Date	18 October 2022
Team ID	PNT2022TMID24237
Project Name	Classification of Arrhythmia by using Deep Learning with 2-D ECG spectral image representation
Maximum Marks	

PROJECT PLANNING

TITLE OF THE PROJECT

Classification of Arrhythmia by using Deep Learning with 2-D ECG Spectral Image Representation

PROJECT OVERVIEW

According to the World Health Organization (WHO), cardiovascular diseases (CVDs) are the number one cause of death today. Over 17.7 million people died from CVDs in the year 2017 all over the world which is about 31% of all deaths, and over 75% of these deaths occur in low and middle-income countries. Arrhythmia is a representative type of CVD 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. In this project, we build 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 are creating 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.

OBJECTIVES

By the end of this project you will:

- Know fundamental concepts and techniques of the Artificial Neural Network and Convolution Neural Networks
- Gain a broad understanding of image data.
- Work with Sequential type of modeling
- Work with Keras capabilities
- Work with image processing techniques
- know how to build a web application using the Flask framework.

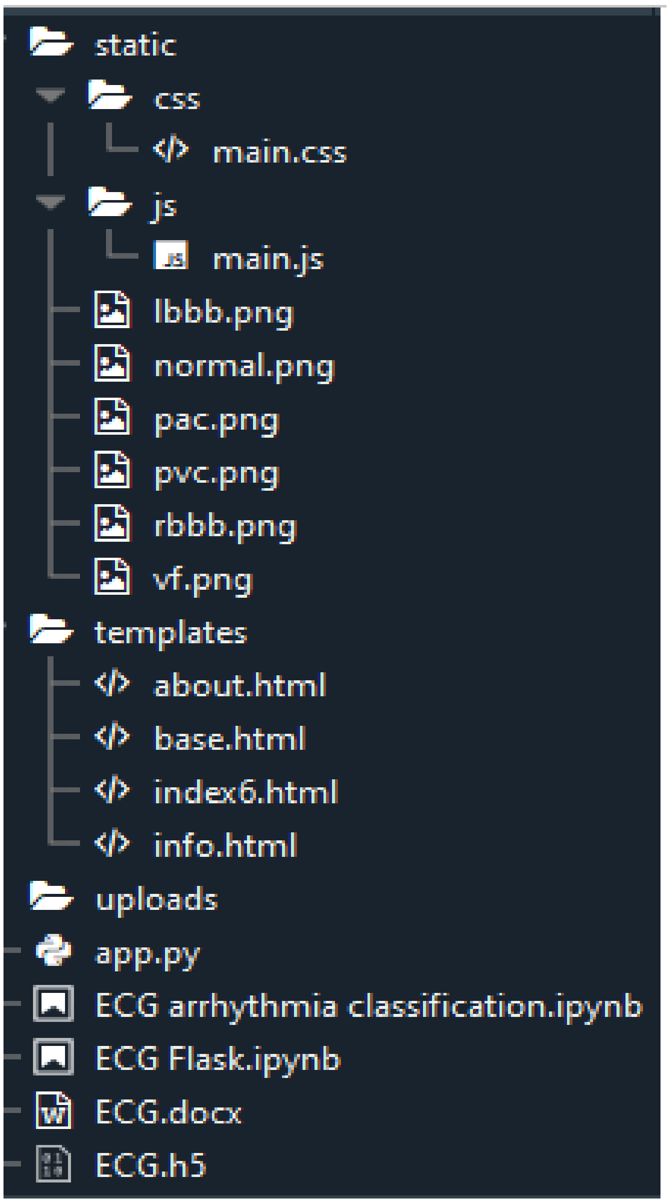
PROJECT FLOW

- User interacts with User interface to upload image
- Uploaded image is analyzed by the model which is integrated
- Once model analyses the uploaded image, the prediction is showcased on the UI

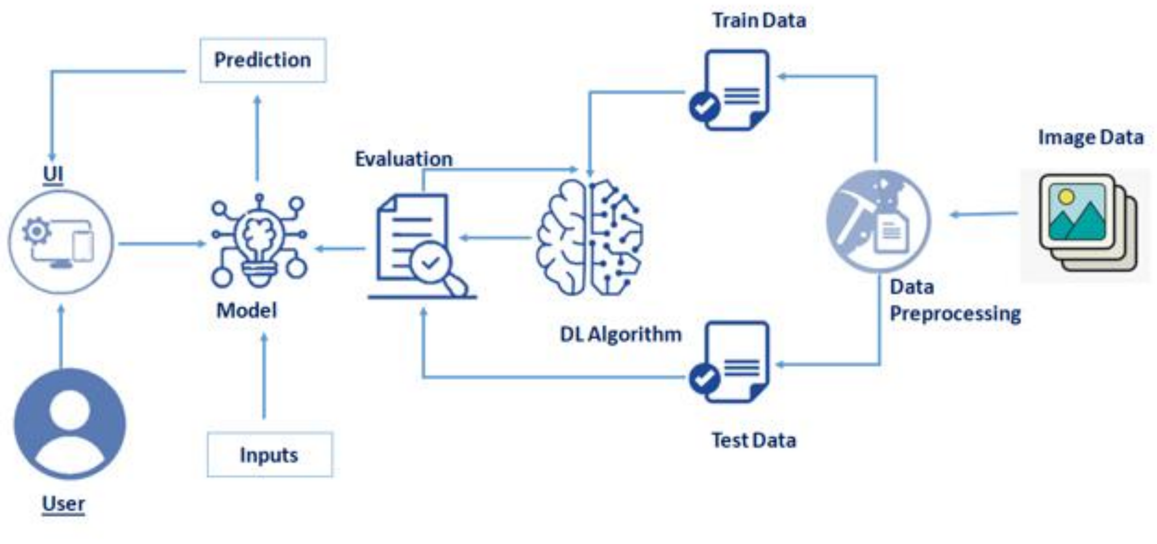
To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
    - Collect the dataset or Create-the dataset
- Data Pre-processing.
    - Import the Image Data Generator library
    - Configure Image Data Generator class
    - Apply Image Data Generator 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

