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

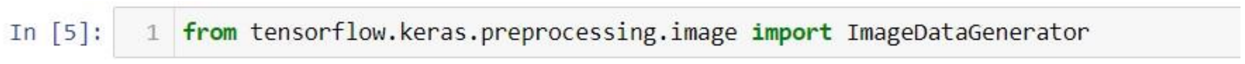
b. Image Preprocessing:

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]: from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

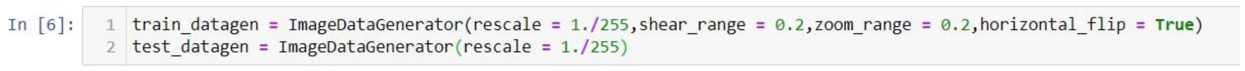
- **Configure ImageDataGenerator Class:**

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.

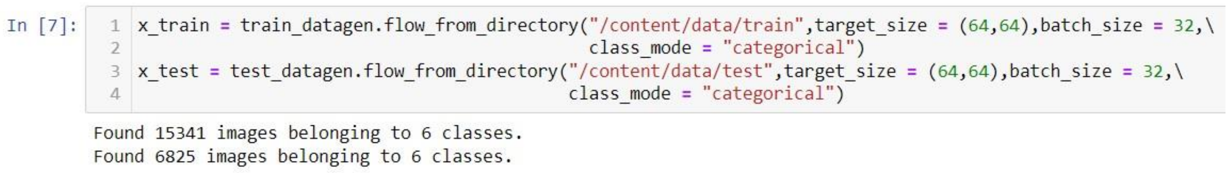
An instance of the ImageDataGenerator class can be constructed for train and test.



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



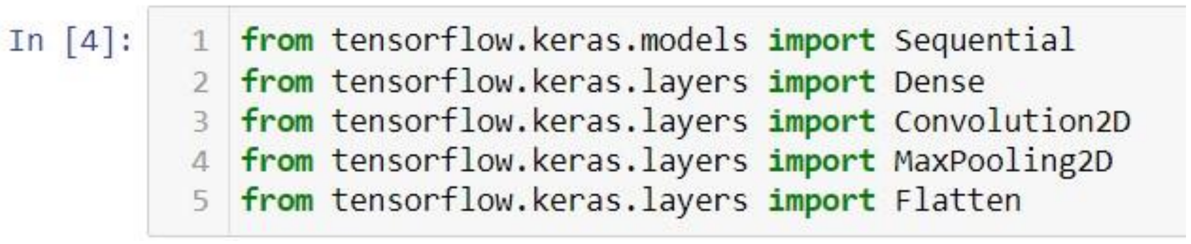
```
In [7]:
x_train = train_datagen.flow_from_directory("/content/data/train",target_size = (64,64),batch_size = 32,\
                                            class_mode = "categorical")
x_test = test_datagen.flow_from_directory("/content/data/test",target_size = (64,64),batch_size = 32,\
       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

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]:
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:**

Keras has 2 ways to define a neural network:

- Sequential
- Function API

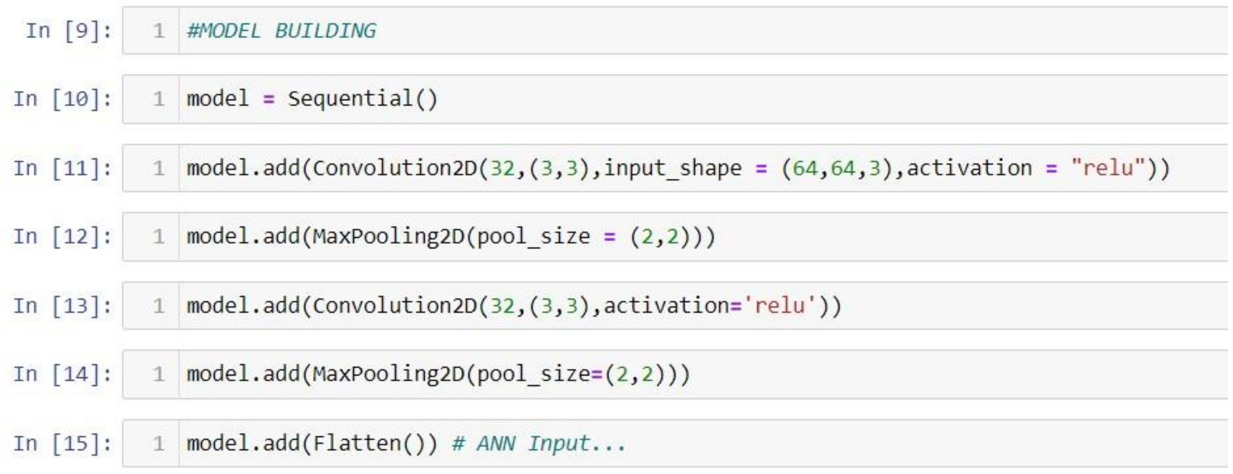
The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add () method.
Now, will initialize our model.

- **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.

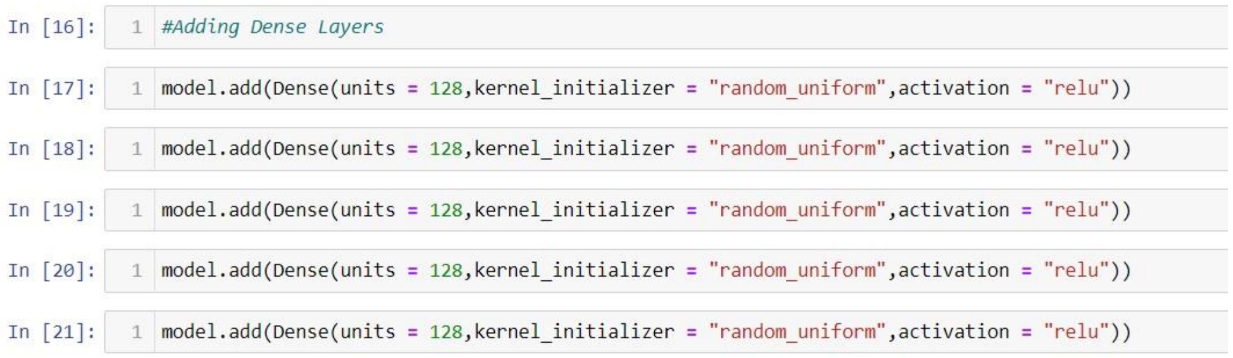
The flatten layer flattens the input.



