image. The images will be converted into this shape during pre-processing. If the image is black and white it will be converted into a 2D array and if the image is coloured it will be converted into a 3D array. In this case, we'll assume that we are working with coloured images. Input_shape is passed in a tuple with the number of channels, which is 3 for a coloured image, and the dimensions of the 2D array in each channel. If you are not using a GPU it's advisable to use lower dimensions to reduce the computation time. The final parameter is the activation function. Classifying images is a nonlinear problem. So, we use the rectifier function to ensure that we don't have negative pixel values during computation.

Ex; model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))

POOLING:

The Pooling layer is responsible for reducing the spatial size of the convolved feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training the model.There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Generally, we use max pooling. In this step we reduce the size of the feature map. Generally, we create a pool size of 2x2 for max pooling. This enables us to reduce the size of the feature map while not losing important

image information.
Ex; model.add(MaxPooling2D(pool_size=(2,2)))

FLATTENING:

   In this step, all the pooled feature maps are taken and put into a single vector for inputting it to the next layer. The Flatten function flattens all the feature maps into a single long column. Ex; model.add(Flatten())

FULLY CONNECTION:

        The next step is to use the vector we obtained above as the input for the neural network by using the Dense function in Keras. The first parameter is output which is the number of nodes in the hidden layer. You can determine the most appropriate number through experimentation. The higher the number of dimensions the more computing resources you will need to fit the model. A common practice is to pick the number of nodes in powers of two. The next layer we have to add is the output layer. In this case, we'll use the sigmoid activation function since we expect a binary outcome. If we expected more than two outcomes, we would use the SoftMax function.

4.4 Test the model :

        The model is to be tested with different images to know if it is predicting correctly. . In split the data we set the image as 80% Training Data and 20% Testing Data. Then build CNN model train deep neural network for epochs


4.5 Application Building

        After the model is built, we will be integrating it into a web application so that normal users can also use it. The users need to give the scan to know if they are infected or not.