PROJECT STRUCTURES


```
static
  css
    main.css
  js
    main.js
  lbbb.png
  normal.png
  pac.png
  pvc.png
  rbbb.png
  vf.png
templates
  about.html
  base.html
  index6.html
  info.html
uploads
app.py
ECG arrhythmia classification.ipynb
ECG Flask.ipynb
ECG.docx
ECG.h5
```

- We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for serverside scripting
- we need the model which is saved and the saved model in this content is ECG.h5
- The static folder will contain js and CSS files.
- Whenever we upload an image to predict, that images are saved in the uploads folder.
- Download the project files from this link



PREREQUISITES

To complete this project you should have the following software and packages

Anaconda Navigator:

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupiter notebook and spyder

To build Deep learning models you must require the following packages

Tensor flow: 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 leverages various optimization techniques to make high level neural network API easier and more performant. It supports the following features:

- Consistent, simple and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is user friendly framework which runs on both CPU and GPU.
- Highly scalability of computation.

Flask: Web frame work used for building Web applications

PRIOR KNOWLEDGE

One should have knowledge on the following Concepts:

- **Supervised and unsupervised learning**
- **Regression Classification and Clustering**
- **Artificial Neural Networks**
- **Convolution Neural Networks**
- **Flask**

REQUIREMENTS/TASKS

DATASET COLLECTION

Artificial Intelligence is a data hunger technology, it depends heavily on data, without data, it is impossible for a machine to learn. It is the most crucial aspect that makes algorithm training possible. In Convolutional Neural Networks, as it deals with images, we need training and testing data set. It is the actual data set used to train the model for performing various actions. In this activity lets focus of gathering the dataset

IMAGE PREPROCESSING

Image Pre-processing includes the following main tasks

- Import ImageDataGenerator Library.

- Configure ImageDataGenerator Class.
- Applying ImageDataGenerator functionality to the trainset and test set.

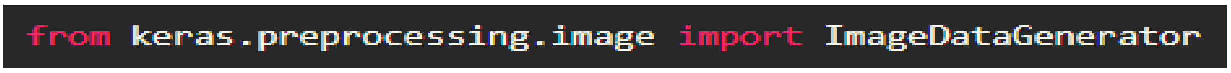
Note: The ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data.

IMPORT THE IMAGE DATA GENERATOR LIBRARY

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from Keras



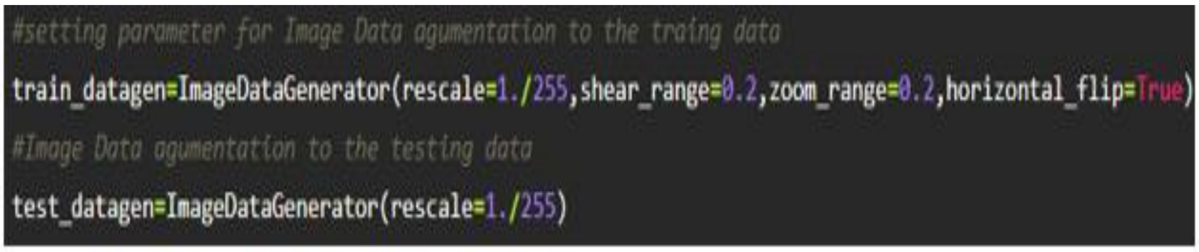
```
from keras.preprocessing.image import ImageDataGenerator
```

CONFIGURE IMAGE DATA GENERATION CLASS

There are five main types of data augmentation techniques for image data; specifically:

- Image shifts via the width_shift_range and height_shift_range arguments.
- Image flips via the horizontal_flip and vertical_flip arguments.
- Image rotates via the rotation_range argument
- Image brightness via the brightness_range argument.
- Image zooms via the zoom_range argument.

An instance of the ImageDataGenerator class can be constructed for train and test.

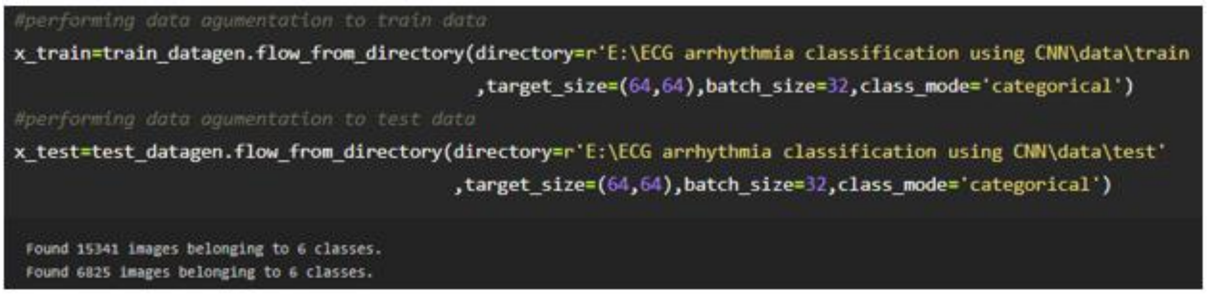


```
#setting parameter for Image Data agumentation to the traing data
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
#Image Data agumentation to the testing data
test_datagen=ImageDataGenerator(rescale=1./255)
```

APPLY IMAGE DATA GENERATOR FUNCTIONALITY TO TRAINSET AND TESTSET

Let us apply ImageDataGenerator functionality to Train set and Test set by using the following code

This function will return batches of images from the subdirectories Left Bundle Branch Block, Normal, Premature Atrial Contraction, Premature Ventricular Contractions, Right Bundle Branch Block and Ventricular Fibrillation, together with labels 0 to 5{'Left Bundle Branch Block': 0, 'Normal': 1, 'Premature Atrial Contraction': 2, 'Premature Ventricular Contractions': 3, 'Right Bundle Branch Block': 4, 'Ventricular Fibrillation': 5}



```
#performing data agumentation to train data
x_train=train_datagen.flow_from_directory(directory=r'E:\ECG arrhythmia classification using CNN\data\train
                                          ,target_size=(64,64),batch_size=32,class_mode='categorical')
#performing data agumentation to test data
x_test=test_datagen.flow_from_directory(directory=r'E:\ECG arrhythmia classification using CNN\data\test'
                                        ,target_size=(64,64),batch_size=32,class_mode='categorical')

Found 15341 images belonging to 6 classes.
Found 6825 images belonging to 6 classes.
```

We can see that for training there are 15341 images belonging to 6 classes and for testing there are 6825 images belonging to 6 classes.