```
In [9]:  #MODEL BUILDING

In [10]: model = Sequential()

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

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

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

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

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

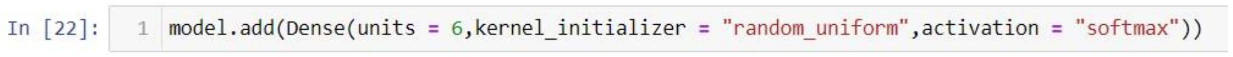
- **Adding Hidden Layers:**

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



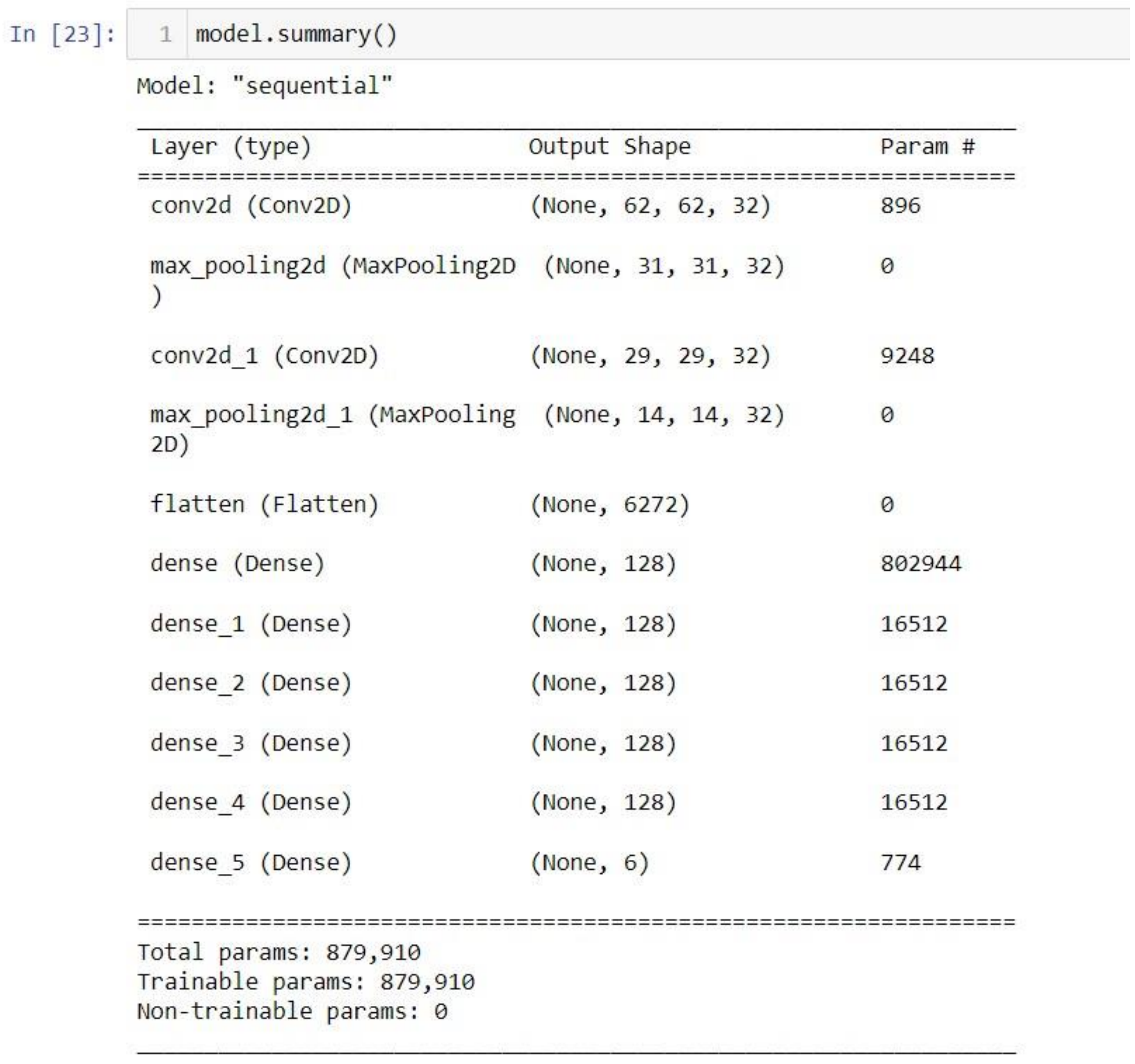
```
In [16]: #Adding Dense Layers

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

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

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

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

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

- **Adding Output Layer:**



