

# ECG- Image Based Heartbeat Classification For Arrhythmia Detection Using IBM Watson Studio

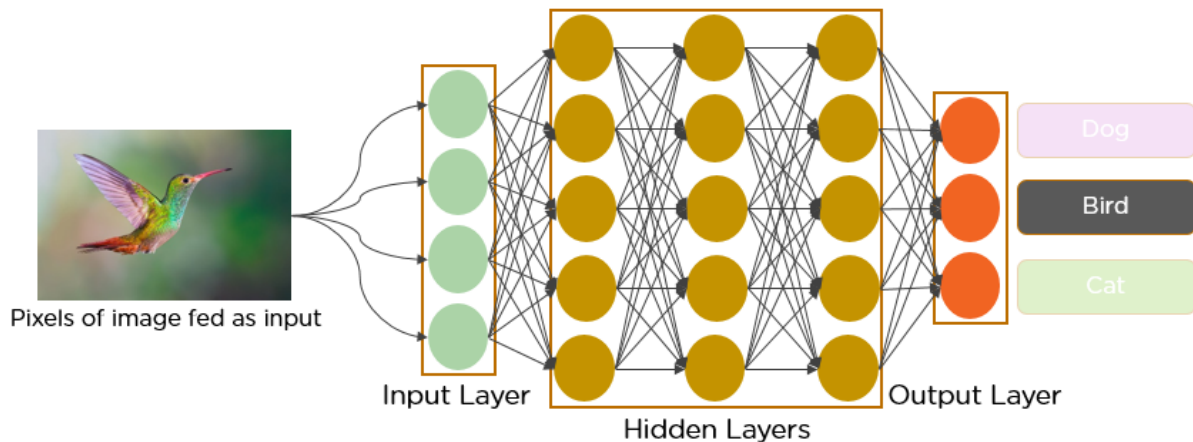
## Introduction:

### 1.1 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.

### 1.2 Purpose:

In the past few decades, Deep Learning has proved to be a compelling tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks.



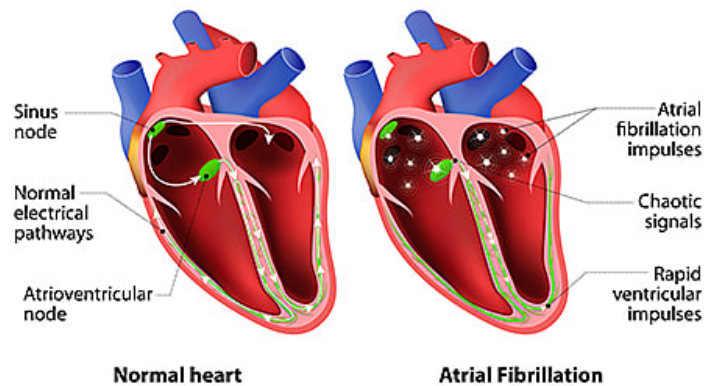
In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

## Literature Survey:

### 2.1 Existing Problem:

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.

