GLAUCOMA DETECTION USING PYTHON KERAS

The python programming language is becoming a very popular programming language which now supports all form of projects ranging from webs to applications of all sort.

For this research we trying to design a system that can easily detects if a fundus image has GLAUCOMA or not.

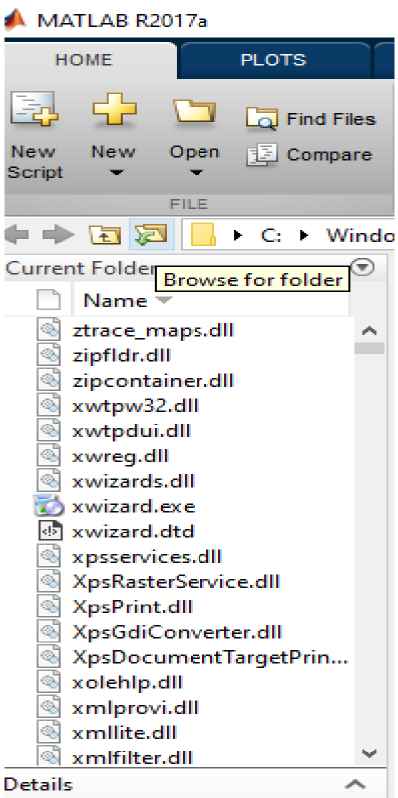
The feature extraction has been done on a matlab application using the GLCM texture analysis technique. Which helps gets certain features such as Contrasts, Correlation, Energy, Homogeneity and Entropy.

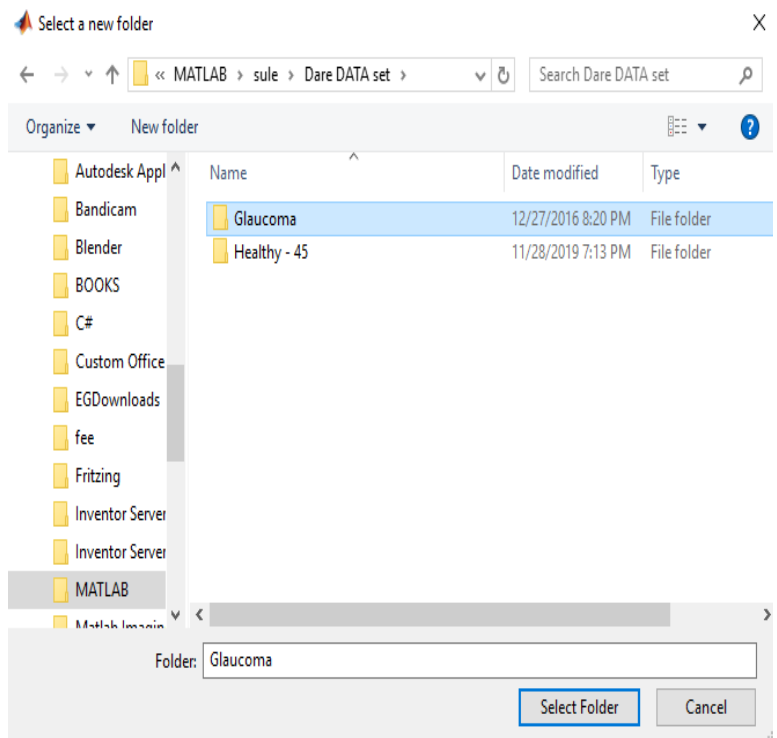
For this project a GUI application was design on the matlab IDE using the *Guide ToolBox.*

Step 1:

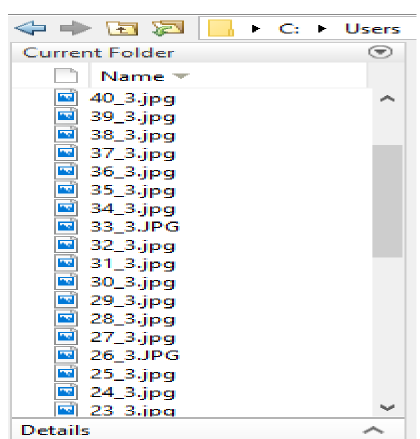
- Fundus image are gotten from a publicly available online fundus database.
- The image were then split into two folders which are GLAUCOMA and HEALTHLY image.

- The direction was then added through the matlab directory before any code can be run on it.