```
In [22]: 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]: model.summary()
```

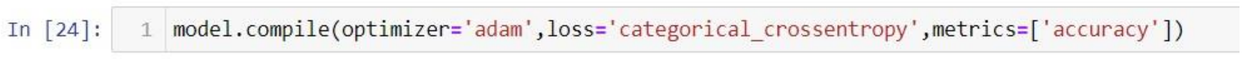
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 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

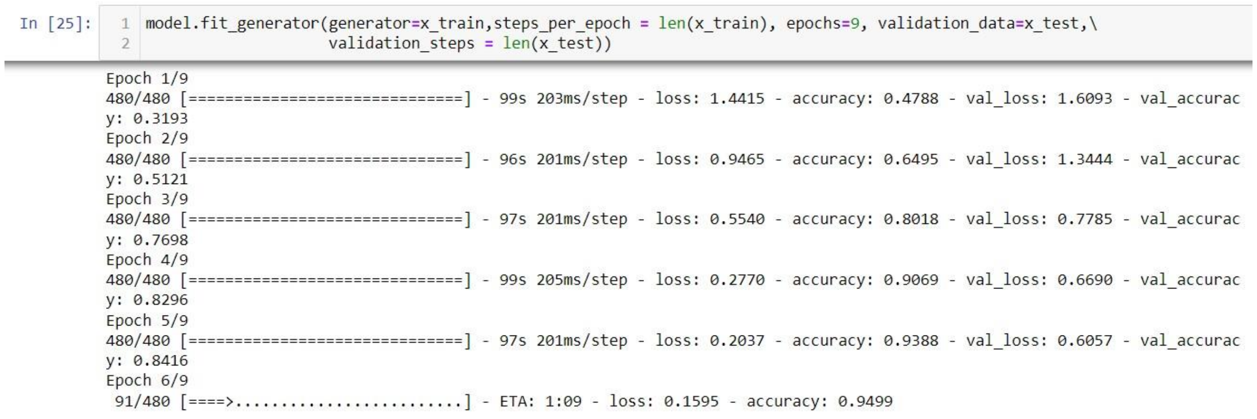
■ Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in the training process.



```
In [24]: 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.