Arguments:

- directory: Directory where the data is located. If labels is "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data. Default: 32.
- target_size: Size to resize images to after they are read from disk.
- class_mode:

- ‘int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss).
- 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).
- 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
- None (no labels).

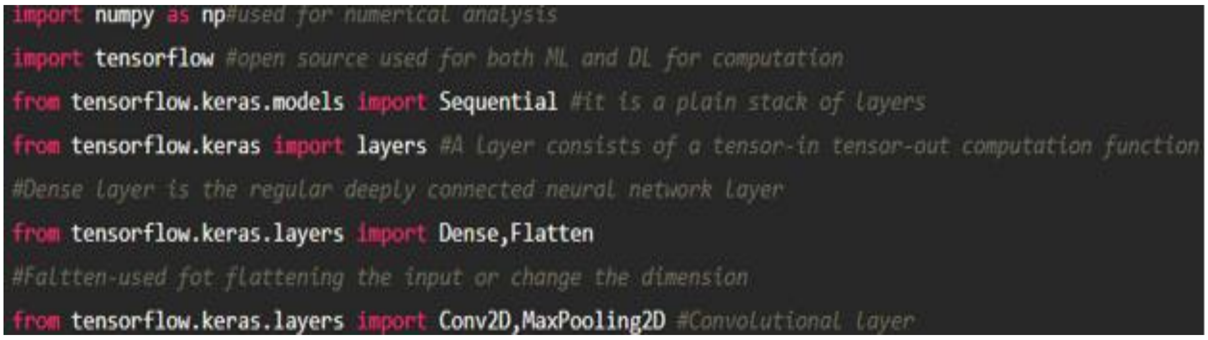
MODEL BULIDING

We are ready with the augmented and pre-processed image data, Lets begin our model building, this activity includes the following steps

- Import the model building Libraries
- Initializing the model
- Adding CNN Layers
- Adding Hidden Layer
- Adding Output Layer
- Configure the Learning Process
- Training and testing the model
- Saving the model

IMPORT THE LIBRARIES

This is a very crucial step in our deep learning model building process. We have to define how our model will look and that requires.



```
import numpy as np#used for numerical analysis
import tensorflow #open source used for both ML and DL for computation
from tensorflow.keras.models import Sequential #it is a plain stack of layers
from tensorflow.keras import layers #A layer consists of a tensor-in tensor-out computation function
#Dense layer is the regular deeply connected neural network layer
from tensorflow.keras.layers import Dense,Flatten
#Faltten-used fot flattening the input or change the dimension
from tensorflow.keras.layers import Conv2D,MaxPooling2D #Convolutional layer
```

INITIALIZE THE MODEL

Keras has 2 ways to define a neural network:

- Sequential
- Function API

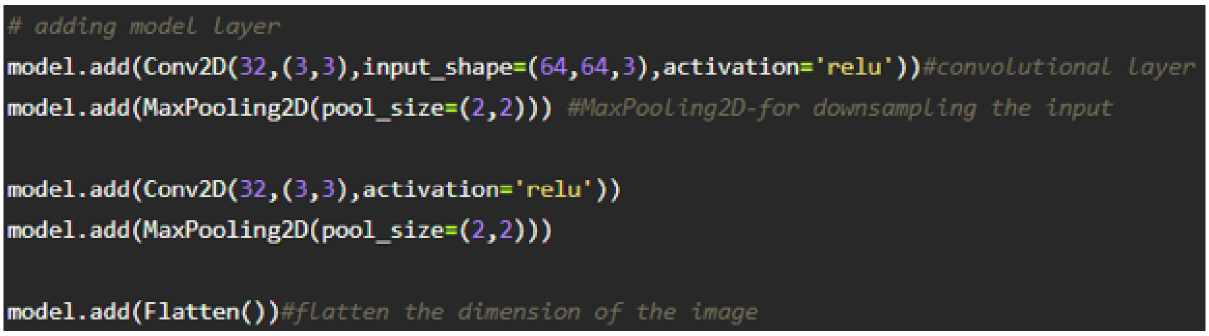
The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add () method.
Now, will initialize our model.

ADDING CNN LAYERS

We are adding a **convolution layer** with an activation function as “relu” and with a small filter size (3,3) and a number of filters as (32) followed by a max-pooling layer.

The **Max pool layer** is used to down sample the input.