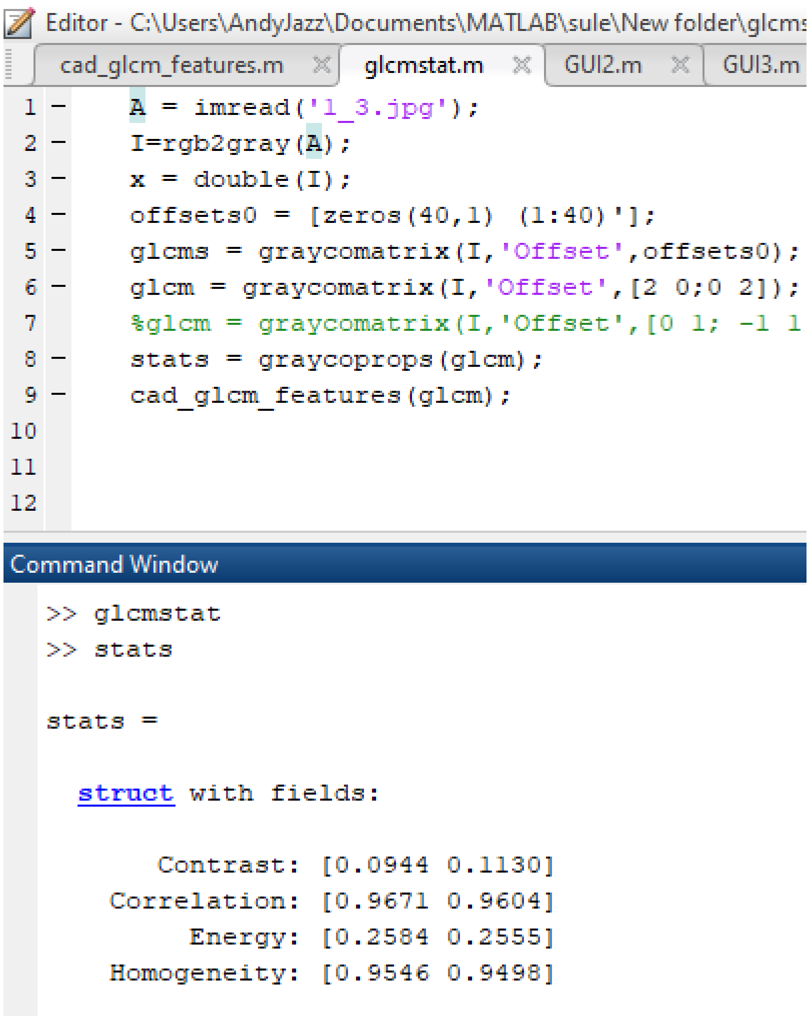


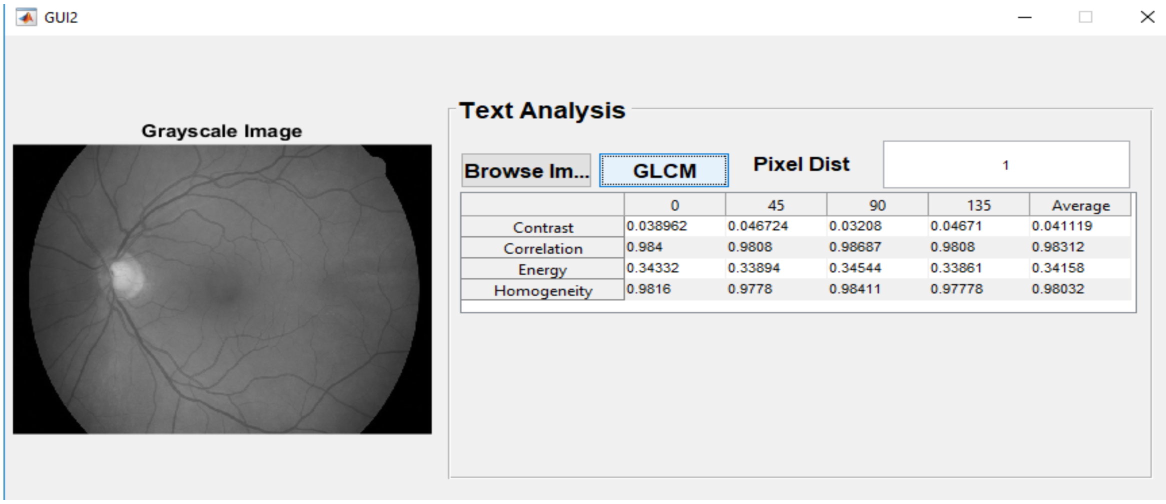
- From the above image we first go to the browse directory and locate where the image is on our system directory.



