**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.

To know more about the data generator class  click on this [**link**](https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/)

### Import The 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.

Let us import the ImageDataGenerator class from Keras

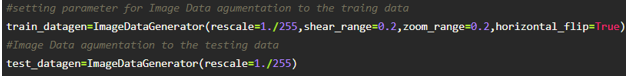
C:\Users\ELCOT\AppData\Local\Packages\Microsoft.Office.Desktop_8wekyb3d8bbwe\AC\INetCache\Content.MSO\A525E6E2.tmp

**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.

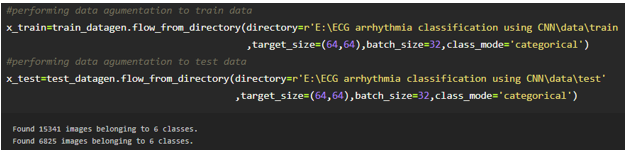
An instance of the ImageDataGenerator class can be constructed for train and test.



**Apply ImageDataGenerator 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}



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 Building

### 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

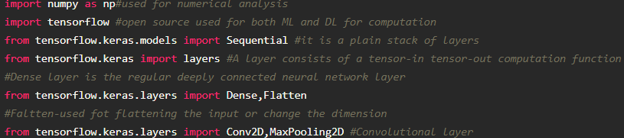
### Training and testing the model

### Saving the model

### To know more about model building please [click here](https://keras.io/about/)

### 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.



**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.

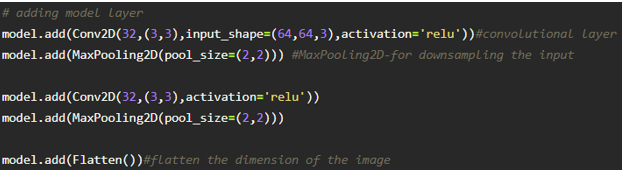
### Adding CNN Layers

For information regarding CNN Layers refer to the [**link**](link:%20https://victorzhou.com/blog/intro-to-cnns-part-1/)

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.



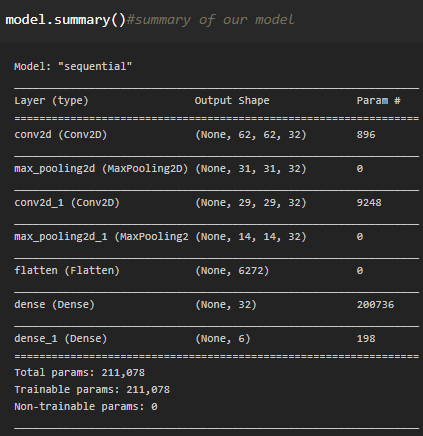
### Adding Dense Layers

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

C:\Users\ELCOT\AppData\Local\Packages\Microsoft.Office.Desktop_8wekyb3d8bbwe\AC\INetCache\Content.MSO\213DDA98.tmp

 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.



**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

C:\Users\ELCOT\AppData\Local\Packages\Microsoft.Office.Desktop_8wekyb3d8bbwe\AC\INetCache\Content.MSO\14BAC564.tmp

**Train The Model**

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

C:\Users\ELCOT\AppData\Local\Packages\Microsoft.Office.Desktop_8wekyb3d8bbwe\AC\INetCache\Content.MSO\F835A512.tmp

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 

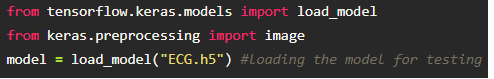
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

C:\Users\ELCOT\AppData\Local\Packages\Microsoft.Office.Desktop_8wekyb3d8bbwe\AC\INetCache\Content.MSO\DCC5D6F0.tmp

### Test The Model

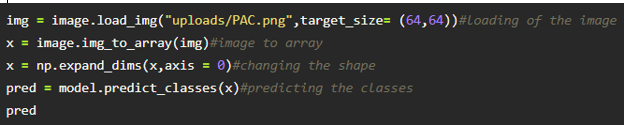
Open the other jupyter file and write the code mentioned below

Load necessary libraries, Load the saved model using load\_model

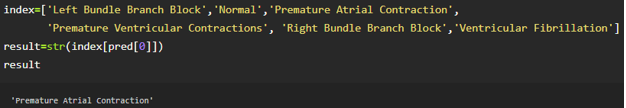


Taking an image as input and checking the results 

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



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



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.