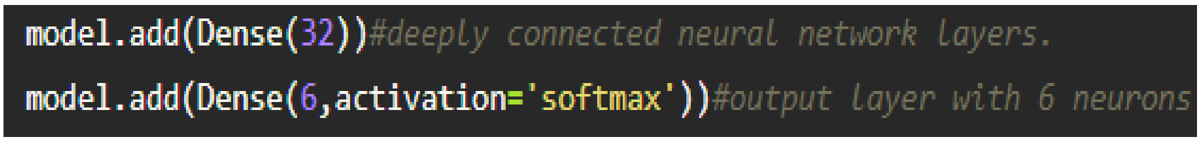
The **flatten layer** flattens the input.



```
# 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
```

ADDING DENSE LAYERS

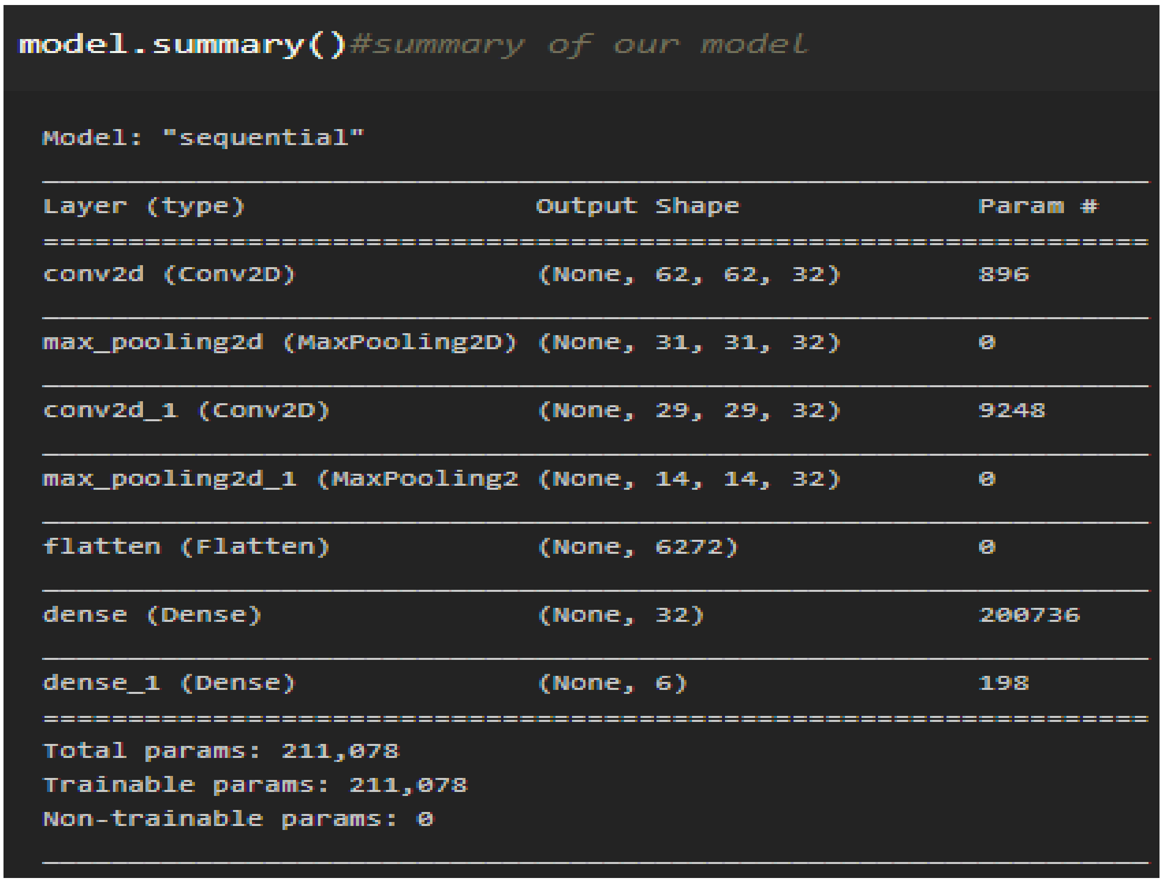
Dense layer is deeply connected neural network layer. It is most common and frequently used layer.



```
model.add(Dense(32))#deeply connected neural network layers.
model.add(Dense(6,activation='softmax'))#output layer with 6 neurons
```

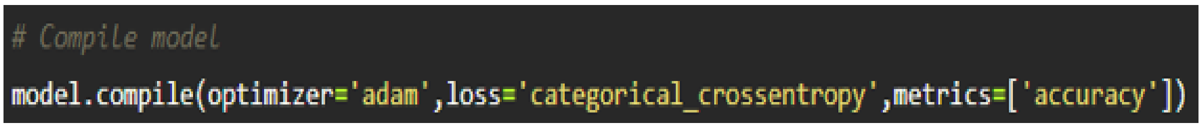
We have 6 neurons in op layer as we have considered 6 classes

Understanding the model is very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.


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

  Model: "sequential"
  _________________________________________________________________
  Layer (type)                 Output Shape              Param #
  =================================================================
  conv2d (Conv2D)              (None, 62, 62, 32)        896
  _________________________________________________________________
  max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
  _________________________________________________________________
  conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248
  _________________________________________________________________
  max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
  _________________________________________________________________
  flatten (Flatten)            (None, 6272)              0
  _________________________________________________________________
  dense (Dense)                (None, 32)                200736
  _________________________________________________________________
  dense_1 (Dense)              (None, 6)                 198
  =================================================================
  Total params: 211,078
  Trainable params: 211,078
  Non-trainable params: 0
  _________________________________________________________________
```

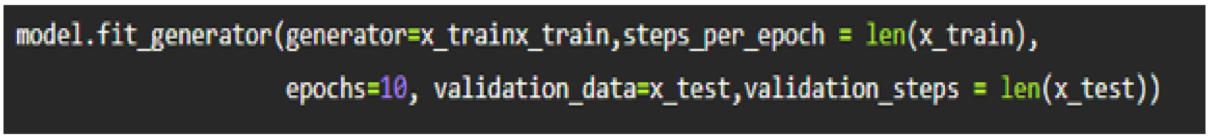
CONFIGURE THE LEARNING PROCESS

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find error or deviation in the learning process. Keras requires loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in the training process


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

TRAIN THE MODEL

Now, let us train our model with our image dataset. fit_generator functions used to train a deep learning neural network



```
model.fit_generator(generator=x_trainx_train,steps_per_epoch = len(x_train),
                    epochs=10, validation_data=x_test,validation_steps = len(x_test))
```

steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your train dataset divided by the batch size.

Epochs: an integer and number of epochs we want to train our model for.

Validation_data can be either:

- an inputs and targets list
- a generator
- an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

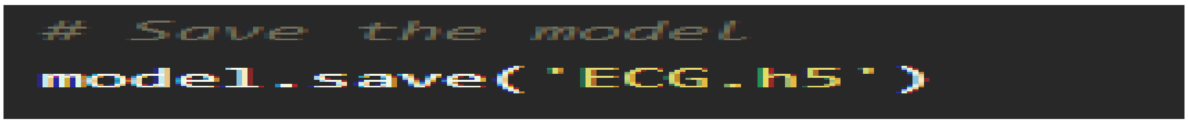
validation_steps: only if the **validation_data** is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your test dataset divided by the validation batch size.

Note: it will take time for training your data based on epochs

SAVE THE MODEL

The model is saved with .h5 extension as follows

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

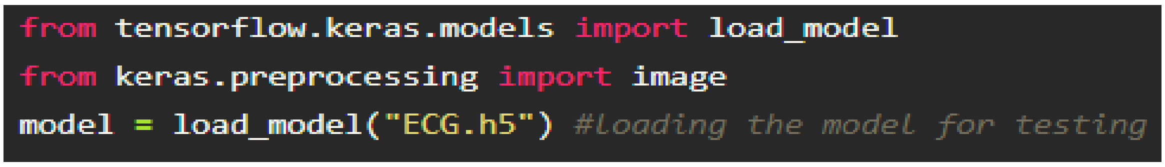


```
# Save the model
model.save('ECG.h5')
```

TEST THE MODEL

Open the other jupyter file and write the code mentioned below

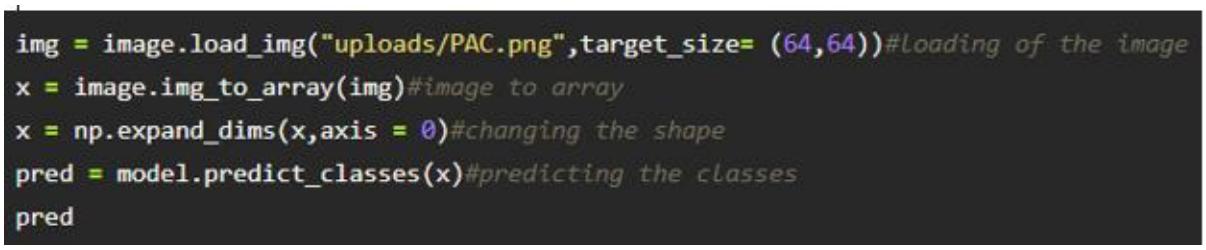
Load necessary libraries, Load the saved model using load_model



```
from tensorflow.keras.models import load_model
from keras.preprocessing import image
model = load_model("ECG.h5") #loading the model for testing
```

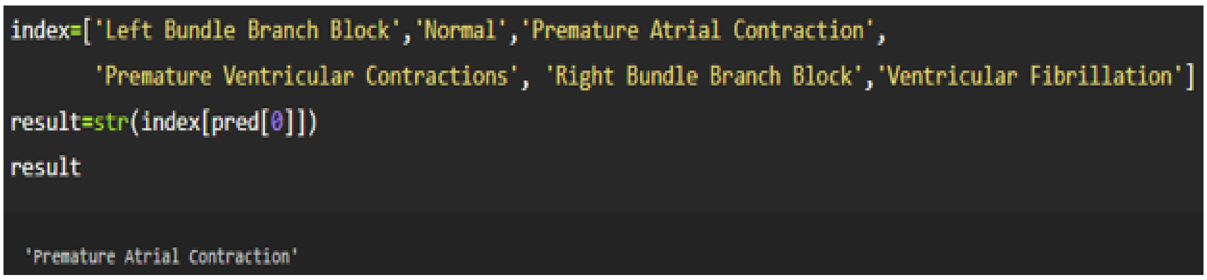
Taking an image as input and checking the results

Note the target size should for the image that is should be the same as the target size that you have used for training



```
img = image.load_img("uploads/PAC.png",target_size= (64,64))#loading of the image
x = image.img_to_array(img)#image to array
x = np.expand_dims(x,axis = 0)#changing the shape
pred = model.predict_classes(x)#predicting the classes
pred
```

By using the model we are predicting the output for the given input images



```
index=['Left Bundle Branch Block','Normal','Premature Atrial Contraction',
       'Premature Ventricular Contractions', 'Right Bundle Branch Block','Ventricular Fibrillation']
result=str(index[pred[0]])
result