## Cardiac arrhythmia

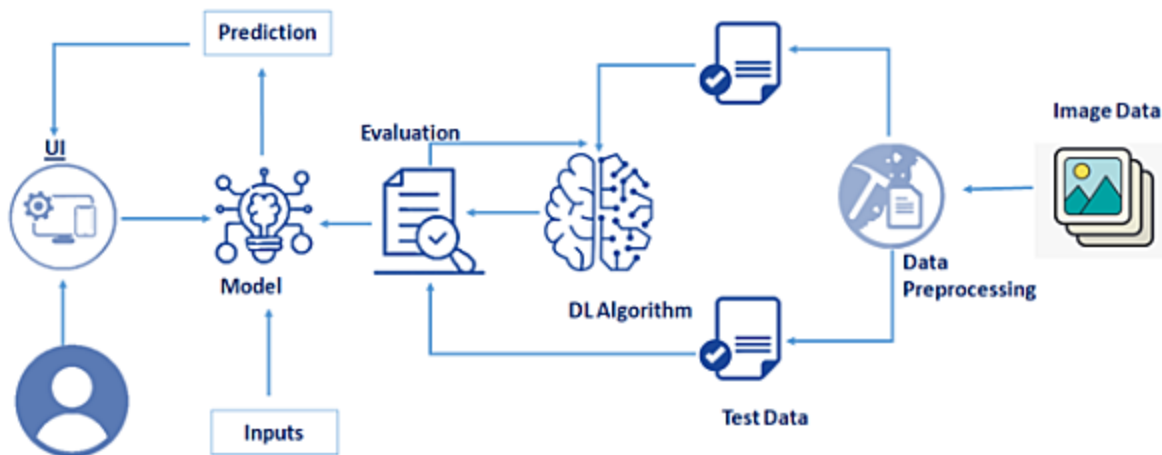


### 2.1 Proposed Solution:

An "ambulatory electrocardiogram" or an ECG) about the size of a postcard or digital camera that the patient will be using for 1 to 2 days, or up to 2 weeks. The test measures the movement of electrical signals or waves through the heart. These signals tell the heart to contract (squeeze) and pump blood. The patient will have electrodes taped to your skin. It's painless, although some people have mild skin irritation from the tape used to attach the electrodes to the chest. They can do everything but shower or bathe while wearing the electrodes. After the test period, patient will go back to see your doctor. They will be downloading the information.

## Theoretical Experience:

### 3.1 Block Diagram:



We will prepare the project by following the below steps:

- We will be working with Sequential type of modeling
- We will be working with Keras capabilities
- We will be working with image processing techniques
- We will build a web application using the Flask framework.
- Afterwards we will be training our dataset in the IBM cloud and building another model from IBM and we will also test it.

### 3.2 HARDWARE & SOFTWARE Designing:

#### Hardware Components used:

Since we are using the IBM cloud as a platform to execute this project we don't need any hardware components other than our system.

#### Software Components Used:

We will be using Anaconda Navigator which is installed in our system and Watson studio from the IBM cloud to complete the project.

##### 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

### **WATSON STUDIO:**

Watson Studio is one of the core services in Cloud Pak for Data as a Service.

Watson Studio provides you with the environment and tools to solve your business problems by collaboratively working with data. You can choose the tools you need to analyze and visualize data, to cleanse and shape data, or to build machine learning models.

This illustration shows how the architecture of Watson Studio is centered around the project. A project is a workspace where you organize your resources and work with data.

Watson Studio projects fully integrate with the catalogs and deployment spaces:

- Deployment spaces are provided by the Watson Machine Learning service
- You can easily move assets between projects and deployment spaces.

### **Experimental Investigations:**

In this project, we have deployed our training model using CNN on IBM Watson studio and in our local machine. We are deploying 4 types of CNN layers in a sequential manner , starting from :

1. **Convolutional layer 2D:**A 2-D convolutional layer applies sliding convolutional filters to 2-D input. The layer convolves the input by moving the filters along the input vertically and horizontally and computing the dot product of the weights and the input, and then

adding a bias term.

2. **Pooling Layer** :Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network. The pooling layer summarises the features present in a region of the feature map generated by a convolution layer.
3. **Fully-Connected layer** :After extracting features from multiple convolution layers and pooling layers, the fully-connected layer is used to expand the connection of all features. Finally, the SoftMax layer makes a logistic regression classification. Fully-connected layer transfers the weighted sum of the output of the previous layer to the activation function.
4. **Dropout Layer** :There is usually a dropout layer before the fully-connected layer. The dropout layer will temporarily disconnect some neurons from the network according to the certain probability during the training of the convolution neural network, which reduces the joint adaptability between neuron nodes, reduces overfitting, and enhances the generalization ability of the network.

## Flow Chart & Results with Screenshots:

### 5.1 Flow Chart & Results by training model in local machine:

#### a. Dataset Collection:

The dataset contains six classes:

1. Left Bundle Branch Block
2. Normal
3. Premature Atrial Contraction
4. Premature Ventricular Contractions
5. Right Bundle Branch Block
6. Ventricular Fibrillation

## b. Image Preprocessing:

Image Pre-processing includes the following main tasks

- **Import ImageDataGenerator 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.

```
In [5]: 1 from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

- **Configure ImageDataGenerator 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.