STEP 2:

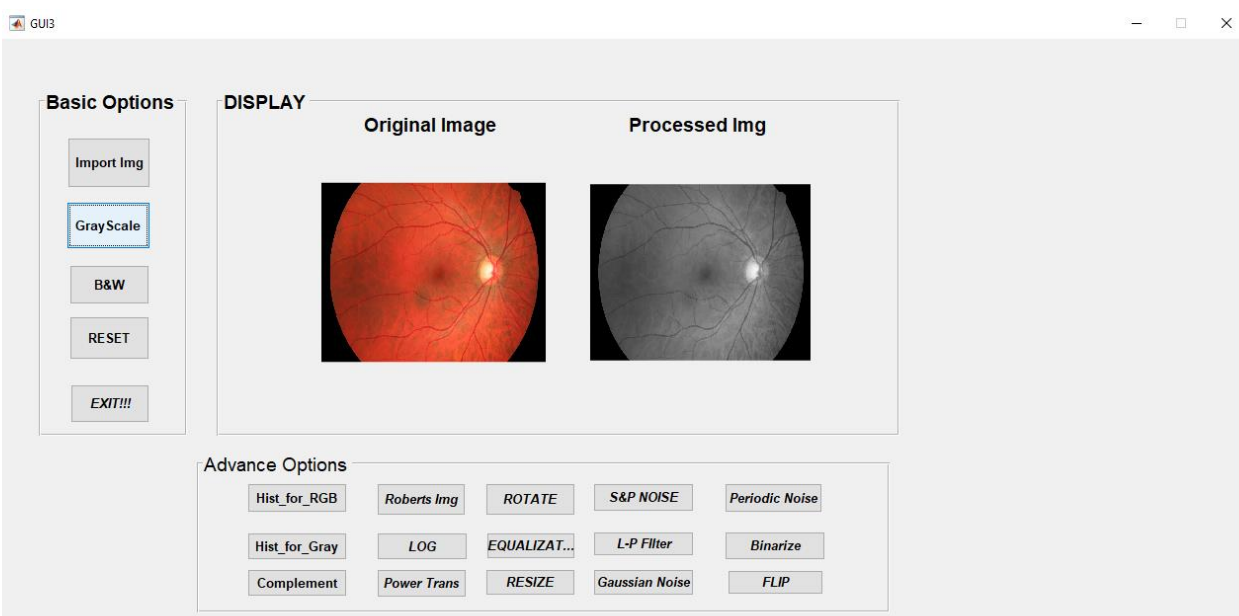
- This step has to do with the creation of GLCM code. The code his used in getting the feature of the fundus image. Which makes it easier for the network to understand.
- For this aspect a command line and a GUI was used in getting the necessary features.





From the above it can be seen that the feature extraction can be gotten in any form depending on the user. But a Graphic User Interface seems friendlier for non-programmers to understand better.

Also a GUI was created for image processing.



STEP 3:

- Tabulating of data is very import. All the features extracted are then recorded

	A	B	C	D	E	F
1	CONTRAST	CORRELAT	ENERGY	Homogen	ENTROPY	OUTPUT
2	0.0944	0.9671	0.2584	0.9546	1.6474	1
3	0.0817	0.9636	0.2808	0.9593	1.6774	1
4	0.0755	0.9685	0.2499	0.9628	1.688	1
5	0.0811	0.9684	0.2312	0.9598	1.7332	1
6	0.052	0.9772	0.286	0.9744	1.4939	1
7	0.0687	0.97	0.2936	0.9661	1.4788	1
8	0.0609	0.974	0.2934	0.9706	1.4974	1
9	0.0656	0.9744	0.3106	0.9682	1.479	1
10	0.0739	0.9717	0.3348	0.9648	1.458	1
11	0.0515	0.9789	0.3349	0.9751	1.3912	1
12	0.0732	0.9656	0.2614	0.9638	1.5694	1

With the data recorded it makes it easier for the model to understand and make prediction easier.

NOTE: A total of 45 Healthy and 45 Glaucoma dataset was used in carrying out this research.

STEP 4:

After all the features have been tabulated in the windows spreadsheet we then head to the Python IDE. Using the Anaconda environment.

The jupyter anaconda was used for this research cause of my familiarity with the User Interface. Note any other IDE can be used such as VScode, Spyder or ipython. What is most important is how comfortable you can be using the environment.

To download the Anaconda Environment go to your browser and search download anaconda for your OS type. Note if you get stuck at any point always search on ***Google and Youtube*** for solution.