 'Premature Atrial Contraction'
```

The predicted class index name will be printed here.

APPLICATION BUILDING

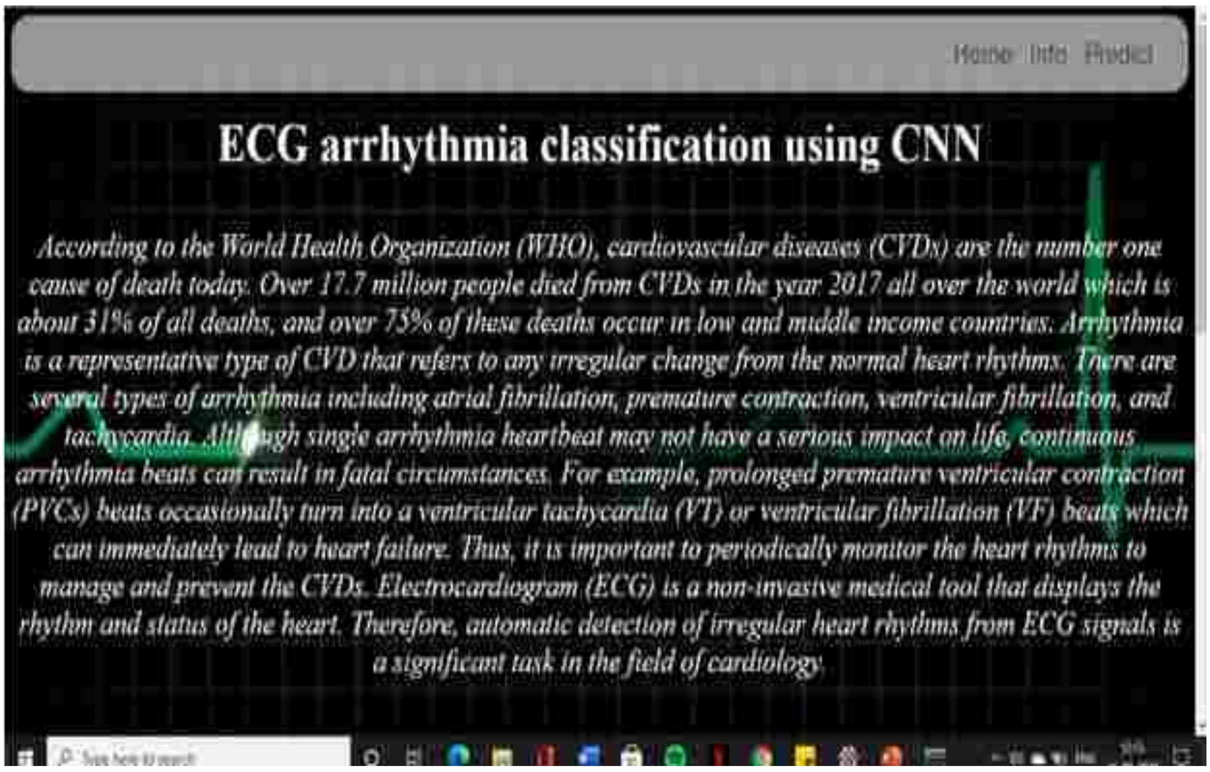
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has uploaded an image. The uploaded image is given to the saved model and prediction is showcased on the UI.
This section has the following tasks

- Building HTML Pages
- Building server-side script

CREATE HTML FILES

- We use HTML to create the front end part of the web page.
- Here, we created 4 html pages- about.html, base.html, index6.html, info.html.
- about.html displays the home page.
- Info.html displays all important details to be known about ECG.
- base.html and index6.html accept input from the user and predicts the values.

Your Home page looks like



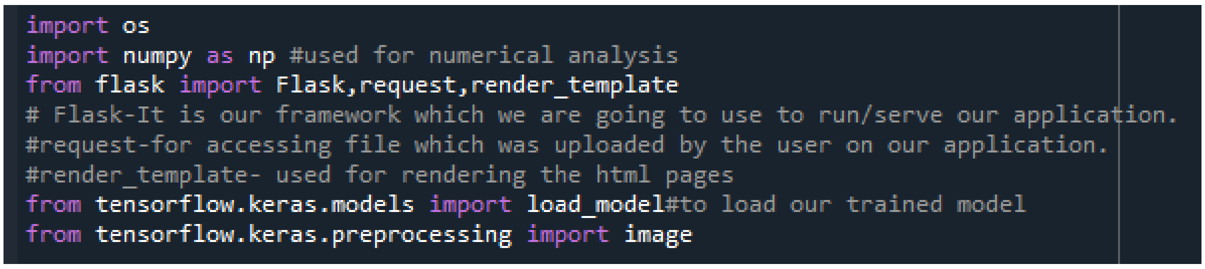
ECG arrhythmia classification using CNN

According to the World Health Organization (WHO), cardiovascular diseases (CVDs) are the number one cause of death today. Over 17.7 million people died from CVDs in the year 2017 all over the world which is about 31% of all deaths, and over 75% of these deaths occur in low and middle income countries. Arrhythmia is a representative type of CVD 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 single arrhythmia heartbeat may not have a serious impact on life, continuous arrhythmia beats can result in fatal circumstances. For example, prolonged premature ventricular contraction (PVCs) beats occasionally turn into a ventricular tachycardia (VT) or ventricular fibrillation (VF) beats which can immediately lead to heart failure. Thus, it is important to periodically monitor the heart rhythms to manage and prevent the CVDs. Electrocardiogram (ECG) is a non-invasive medical tool that displays the rhythm and status of the heart. Therefore, automatic detection of irregular heart rhythms from ECG signals is a significant task in the field of cardiology.

BUILD PYTHON CODE

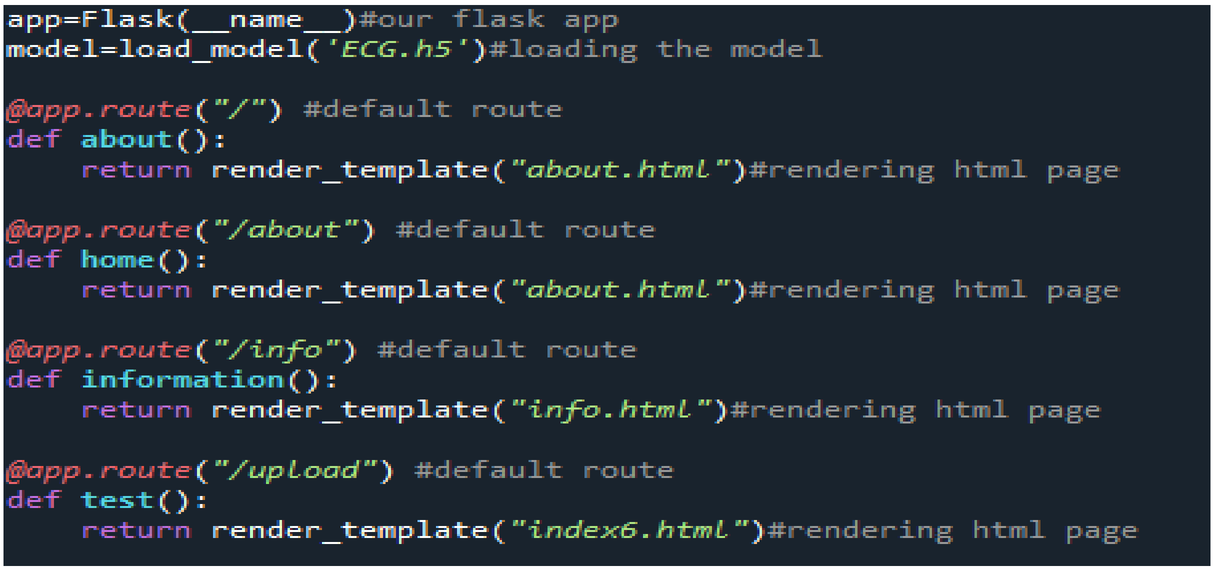
- Let us build the flask file ‘app.py’ which is a web framework written in python for server-side scripting. Let’s see step by step procedure for building the backend application.
- The app starts running when the “__name__” constructor is called in main.
- render_template is used to return HTML file.
- “GET” method is used to take input from the user.
- “POST” method is used to display the output to the user.

Import the libraries