```
In [6]: 1 train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)
        2 test_datagen = ImageDataGenerator(rescale = 1./255)
```

- **Applying ImageDataGenerator functionality to the trainset and test set:**

We will apply ImageDataGenerator functionality to Trainset and Testset 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}

```
In [7]: 1 x_train = train_datagen.flow_from_directory("/content/data/train",target_size = (64,64),batch_size = 32,\
2                                               class_mode = "categorical")
3 x_test = test_datagen.flow_from_directory("/content/data/test",target_size = (64,64),batch_size = 32,\
4                                           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.

### c. Model Building

We are ready with the augmented and pre-processed image data,we will begin our build our model by following the below steps:

- **Import the model building Libraries:**

```
In [4]: 1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Dense
3 from tensorflow.keras.layers import Convolution2D
4 from tensorflow.keras.layers import MaxPooling2D
5 from tensorflow.keras.layers import Flatten
```

- **Initializing 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 downsample the input.

The flatten layer flattens the input.



```

In [9]: 1 #MODEL BUILDING

In [10]: 1 model = Sequential()

In [11]: 1 model.add(Convolution2D(32,(3,3),input_shape = (64,64,3),activation = "relu"))

In [12]: 1 model.add(MaxPooling2D(pool_size = (2,2)))

In [13]: 1 model.add(Convolution2D(32,(3,3),activation='relu'))

In [14]: 1 model.add(MaxPooling2D(pool_size=(2,2)))

In [15]: 1 model.add(Flatten()) # ANN Input...

```

### ● Adding Hidden Layers:

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

```

In [16]: 1 #Adding Dense Layers

In [17]: 1 model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

In [18]: 1 model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

In [19]: 1 model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

In [20]: 1 model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

In [21]: 1 model.add(Dense(units = 128,kernel_initializer = "random_uniform",activation = "relu"))

```

### ● Adding Output Layer:

```

In [22]: 1 model.add(Dense(units = 6,kernel_initializer = "random_uniform",activation = "softmax"))

```

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.

```
In [23]: 1 model.summary()
```

```
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 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 128)	802944
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 128)	16512
dense_4 (Dense)	(None, 128)	16512
dense_5 (Dense)	(None, 6)	774
Total params: 879,910		
Trainable params: 879,910		
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.

```
In [24]: 1 model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

- **Training the model:**

We will train our model with our image dataset. `fit_generator` functions used to train a deep learning neural network.

```
In [25]: 1 model.fit_generator(generator=x_train,steps_per_epoch = len(x_train), epochs=9, validation_data=x_test,\n2 validation_steps = len(x_test))\n\nEpoch 1/9\n480/480 [=====] - 99s 203ms/step - loss: 1.4415 - accuracy: 0.4788 - val_loss: 1.6093 - val_accuac\ny: 0.3193\nEpoch 2/9\n480/480 [=====] - 96s 201ms/step - loss: 0.9465 - accuracy: 0.6495 - val_loss: 1.3444 - val_accuac\ny: 0.5121\nEpoch 3/9\n480/480 [=====] - 97s 201ms/step - loss: 0.5540 - accuracy: 0.8018 - val_loss: 0.7785 - val_accuac\ny: 0.7698\nEpoch 4/9\n480/480 [=====] - 99s 205ms/step - loss: 0.2770 - accuracy: 0.9069 - val_loss: 0.6690 - val_accuac\ny: 0.8296\nEpoch 5/9\n480/480 [=====] - 97s 201ms/step - loss: 0.2037 - accuracy: 0.9388 - val_loss: 0.6057 - val_accuac\ny: 0.8416\nEpoch 6/9\n91/480 [==>.....] - ETA: 1:09 - loss: 0.1595 - accuracy: 0.9499
```

- **Saving 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.

```
In [26]: 1 #Saving Model.\n2 model.save('ECG.h5')
```