4.5 Test the Model in IBM

        At finally test the model in IBM using IBM Watson studio and then build a '.h5' using this file we can predict weather a patient is effected infected or not in web application using flask
        Finally , we detect whether the given Scanned image has infected or not using IBM Watson studio.
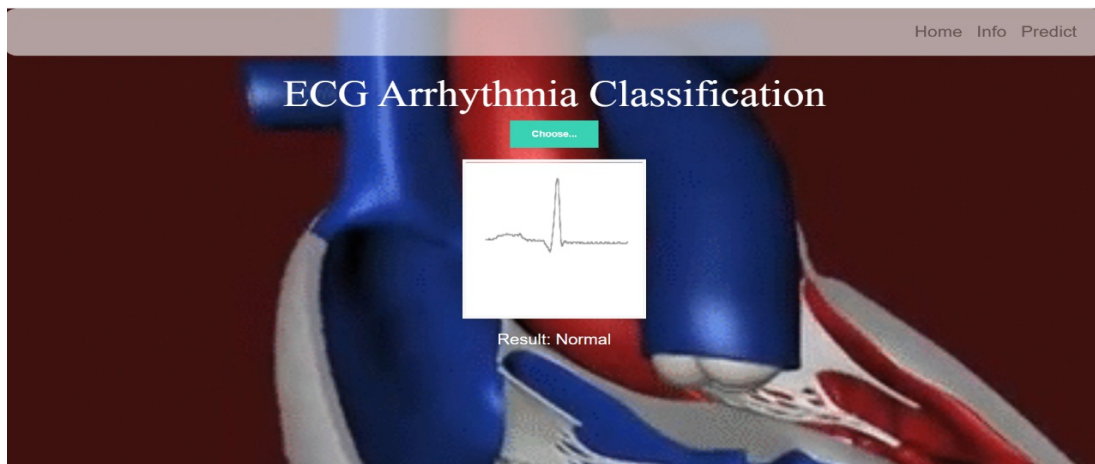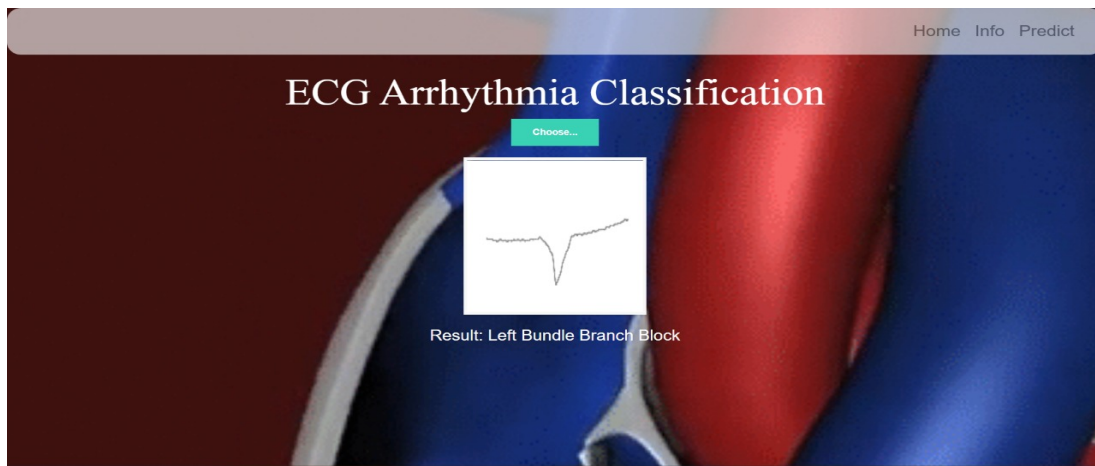
Flowchart
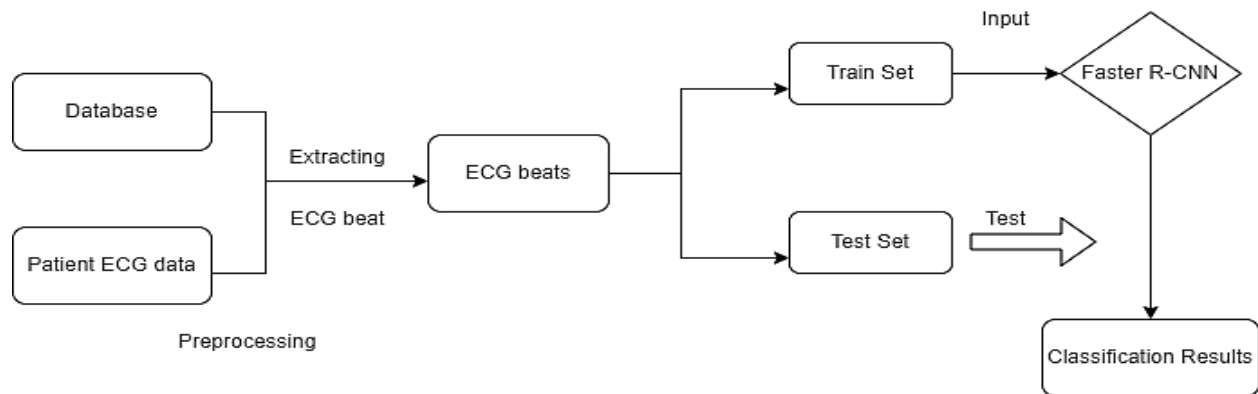
Result

From the project, we obtained the following output based on the classification.
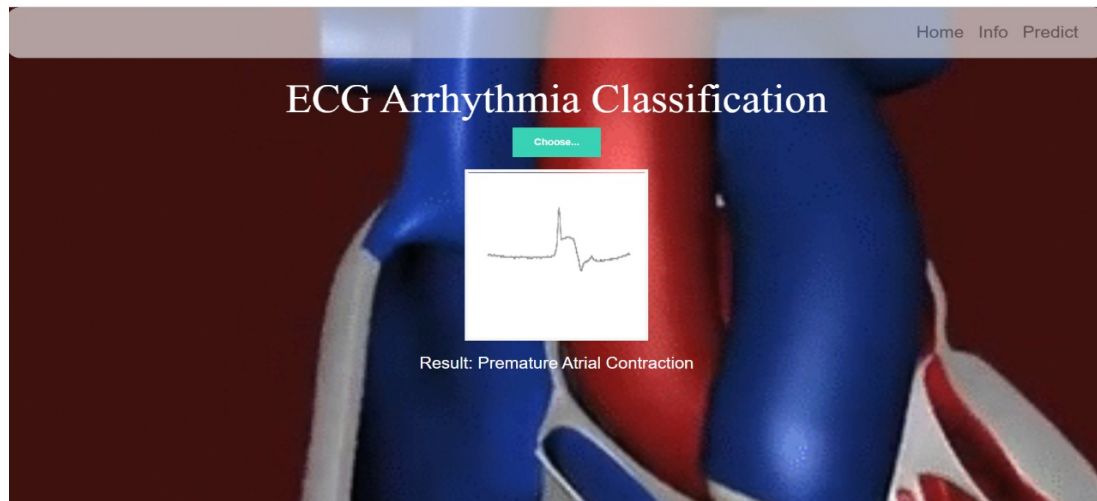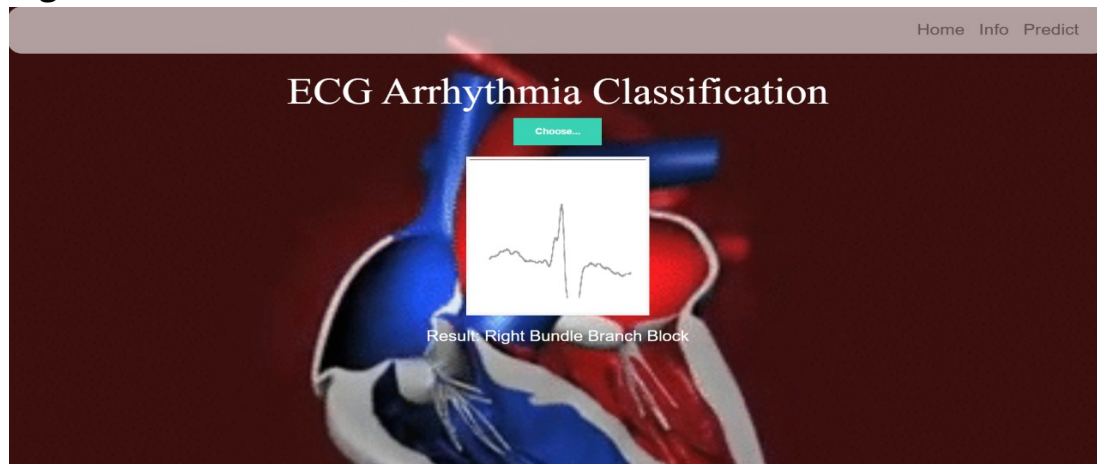
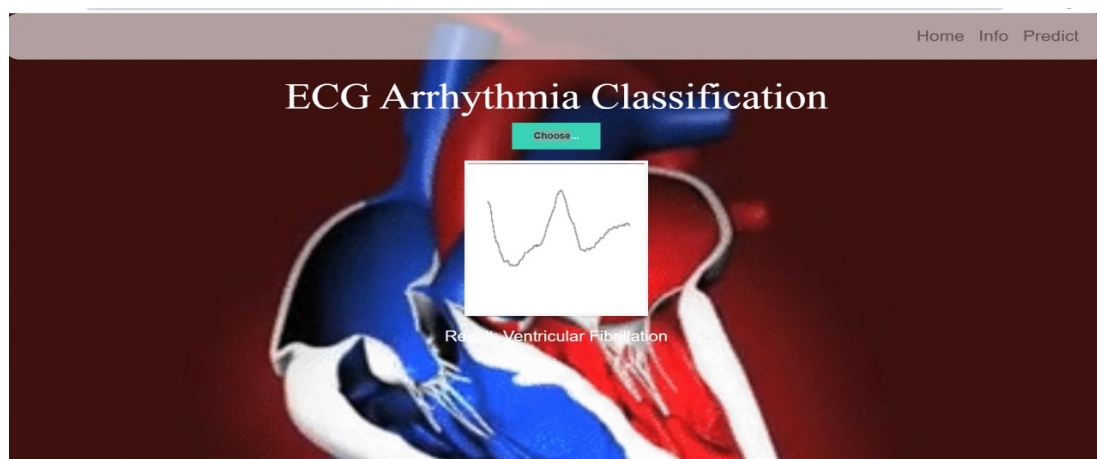### Normal

Left Bundel Branch Block
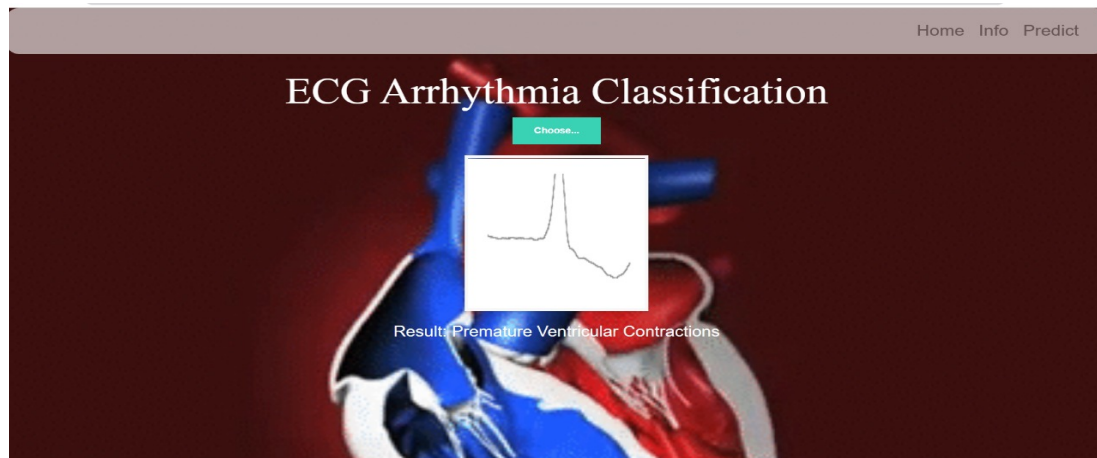
Premature Atrial Contraction

Right Bundle Branch Block

Ventricular Fibrillation

Premature Venticular Contraction

Advantages and Disadvantages

The advantage of this project is that the initiative for the early detection of diseases is a famous study and classification. The issues of biometric authentication and the application of emotional recognition can be resolved by various techniques, unlike heartbeat type detection.

The imbalance of the ECG dataset is a significant issue because certain classes have a lot of data relative to others, which might lead to false information about model performance.This is consider to be the disadavantage of the project

Applications

After the model is built, we will be integrating it into a web application so that normal users can also use it. The users need to give the image of the heartbeat to know if they arrhythmia or not.

Conclusion

In summary, This Project has validated an ability to classify heartbeats. Classification process is using some features of heartbeats and machine learning classification algorithms with local host pc working using only one node, which are crucial for diagnosis of cardiac arrhythmia.

Future Scope

The proposed model has the potential to be introduced into clinical settings as a helpful tool to aid the cardiologists in the reading of ECG heartbeat signals and to understand more about them.

Source Code
Flask Code

```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("upload.html")

@app.route("/predict",methods=["GET","POST"])
@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)
```

```python
        img=image.load_img(filepath,target_size=(64,64))
        x=image.img_to_array(img)
        x=np.expand_dims(x,axis=0)

        preds=model.predict(x)
        pred=np.argmax(preds,axis=1)
        print("prediction",pred)

        index=['Left Bundle Branch Block','Normal','Premature Atrial Contraction',
            'Premature Ventricular Contractions', 'Right Bundle Branch
Block','Ventricular Fibrillation']
        result=str(index[pred[0]])
        return result
    return None

if __name__=="__main__":
    app.run(debug=False)
```

Model Code
```python
model=Sequential()
# adding model layer
model.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
#convolutional layer
model.add(MaxPooling2D(pool_size=(2,2)))
#MaxPooling2D-for downsampling the input

model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())#flatten the dimension of the image
model.add(Dense(32))#deeply connected neural network layers.
model.add(Dense(6,activation='softmax'))#output layer with 6 neurons

model.summary()#summary of our model
```