```
import os
import numpy as np #used for numerical analysis
from flask import Flask,request,render_template
# Flask-It is our framework which we are going to use to run/serve our application.
#request-for accessing file which was uploaded by the user on our application.
#render_template- used for rendering the html pages
from tensorflow.keras.models import load_model#to load our trained model
from tensorflow.keras.preprocessing import image
```

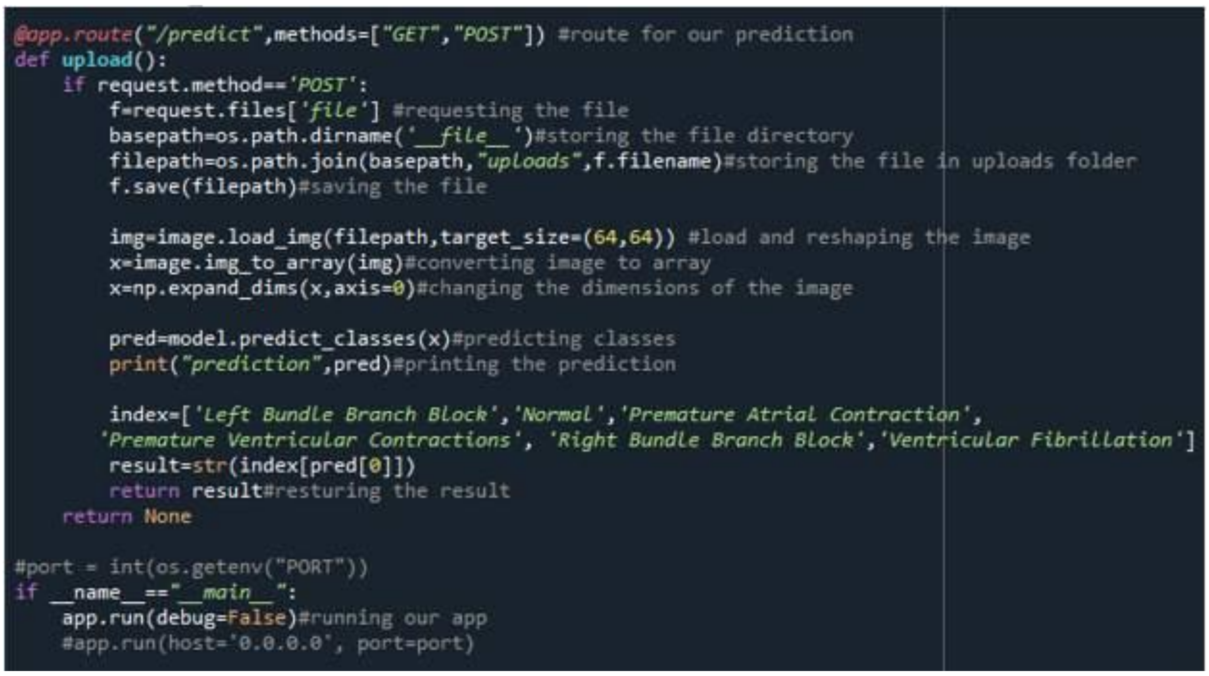
Routing to the HTML Page



```
app=Flask(__name__)#our flask app
model=load_model('ECG.h5')#loading the model

@app.route("/") #default route
def about():
    return render_template("about.html")#rendering html page

@app.route("/about") #default route
def home():
    return render_template("about.html")#rendering html page

@app.route("/info") #default route
def information():
    return render_template("info.html")#rendering html page

@app.route("/upload") #default route
def test():
    return render_template("index6.html")#rendering html page
```

Showcasing prediction on UI

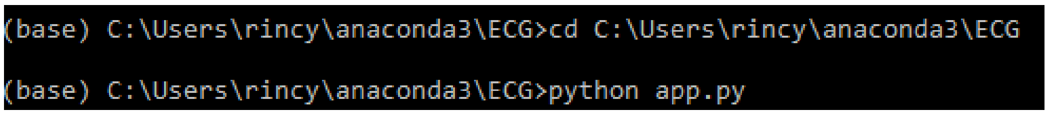
When the image is uploaded, it predicts the category of uploaded the image is either 'Left Bundle Branch Block', 'Normal', 'Premature Atrial Contraction', 'Premature Ventricular Contractions', 'Right Bundle Branch Block', 'Ventricular Fibrillation'. If the image predicts value as 0, then it is displayed as "Left Bundle Branch". Similarly, if the predicted value is 1, it displays "Normal" as output and so on.



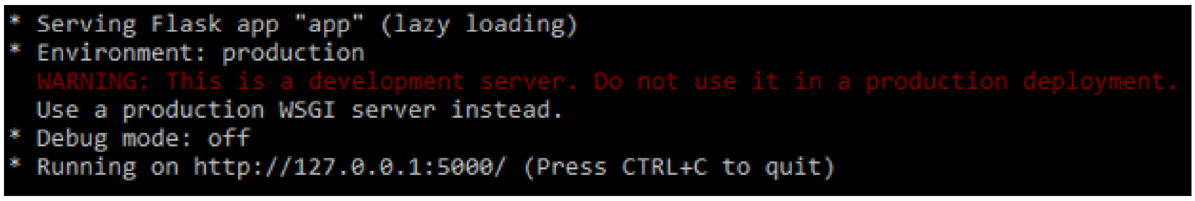
```python
@app.route("/predict",methods=["GET","POST"]) #route for our prediction
def upload():
    if request.method=='POST':
        f=request.files['file'] #requesting the file
        basepath=os.path.dirname('__file__')#storing the file directory
        filepath=os.path.join(basepath,"uploads",f.filename)#storing the file in uploads folder
        f.save(filepath)#saving the file