- **Testing the model:**

Load necessary libraries and load the saved model using `load_model`

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.

```

In [26]: 1 #Saving Model.
          2 model.save('ECG.h5')

In [28]: 1 from tensorflow.keras.models import load_model
          2 from tensorflow.keras.preprocessing import image

In [29]: 1 model=load_model('ECG.h5')

In [30]: 1 img=image.load_img("/content/Unknown_image.png",target_size=(64,64))

In [31]: 1 x=image.img_to_array(img)

In [32]: 1 import numpy as np

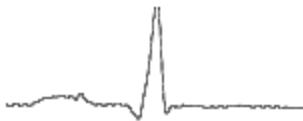
In [33]: 1 x=np.expand_dims(x,axis=0)

In [34]: 1 pred = model.predict(x)
          2 y_pred=np.argmax(pred)
          3 y_pred
Out[34]: 1

In [35]: 1 index=['left Bundle Branch block',
          2         'Normal',
          3         'Premature Atrial Contraction',
          4         'Premature Ventricular Contraction',
          5         'Right Bundle Branch Block',
          6         'Ventricular Fibrillation']
          7 result = str(index[y_pred])
          8 result
Out[35]: 'Normal'

```

The unknown image uploaded is:



Here the output for the uploaded result is normal.

#### d. Application Building:

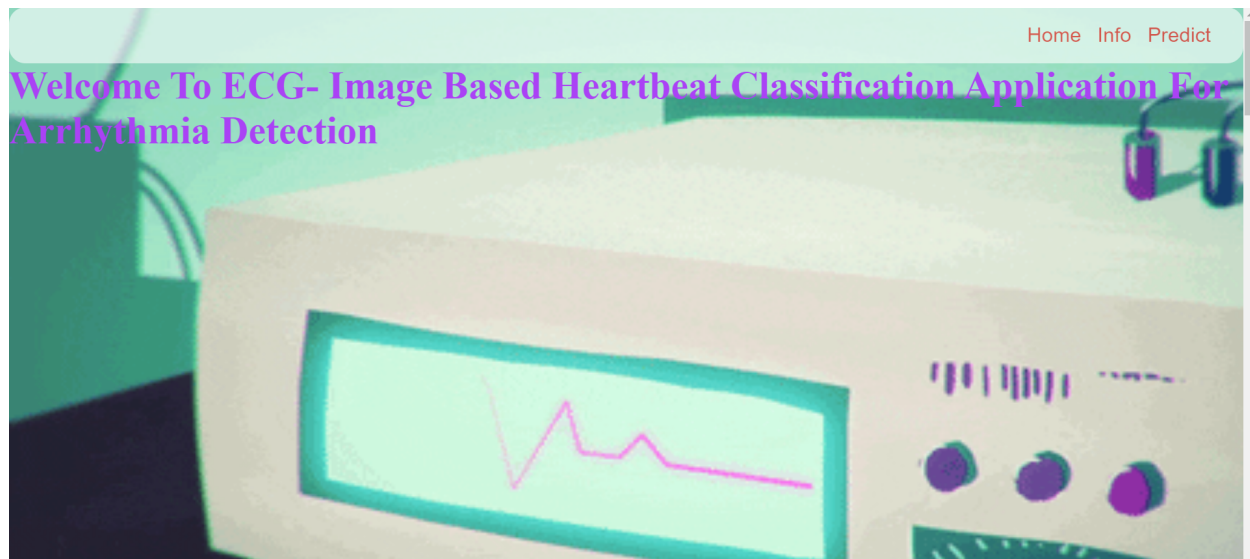
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has uploaded an image. The uploaded image is given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- **Building HTML Pages:**

- We use HTML to create the front end part of the web page.
- Here, we created 4 html pages- home.html, predict\_base.html, predict.html, information.html.

- home.html displays the home page.