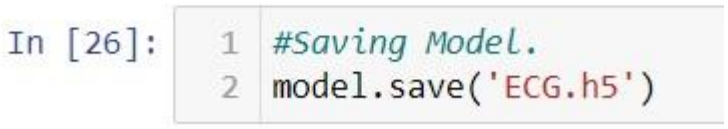
```
In [25]:
model.fit_generator(generator=x_train,steps_per_epoch = len(x_train), epochs=9, validation_data=x_test,\
                    validation_steps = len(x_test))
```

```
Epoch 1/9
480/480 [==============================] - 99s 203ms/step - loss: 1.4415 - accuracy: 0.4788 - val_loss: 1.6093 - val_accurac
y: 0.3193
Epoch 2/9
480/480 [==============================] - 96s 201ms/step - loss: 0.9465 - accuracy: 0.6495 - val_loss: 1.3444 - val_accurac
y: 0.5121
Epoch 3/9
480/480 [==============================] - 97s 201ms/step - loss: 0.5540 - accuracy: 0.8018 - val_loss: 0.7785 - val_accurac
y: 0.7698
Epoch 4/9
480/480 [==============================] - 99s 205ms/step - loss: 0.2770 - accuracy: 0.9069 - val_loss: 0.6690 - val_accurac
y: 0.8296
Epoch 5/9
480/480 [==============================] - 97s 201ms/step - loss: 0.2037 - accuracy: 0.9388 - val_loss: 0.6057 - val_accurac
y: 0.8416
Epoch 6/9
 91/480 [====>.........................] - 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.



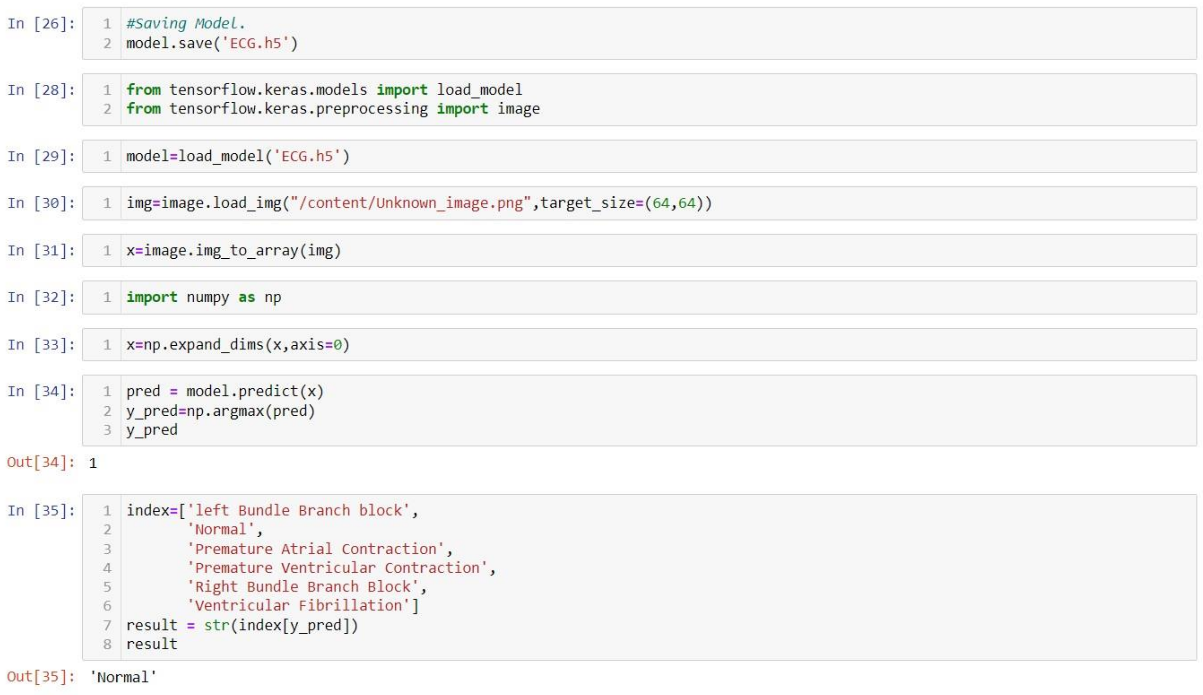
```
In [26]:
#Saving Model.
model.save('ECG.h5')
```

- **Testing the model:**

Load necessary libraries and load the saved model using load_model

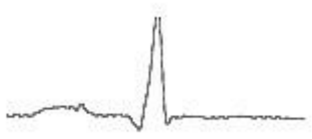
Taking an image as input and checking the results

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]: #Saving Model.
         model.save('ECG.h5')

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

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

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

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

In [32]: import numpy as np

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

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

Out[34]: 1

In [35]: index=['left Bundle Branch block',
                'Normal',
                'Premature Atrial Contraction',
                'Premature Ventricular Contraction',
                'Right Bundle Branch Block',
                'Ventricular Fibrillation']
         result = str(index[y_pred])
         result

Out[35]: 'Normal'
```

The unknown image uploaded is:



Here the output for the uploaded result is normal.