        img=image.load_img(filepath,target_size=(64,64)) #load and reshaping the image
        x=image.img_to_array(img)#converting image to array
        x=np.expand_dims(x,axis=0)#changing the dimensions of the image

        pred=model.predict_classes(x)#predicting classes
        print("prediction",pred)#printing the prediction

        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#resturing the result
    return None

#port = int(os.getenv("PORT"))
if __name__=="__main__":
    app.run(debug=False)#running our app
    #app.run(host='0.0.0.0', port=port)
```

RUN THE APP

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type "python app.py" command
- Navigate to the localhost where you can view your web page
- Then it will run on localhost:5000
- Navigate to the localhost (http://127.0.0.1:5000/)where you can view your web page.



```
(base) C:\Users\rincy\anaconda3\ECG>cd C:\Users\rincy\anaconda3\ECG

(base) C:\Users\rincy\anaconda3\ECG>python app.py
```



```
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Upload an image and see the predicted result

TRAIN THE MODEL ON IBM

In this milestone, you will learn how to build Deep Learning Model Using the IBM cloud.

REGISTER FOR THE IBM CLOUD

To complete this project you must have an IBM account

IBM Account:

- Please click **here** to register for IBM
- Please click **here** to log in to IBM Account.

TRAIN THE MODEL ON IBM WATSON

To train the model on IBM and integrate it with the flask Application.

IDEATION PHASE

In this milestone you are expected to get started with the Ideation process.

LITERATURE SURVEY ON THE SELECTED PROJECT & INFORMATION GATHERING

In this activity you are expected to gather/collect the relevant information on project use case, refer the existing solutions, technical papers, research publications etc.

PREPARE EMPATHY MAP

In this activity you are expected to prepare the empathy map canvas to capture the user Pains & Gains, Prepare list of problem statements.

IDEATION

In this activity you are expected to list the ideas (at least 4 per each team member) by organizing the brainstorming session and prioritize the top 3 ideas based on the feasibility & importance.

PROJECT DESIGN PHASE - 1

From this milestone you will be starting the project design phase. You are expected to cover the activities given.

PROPOSED SOLUTION

In this activity you are expected to prepare the proposed solution document, which includes the novelty, feasibility of idea, business model, social impact, scalability of solution, etc.

PROBLEM SOLUTION FIT

In this activity you are expected to prepare problem - solution fit document and submit for review.

SOLUTION ARCHITECTURE

In this activity you are expected to prepare solution architecture document and submit for review.

PROJECT DESIGN PHASE - II

From this milestone you will be continue working on the project design phase. You are expected to cover the activities given.

CUSTOMER JOURNEY

Prepare the customer journey maps to understand the user interactions & experiences with the application (entry to exit).

FUNCTIONAL REQUIREMENT

In this activity you are expected to prepare the functional requirement document.

DATE FLOW DIAGRAMS

In this activity you are expected to prepare the data flow diagrams and submit for review.

TECHNOLOGY ARCHITECTURE

In this activity you are expected to draw the technology architecture diagram.

PROJECT DEVELOPMENT PHASE

In this milestone you will start the project development and expected to perform the coding & solutioning, acceptance testing, performance testing based as per the sprint and submit them.

PROJECT DEVELOPMENT – DELIVERY OF SPRINT - 1
PROJECT DEVELOPMENT – DELIVERY OF SPRINT - 2
PROJECT DEVELOPMENT – DELIVERY OF SPRINT - 3
PROJECT DEVELOPMENT – DELIVERY OF SPRINT - 4

RECORD YOUR NOTES/RESEARCH

How do you record research notes?

Taking Notes from Research Reading

1. Know what kind of ideas you need to record. Focus your approach to the topic before you start detailed research. ...
2. Don't write down too much. Your essay must be an expression of your own thinking, not a patchwork of borrowed ideas. ...
3. Label your notes intelligently.

OUTLINE THE STEPS/PLAN FOR YOUR PROJECT

Step 1: Set and prioritize goals. ...
Step 2: Define deliverables. ...
Step 3: Create the project schedule. ...
Step 4: Identify issues and complete a risk assessment. ...

SUMMARIZE WHAT YOU LEARNED

In this study, we proposed a 2-D CNN-based classification model for automatic classification of cardiac arrhythmias using ECG signals. An accurate taxonomy of ECG signals is extremely helpful in the prevention and diagnosis of CVDs. Deep CNN has proven useful in enhancing the accuracy of diagnosis algorithms in the fusion of medicine and modern machine learning technologies. The proposed CNN-based classification algorithm, using 2-D images, can classify eight kinds of arrhythmia, namely, NOR, VFW, PVC, VEB, RBB, LBB, PAB, and APC, and it achieved 97.91% average sensitivity, 99.61% specificity, 99.11% average accuracy, and 98.59% positive predictive value (precision). These results indicate that the prediction and classification of arrhythmia with 2-D ECG representation as spectrograms and the CNN model is a reliable operative technique in the diagnosis of CVDs. The proposed scheme can help experts diagnose CVDs by referring to the automated classification of ECG signals. The present research uses only a single-lead ECG signal. The effect of multiple lead ECG data to further improve experimental cases will be studied in future work.

Add the link to your project here:

https://github.com/IBM-EPBL/IBM-Project-51822-1664431472