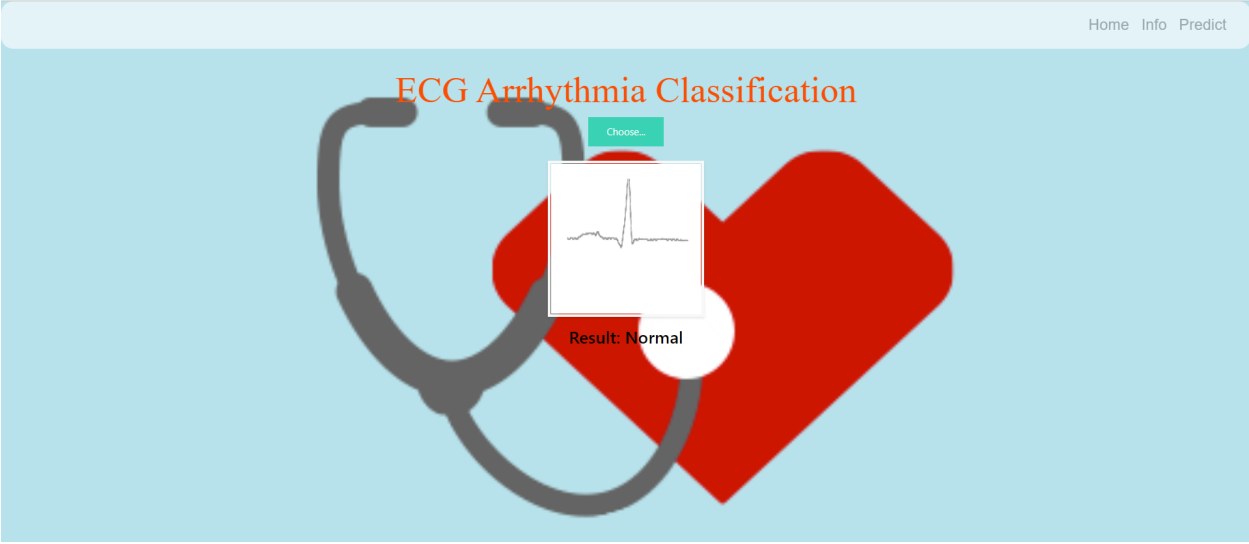
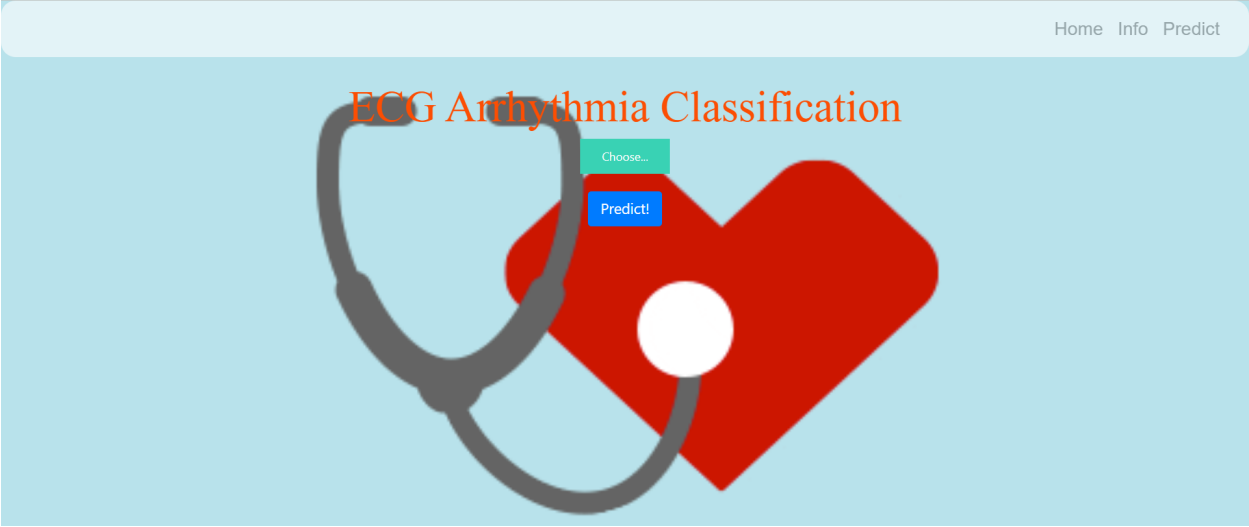


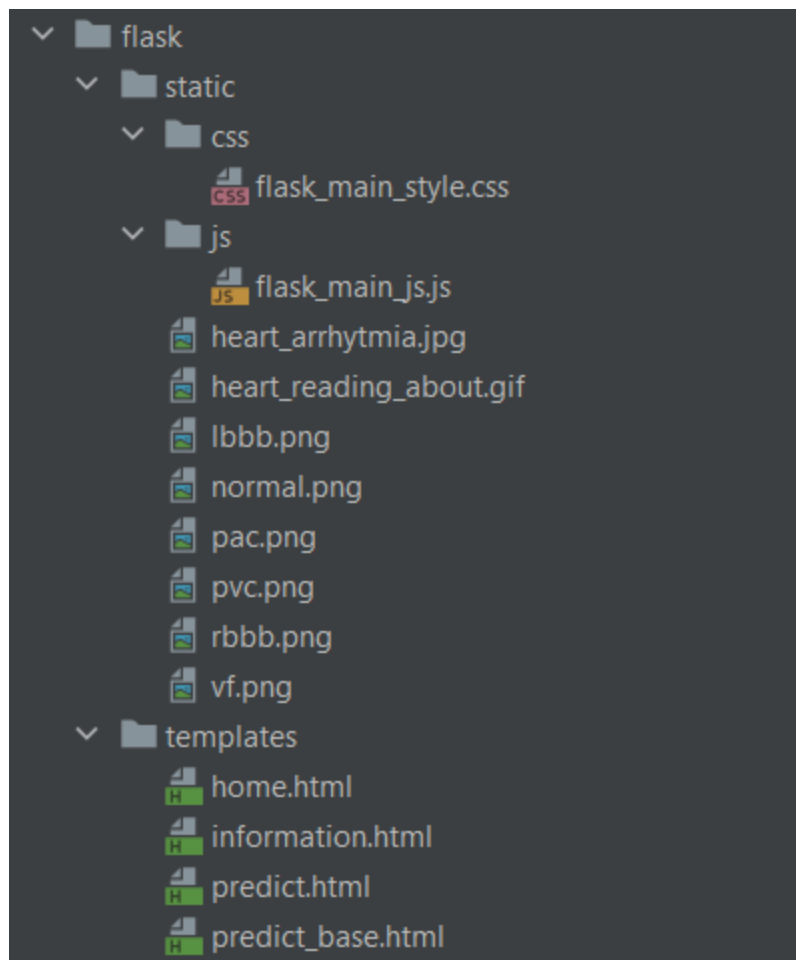
- information.html displays all important details to be known about ECG.

### [ECG- Image Based Heartbeat Classification Information Guide](#)



- predict-base.html and predict.html accept input from the user and predicts the values.





- **Building server-side script:**

We will build the flask file 'app.py' which is a web framework written in python for server-side scripting.

- 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.

```
1 import os
2 import numpy as np #used for numerical analysis
3 from flask import Flask, request, render_template
4 # Flask-It is our framework which we are going to use to run/serve our application.
5 #request-for accessing file which was uploaded by the user on our application.
6 #render_template- used for rendering the html pages
7 from tensorflow.keras.models import load_model#to load our trained model
8 from tensorflow.keras.preprocessing import image
9
10 app=Flask(__name__)#our flask app
11 model=load_model('ECG.h5')#loading the model
12
13 @app.route("/") #default route
14 def about():
15     return render_template("home.html")#rendering html page
16
17 @app.route("/about") #default route
18 def home():
19     return render_template("home.html")#rendering html page
20
21 @app.route("/info") #default route
22 def information():
23     return render_template("information.html")#rendering html page
24
25 @app.route("/upload") #default route
26 def test():
27     return render_template("predict.html")#rendering html page
```



```
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(x) #predicting classes
        y_pred = np.argmax(pred)
        print("prediction",y_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[y_pred])

        return result #returning 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=8000)
```

## ● Running The App:

```
C:\Users\M Sheshikiran Reddy\VIT\20BAI1061\CNN_PROJECT_SMARTINTERNZ\flask>python app_flask.py
```

```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
* Serving Flask app 'app_flask' (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)
```



Navigate to the localhost (<http://127.0.0.1:5000/>) where you can view your web page.

## 5.2 Flow Chart & Results by training model in IBM WATSON STUDIO:


### a. Creating IBM cloud account:

We have to create an IBM Cloud Account and should log in.

### b. Creating Watson Studio Service & Machine Learning Service:


 Machine Learning-7n	Default	Dallas	Machine Learning	✔ Active
 Watson Studio-yy	Default	Dallas	Watson Studio	✔ Active


### c. Create a Project & Deployment space in the watson studio:


 IBM Watson Studio

Search in your workspaces

Buy



Sheshi Kiran Reddy Mandla... 

 SM

## New project

#### Define details

Name

Description

Choose project options

☐ Restrict who can be a collaborator ⓘ

Project includes integration with Cloud Object Storage for storing project assets.

#### Storage

Cancel>Create

## CNN\_ECG\_IBM\_SHESHI

Overview

Assets

Deployments

Jobs

Manage

### Deployments

All

1

✓ Deployed

0

✗ Failed

[View deployments](#)

### Job runs

0

🔄 Active

0

✗ Failed last 24 hours

[View jobs](#)

### Space activity

#### ✓ Online deployment ready

The online deployment **CNN\_SHESHI\_DEPLOYMENT** in space **CNN\_ECG\_IBM\_SHESHI** is ready to accept requests

Yesterday at 06:03 PM

#### 👤 Online deployment created

You created online deployment "CNN\_SHESHI\_DEPLOYMENT" in space **CNN\_ECG\_IBM\_SHESHI**. You must wait for the deployment to enter ready state before submitting requests.

Yesterday at 06:03 PM

## Welcome, Sheshi kiran reddy!

### Take a tutorial

Step through implementing a Data fabric use case in a sample project.

[Learn by example](#)

### Work with data

Create a project for your team to prepare data, find insights, or build models.

[Create a project](#)

### Extend your capabilities

Add tools, databases, or other features by creating services instances.

[Create a service](#)

### Quick start



Build dashboards  
with IBM Cognos Dashboard Embedded



Create data pipelines  
with DataStage



Build customer profiles  
with IBM Match 360 with Watson



Catalog and govern data  
with Watson Knowledge Catalog

## Overview

### Projects

CNN\_ECG\_SHESHI

Yesterday at 02:08 PM

### New in gallery

### Notifications

#### ✓ Online deployment ready

The online deployment **CNN\_SHESHI\_DEPLOYMENT** in space

Yesterday at 06:03 PM

### Deployments ⓘ

CNN\_ECG\_IBM\_SHESHI

Yesterday at 06:02 PM

3 assets

All assets

Asset types

- > Data 2
- </> Source Code 1
  - Notebook 1

Data

Name	Last modified
Unknown_image.png PNG	14 hours ago Sheshi Kiran Reddy Mandla (You)
data.zip application/x-zip-compressed	15 hours ago Sheshi Kiran Reddy Mandla (You)

Find assets

Add asset

New

3 assets

All assets

Asset types

- > Data 2
- </> Source Code 1
- Notebook 1

Notebook

Name	Language	Last modified
CNN_ECG_SHESHI Notebook	Python 3.9	11 hours ago Sheshi Kiran Reddy Mandla (You)

e. Apply CNN algorithm and save the model and deploy it using API key generated:

```
In [77]: model.save('ECG_IBM.h5')
```

```
In [109]: !tar -zcvf ECG-arrhythmia-classification-model_new.tgz ECG_IBM.h5
          ECG_IBM.h5
```

```
In [110]: ls -l
          data/
          ECG-arrhythmia-classification-model_new.tgz
          ECG-classification.tgz
          ECG_IBM.h5
```

```
In [112]: from ibm_watson_machine_learning import APIClient
wml_credentials={
    "url":"https://us-south.ml.cloud.ibm.com",
    "apikey":"EfnNlIAqu-QXB0QsQqQlnwkes_B9ssggk8ipjZQH67"
}
client=APIClient(wml_credentials)

In [121]: client.spaces.list()

Note: 'limit' is not provided. Only first 50 records will be displayed if the number of records exceed 50
-----
ID                                NAME                                CREATED
fc75767f-659b-4dc3-a238-cbb53d201cf2  CNN_ECG_IBM_SHESHI  2022-07-05T10:52:44.075Z
-----

In [122]: space_uid="fc75767f-659b-4dc3-a238-cbb53d201cf2"

In [123]: client.set.default_space(space_uid)
Out[123]: 'SUCCESS'

In [124]: client.set.default_space(space_uid)
Out[124]: 'SUCCESS'

In [126]: software_spec_uid = client.software_specifications.get_uid_by_name("tensorflow_rt2.1-py3.9")
software_spec_uid
Out[126]: 'acd9c798-6974-5d2f-a657-ce06e986df4d'

In [143]: model_details = client.repository.store_model(model='ECG-arrhythmia-classification-model_new.tgz',meta_props={
    client.repository.ModelMetaNames.NAME:"CNN_SHESHI",
    client.repository.ModelMetaNames.TYPE:"tensorflow_2.7",
    client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_spec_uid})
model_id=client.repository.get_model_uid(model_details)

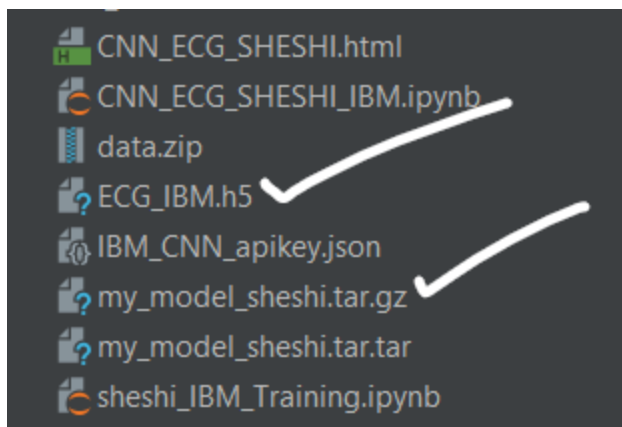
This method is deprecated, please use get_model_id()

In [144]: model_id
Out[144]: '70742fe8-7ac8-4855-994c-b572931ef787'

In [147]: client.repository.download(model_id,'my_model.tar1.gz')
Successfully saved model content to file: 'my_model.tar1.gz'
Out[147]: '/home/wsuser/work/my_model.tar1.gz'
```

**f. For downloading the model we have to run the last part of the above code in the local jupyter notebook:**

```
In [ ]: 1 client.repository.download('70742fe8-7ac8-4855-994c-b572931ef787','my_model_sheshi.tar.gz')
```



g. Now we will extract the .h5 model file and will do the app deployment using flask as done in the previous training:

```
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 app
# request-for accessing file which was uploaded by the user on our applicat
# 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

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

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

Hence we trained the model using IBM Watson.

## Advantages & Disadvantages:

### 6.1 Advantages:

- The proposed model predicts Arrhythmia in images with a high accuracy rate of nearly 96%
- The early detection of Arrhythmia gives better understanding of disease causes, initiates therapeutic interventions and enables developing appropriate treatments.

### 6.2 Disadvantages:

- Not useful for identifying the different stages of Arrhythmia disease.
- Not useful in monitoring motor symptoms

### Applications :

- It is useful for identifying the arrhythmia disease at an early stage.
- It is useful in detecting cardiovascular disorders

### Conclusion:

- Cardiovascular disease is a major health problem in today's world. The early diagnosis of cardiac arrhythmia highly relies on the ECG.
- Unfortunately, the expert level of medical resources is rare, visually identify the ECG signal is challenging and time-consuming.
- The advantages of the proposed CNN network have been put to evidence.
- It is endowed with an ability to effectively process the non-filtered dataset with its potential anti-noise features. Besides that, ten-fold cross-validation is implemented in this work to further demonstrate the robustness of the network.

### Future Scope:

For future work, it would be interesting to explore the use of optimization techniques to find a feasible design and solution. The limitation of our study is that we have yet to apply any optimization techniques to optimize the model parameters and we believe that with the implementation of the optimization, it will be able to further elevate the performance of the proposed solution to the next level.

### References:

- <https://github.com/Anshuman151/ECG-Image-Based-Heartbeat-Classification-for-Arrhythmia-Detection-Using-IBM-Watson-Studio/blob/main/README.md>
- <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>
- <https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.convolution2dlayer.html;jsessionid=0a7e3bc26fabda07a5032030294b>

**THE END**

Name: Mandla Sheshi Kiran Reddy

Email-id: sheshikiranmandla2589@gmail.com