REQUIREMENTS:

h5py==2.8.0
Keras==2.2.0
Keras-Applications==1.0.2
Keras-Preprocessing==1.0.1
numpy==1.14.5
PyYAML==3.12
scikit-learn==0.19.1
scipy==1.1.0
six==1.11.0
sklearn==0.0
tensorflow == 1.13.1

I. Importing the necessary libraries

Importing All Necessary Features

In [49]:
```
import numpy as np
from sklearn import preprocessing,neighbors
import pandas as pd
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

II. Giving a variable to the dataset. That is representing the data as train

Giving the Glaucoma data a variable

In [50]:
```
train = pd.read_csv('Glaucoma.csv')
train.replace('?', -99999, inplace=True)
train.head() # the first 5 values
```

Out[50]:

	CONTRAST	CORRELATION	ENERGY	Homogeneity	ENTROPY	OUTPUT
0	0.0944	0.9671	0.2584	0.9546	1.6474	1
1	0.0817	0.9636	0.2808	0.9593	1.6774	1
2	0.0755	0.9685	0.2499	0.9628	1.6880	1
3	0.0811	0.9684	0.2312	0.9598	1.7332	1
4	0.0520	0.9772	0.2860	0.9744	1.4939	1

III. Split the data into input and output features.

X= features

Y= labels

Creating Input (X) and Output(y) value of the dataset

In [52]:
```
X = np.array(train.drop('OUTPUT', axis=1))
y = np.array(train['OUTPUT'])
```

In [53]:
```
X.shape, y.shape # Getting the shape of the data
```

Out[53]:
```
((90, 5), (90,))
```

In [54]:
```
train.info() # getting necessary info of type of data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 6 columns):
CONTRAST       90 non-null float64
CORRELATION    90 non-null float64
ENERGY         90 non-null float64
Homogeneity    90 non-null float64
ENTROPY        90 non-null float64
OUTPUT         90 non-null int64
dtypes: float64(5), int64(1)
memory usage: 4.3 KB
```

In [55]:
```
train.describe() # Getting the description of the data set
```

IV. Next we split the data to train and test set and after we standardized the dataset. We can use different technique for this aspect.

Split data into train and test set. with test set of 20%

In [57]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state= 0)
```

Data scaling and standardization

In [58]:
```
#Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

V. We then created the neural network of 6 hidden layers. The *keras* library was used with *tensorflow* backend. Note we can use either *theano* or tensorflow, but for this research the *tensorflow* was used.

Relu (Rectified linear) was used for activation

Dropout of 20% was used to prevent *overfitting* of the network

The first number of neuron was set as 16 neuron and increases gradually for each layers

The output layer activation was set as *Sigmoid* with two output 1, 0

Creatin the neural network using keras

and tensorflow as backend or we can use Theano

In [59]:
```
import keras
from keras import backend as k
from keras.models import Sequential
from keras.layers.core import Dense, Dropout
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.callbacks import EarlyStopping

classifier = Sequential() # Initialising the ANN

n_cols = X.shape[1] # the X.shape is the number of input that would be inside the network

classifier.add(Dense(16, input_dim=n_cols, activation='relu')) # input layer requires input_dim param
classifier.add(Dense(32, activation='relu'))
#classifier.add(Dropout(0.5))
classifier.add(Dense(64, activation='relu'))
classifier.add(Dense(32, activation='relu'))
classifier.add(Dense(16, activation='relu'))
classifier.add(Dropout(.2))
classifier.add(Dense(2, activation='sigmoid')) # sigmoid instead of relu for final probability between 0 and 1
```

VI. The next gives a detail info of the network

```
In [60]: classifier.summary() # gives the summary of the data set
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 16)                96
_________________________________________________________________
dense_8 (Dense)              (None, 32)                544
_________________________________________________________________
dense_9 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_10 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_11 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 34
=================================================================
Total params: 5,394
Trainable params: 5,394
Non-trainable params: 0
_________________________________________________________________
```

VII. Setting the learning rate, loss and metrics of the model.

The learning rate was set at 0.0001, the loss(entropy) and matric(accuracy)

```
In [61]: classifier.compile(Adam(lr=.0001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Early stopping usually know as drawback was set a 3

which means if the accuracy remains the same after 3 iteration the training would stop.

The draw back was later remove as it didnt allow the model to train more

```
In [62]: early_stopping_monitor = EarlyStopping(patience=3)
```

VIII. Fitting the model with the classifier

```
In [64]: history = classifier.fit(X, y, validation_split=0.2, batch_size = 1, epochs = 200, shuffle=True, verbose=2,  validation_data=(X
```

IX. