1. Introduction

1.1 Overview

In this project we designed a convolution neural network model of ECG arrhythmia classification and deployed it through Flask. This model helps the user classify their type of arrhythmia using CNN, a deep-learning algorithms which helps us to classify images and analyze them.

1.2 Purpose

The primary purpose of the project was to produce a CNN model which would identify the 5 types of arrhythmia utilizing an image of the ECG diagram of the user.

2. Literature Survey

2.1 Present Problem

World Health association expresses that cardiovascular infections are the main sources of death (CVD's) on the planet . Around 17.9 million individuals passed on of CVD's in 2019, and primarily in low and middle income nations. Arrhythmia is a typical sort of CVD which causes around 15 - 20 % deaths all around the world, it is an issue which refers to unpredictable heartbeat rather than the normal one.

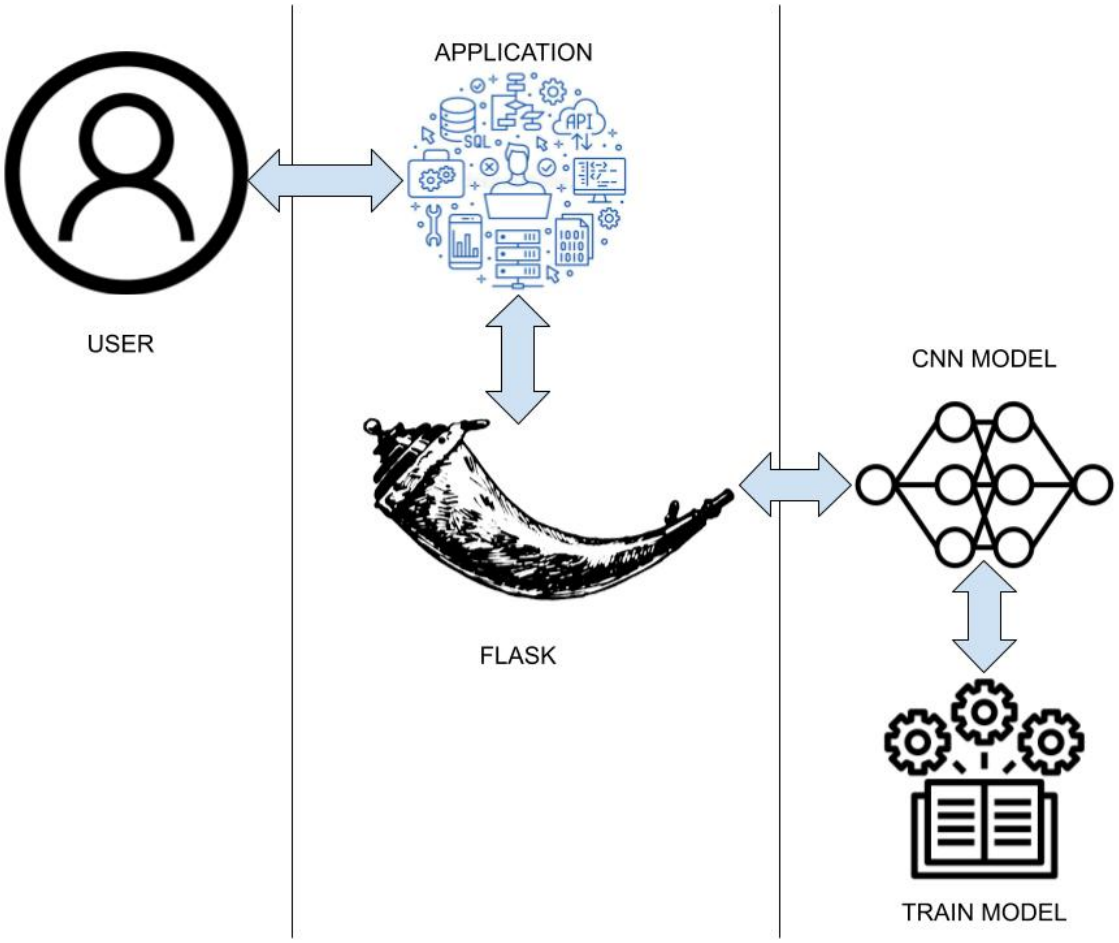
There are five kinds of Arrhythmia which the model will actually want to predict by utilizing the ECG, Electrocardiogram which is clinical device that shows the rhythms of the heart, utilizing this we can distinguish the anomalies in the beat and discover the problem.The five sorts of Arrhythmia are Left pack Branch Block, Premature Atrial Contraction, Premature Ventricular Contractions, Right Bundle Branch Block, Ventricular Fibrillation

2.2 Proposed Solution

In this project we propose an effective electrocardiogram(ECG) arrhythmia classification method using a deep two-dimensional convolutional neural network (CNN) , in which we classify ECG into six categories , one the normal rhythm and the other five the types of arrhythmia stated above. We will create a web application in which the user feeds an image of the ecg to the model and the predicted answer would be displayed on the screen.

3. Theoretical Analysis

3.1 Block Diagram



3.2 Hardware/ Software Requirements

The requirements are to install Latest version ofFlask, which is a python web framework, which would render our html template. The installation of of TensorFlow a package which helps us build neural network models, with the TensorFlow, keras is also required. Install the latest versions of all of these modules.

4. Experimental Investigations

Step 1- Data Collection

- Collected images of the 6 types of output in the model, Normal ,Left pack Branch Block, Premature Atrial Contraction, Premature Ventricular Contractions, Right Bundle Branch Block, Ventricular Fibrillation. Collected around total of around 22166 images

- Divided the data into Training and testing folders, in a ratio of 70:30, so 15341 images in training folder and 6825 images in testing folder.

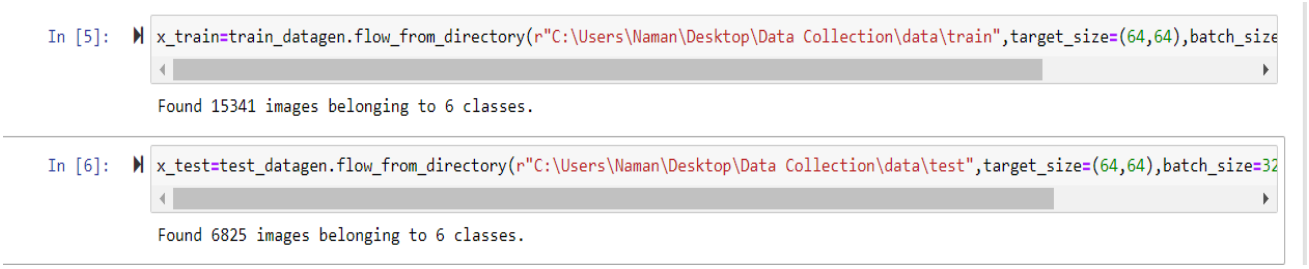
Step 2 -Data Pre processing

- Imported the ImageDataGenerator class

- Defined the parameters of ImageDataGenerator classes

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

- Applied ImageDataGenerator functionality to the train and test folders. Kept the target_size of the images 64*64, the batch size 32. The ImageDataGenerator class would apply a series of random transformations ,based on the parameters defined, on the batches of images.



Step -3 Model Building

Adding Convolution layers and the networked layer

- Import all the libraries required for the model building

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
```

- Initializing the model
  Model= Sequential()
  Sequential class is used to define linear initialization of network layers which then, collectively, constitute the model

- Adding All the CNN Layers
  Convolution Layer - This layer accepts the images as inputs and

then applies different feature detectors on them to capture the different features of the photo using there pixel values and returns the feature maps of the photo. The activation function of the Convolution layer is relu.

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

Max pooling Layers - It is used to tackle the problem of over-fitting, it takes all the feature maps and makes a grid of some of the main features of the photos

```
In [12]: model.add(MaxPooling2D(pool_size=(2,2)))
```

Flatten Layers- This layer is used to convert two dimensional layers to one dimensional so that the features could be accepted as an input.

```
In [13]: model.add(Flatten())
```

Adding the Dense Layers

- The Artificial Neural network needs to start so we need to add the Dense layers

```
In [14]: model.add(Dense(units=600,activation='relu'))
In [15]: model.add(Dense(units=600,activation='relu'))
In [16]: model.add(Dense(units=600,activation='relu'))
In [17]: model.add(Dense(units=6,activation='softmax'))
```

Step 4- Configuring the learning process

We have both training data and the model defined, so we now

have to configure the learning process, by using compile(), a method of sequential model class. Compilation would require the optimizer, a loss function and a list of metrics. Since we have a classification problem we will use Adam as our optimizer, categorical cross entropy loss function and accuracy as our metrics.

```
In [18]: model.compile(loss='categorical_crossentropy', optimizer='adam',metrics='accuracy')
```

Step 5- Train the model

Now we train the model by passing the data through it, the training is the done by iterating over the data. The time in which the training happens totally depends on the number of epochs you give.

```
In [20]: model.fit_generator(x_train, steps_per_epoch=50, epochs=40, validation_data=x_test, validation_steps=20)

C:\Users\Naman\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/40
50/50 [==============================] - 8s 165ms/step - loss: 0.1492 - accuracy: 0.9481 - val_loss: 0.3638 - val_accuracy: 0.8969
Epoch 2/40
50/50 [==============================] - 8s 160ms/step - loss: 0.1394 - accuracy: 0.9550 - val_loss: 0.4391 - val_accuracy: 0.8859
Epoch 3/40
50/50 [==============================] - 9s 176ms/step - loss: 0.1210 - accuracy: 0.9614 - val_loss: 0.3842 - val_accuracy: 0.9125
Epoch 4/40
50/50 [==============================] - 13s 253ms/step - loss: 0.1343 - accuracy: 0.9581 - val_loss: 0.4197 - val_accuracy: 0.8953
Epoch 5/40
50/50 [==============================] - 10s 204ms/step - loss: 0.1544 - accuracy: 0.9431 - val_loss: 0.2907 - val_accuracy: 0.9187
Epoch 6/40
```

Step 6- Save the model

Saving the model , so that it could be used for the flask app.

Step 7- Create An html

In the main project folder, we will create a templates folder

where we would save our html file. The HTML file would accept the image of the user and feed it to the flask for the prediction of the type of Arrhythmia. HTML file code is given in the last section

Step 8-Creating the flask app

- In python first we will load all the required libraries and the model

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

- Then we will route to our html page which we created earlier.

```
@app.route('/')
def upload():
    return render_template('index.html')
```

Using this we will render our html on the home page of our app.

- Showcasing the predictions on the UI

```
@app.route('/success',methods=['POST'])
def success():
    if request.method == 'POST':

        a = request.files['fileToUpload']
        a.save(a.filename)
        if a.filename.endswith('jpg') or a.filename.endswith('png') or a.filename.endswith('jpeg'):

            name=image.load_img(a.filename, target_size=(64,64))
            x=image.img_to_array(name)


            x=np.expand_dims(x,axis=0)
            pred=model.predict_classes(x)
            index=['Left Bundle Branch Block', 'Normal', 'Premature Atrial Contraction','Premature Ventricular Contractions','Right Bundle
            answer=index[pred[0]]


            return render_template('index.html', prediction_text='Type {}'.format(answer))
        else:
            return render_template('index.html', prediction_text='Type {}'.format('Error Use image folder'))
```

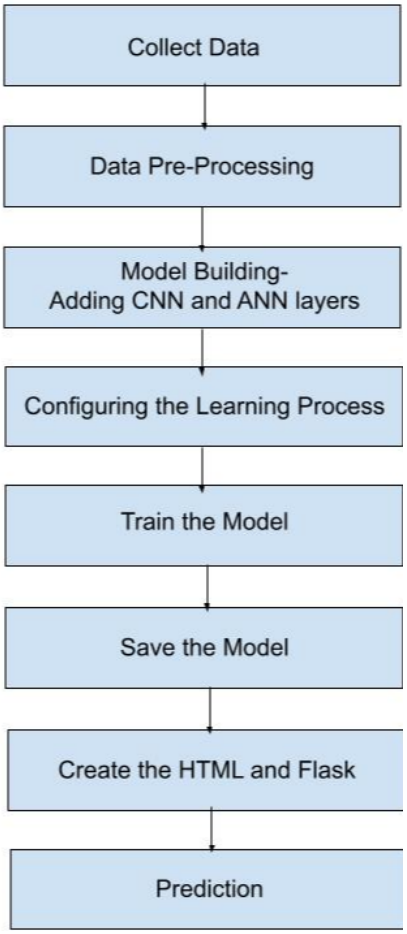
Here we are defining a function which would request the image file and would feed it to the model, by converting it into array format and by shaping it the required input size. The model would predict the answer and the answer would then be returned

to the html page so that it could display it.

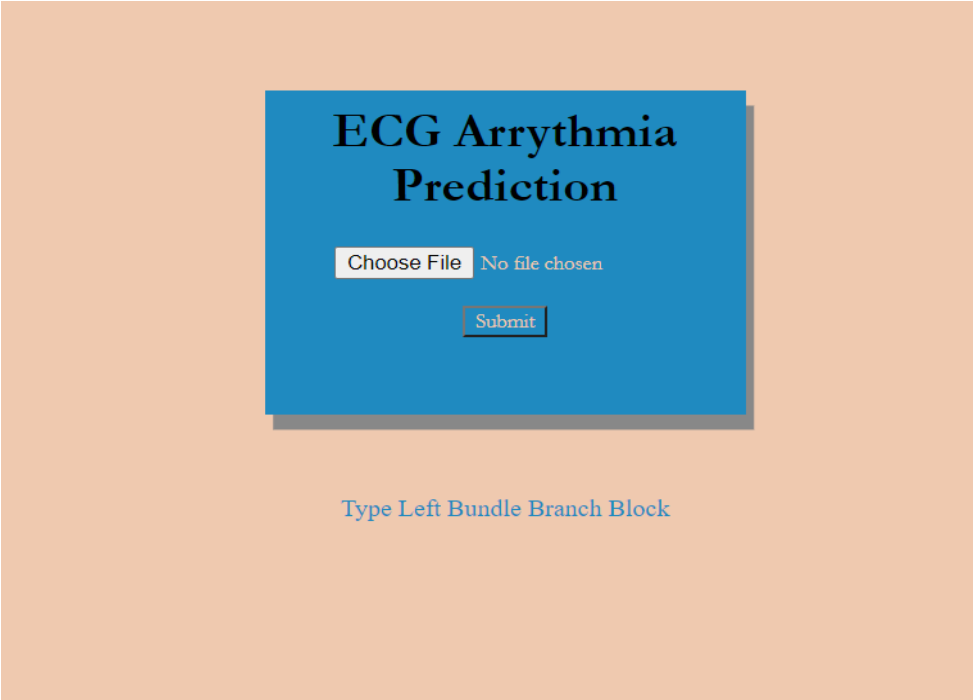
- The main function is then used to run the app on the local host

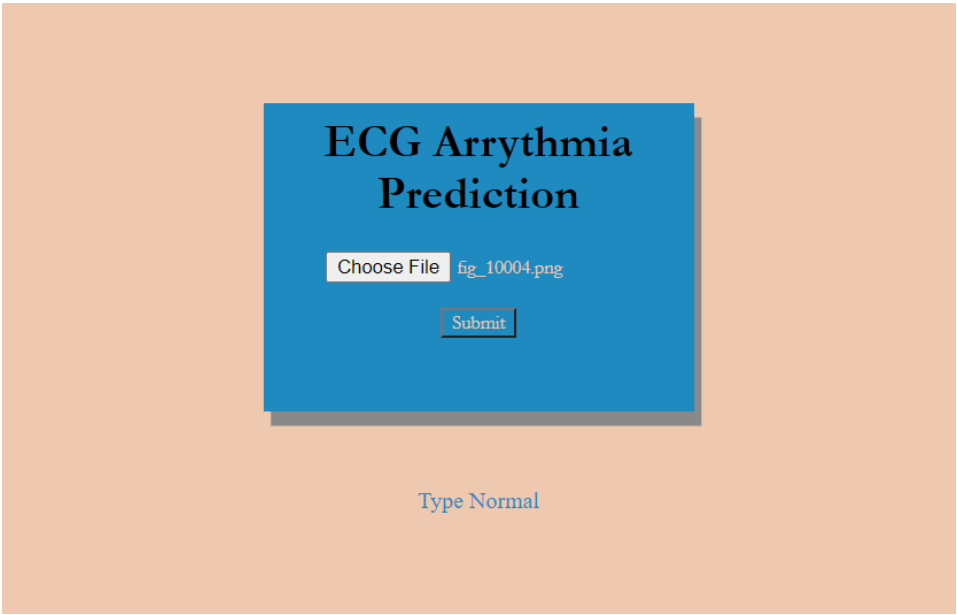
```
if __name__ == "__main__":
    #app.debug = True

    app.run()
```

5. Flow Chart



6. Results


ECG Arrythmia Prediction
Choose File
No file chosen
Submit
Type Left Bundle Branch Block


ECG Arrythmia Prediction
Choose File
fig_10004.png
Submit
Type Normal

7. Advantages and Disadvantages

Advantages

1. Quicker Detection of the kind of Arrhythmia
2. Can help patients Identify the issue on their own
3. Can decrease the responsibility of the specialists, who physically attempt to recognize the Arrhythmia

Disadvantages

1. Less Accurate
2. Due to inaccuracy, can lead to wrong ouputs and henceworth can be harmful for the patient.

8. Applications

The ECG Arrhythmia classfication model could help specialists in different medical clinics, where they could easily recognize the kind of Arrhythmia utilizing the application and Cure the patient quickly.This app would help increase the pace of the treatment and would also reduce the burden of the specialist

9. Conclusion

Thus by using CNN and Flask we have been able develop a model which could successfully predicts the type of Arrhythmia the patient has. This could also get more accurate by adding more data to the training and testing folder, the more you train and test the model the better it will get.

10. Future Scope

As we collect more and more data, a more accurate model can be made which is more trained and this technique could also be used to solve other problems with CVD's. Using the data collected by the CNN model we could predict what the future functioning of the heart would be and if there is a problem we could try to solve it.

11. Bibliography

1. https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)
2. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6020177/#:~:text=Sudden%20cardiac%20death%20(SCD)%20and,%E2%80%9320%20%25%20of%20all%20deaths.
3. https://www.tensorflow.org/install
4. https://pypi.org/project/keras/
5. https://pypi.org/project/Flask/
6. https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
7. Github Repo- https://github.com/namanndjain9/Smart-Internz-Project-ECG-Arrhythmia-Classfication

12. Appendix

12.1 Source Code

a) Training the model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
```

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
train_datagen=ImageDataGenerator(rescale=1./255,zoom_range=0.2,
horizontal_flip=True,vertical_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)

x_train=train_datagen.flow_from_directory(r"C:\Users\Naman\Desktop
\Data
Collection\data\train",target_size=(64,64),batch_size=32,class_mode=
"categorical")

x_test=test_datagen.flow_from_directory(r"C:\Users\Naman\Desktop\
Data
Collection\data\test",target_size=(64,64),batch_size=32,class_mode="
categorical")

from tensorflow.keras.layers import Dense,Convolution2D,
MaxPooling2D, Flatten

from tensorflow.keras.models import Sequential
model=Sequential()
model.add(Convolution2D(32,(3,3), input_shape=(64,64,3),
activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(units=600,activation='relu'))
model.add(Dense(units=600,activation='relu'))
model.add(Dense(units=600,activation='relu'))
```

```
model.add(Dense(units=6,activation='softmax'))
model.compile(loss='categorical_crossentropy',
optimizer='adam',metrics='accuracy')
model.fit_generator(x_train, steps_per_epoch=50, epochs=40,
validation_data=x_test, validation_steps=20)
model.save('project.h5')
```

b) Flask Code Python

```
from flask import Flask, request, render_template
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

model=load_model('project.h5')
app = Flask(__name__)



@app.route('/')
def upload():
    return render_template('index.html')

@app.route('/success',methods=['POST'])
def success():
    if request.method == 'POST':

        a = request.files['fileToUpload']
        a.save(a.filename)
        if a.filename.endswith('jpg') or a.filename.endswith('png') or
a.filename.endswith('jpeg'):
```

```
        name=image.load_img(a.filename, target_size=(64,64))
        x=image.img_to_array(name)



        x=np.expand_dims(x,axis=0)
        pred=model.predict_classes(x)
        index=['Left Bundle Branch Block', 'Normal', 'Premature Atrial
Contraction','Premature Ventricular Contractions','Right Bundle Branch
Block','Ventricular Fibrillation']
        answer=index[pred[0]]



        return render_template('index.html', prediction_text='Type
{}'.format(answer))
    else:
        return render_template('index.html', prediction_text='Type
{}'.format('Error Use image folder'))




if __name__ == "__main__":
    #app.debug = True

    app.run()
```

c) HTML Template

```
<html lang="en" >
<head>
  <meta charset="UTF-8">
  <title>Arrhythmia Prediction</title>
  <link rel="stylesheet" href="./style.css">

</head>
<style>
body{
    background-color: #efc9af

}
#demobox{
    background-color: #1f8ac0 ;
    border:  black;
    width: 300px;
    height: 215px;
    margin:auto;
    font-family: garamond;
    box-shadow: 5px 10px #888888;
```

```
}
input{
    font-family:garamond;
    color:#efc9af;
    background-color:#1f8ac0;
}
h1{
    padding-top: 10px;
}
p{
    color:#1f8ac0;

}

</style>
<!-- partial:index.partial.html -->

<div id ="demobox" style="margin-top: 250px" >
<h1 style="text-align:center" style="justify-content:center">ECG
Arrythmia Prediction </h1>

    <!-- Main Input For Receiving Query to our ML -->

   <form style="text-align:center" style="justify-content:center"
action="/success" method="post" enctype="multipart/form-data">

     <input type="file" name="fileToUpload" id="fileToUpload"
accept="image/*" >
```

```
    <br></br>

   <input type = "submit"  value="Submit" >

    </form>
    </div>
   <br>
   <br>
   <p style="text-align:center" style="justify-content:center">{{
prediction_text }}</p>


<!-- partial -->


</body>
</html>
```

12.2 UI output Screenshot