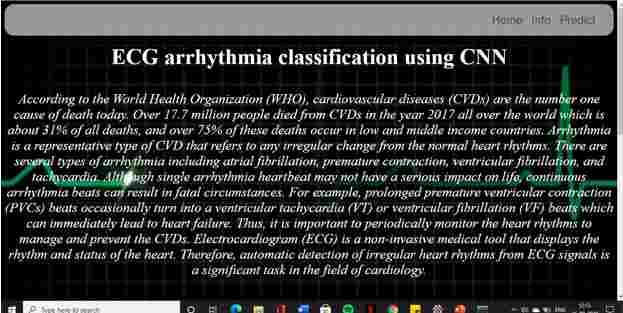
This section has the following tasks

* Building HTML Pages
* Building server-side script

**Create HTML Files**

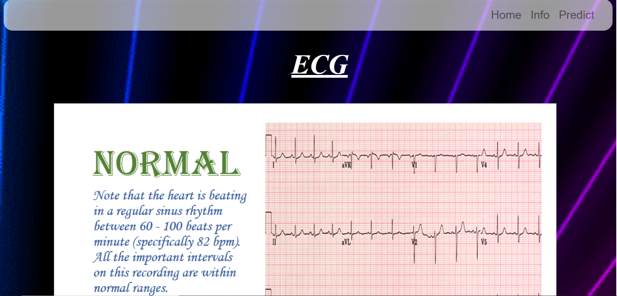
* We use HTML to create the front end part of the web page.
* Here, we created 4 html pages- about.html, base.html, index6.html, info.html.
* about.html displays the home page.
* Info.html displays all important details to be known about ECG.
* base.html and index6.html accept input from the user and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

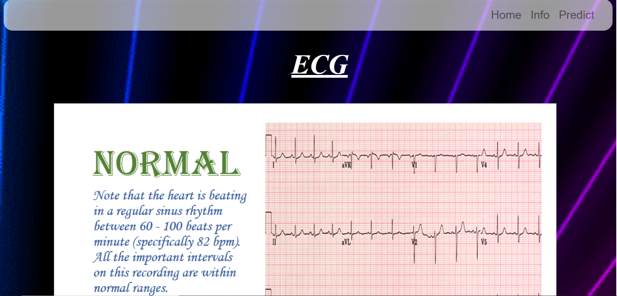
Your Home page looks like



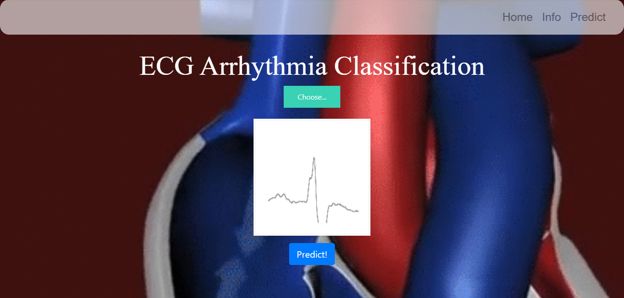
When the “Info” button is clicked, localhost redirects to “info.html”

Your info.html Page looks like:





Your prediction page will look like :

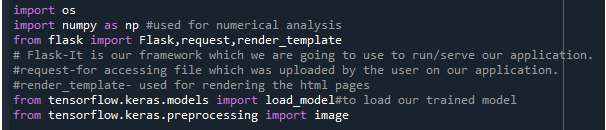


Upload the image and click on the Predict button to view the result on the “base.html” page on the localhost.

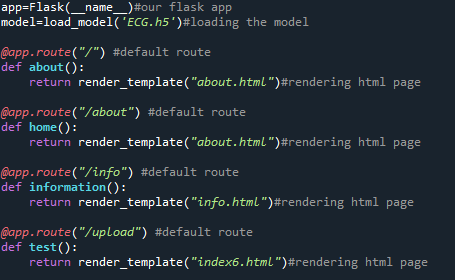
**Build Python Code**

* 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**

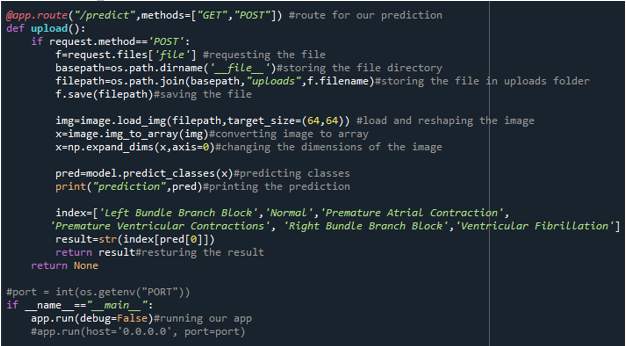


**Routing to the HTML Page**



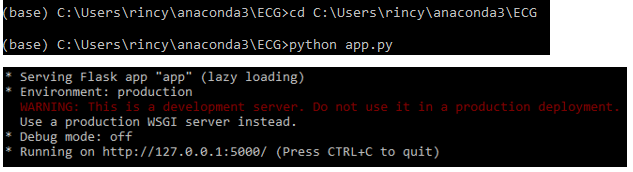
Showcasing prediction on UI

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.



### 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.



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 IBM Cloud**

**To complete this project you must have an IBM account**

**IBM Account:**

* Please click [**here**](https://cloud.ibm.com/registration)to register for IBM
* Please click [**here**](https://cloud.ibm.com/login)to log in to IBM Account.

**Watch the below video to register and login into your IBM account**

Train The Model On IBM Watson

**Please watch the below video to train the model on IBM  and integrate it with the flask Application.**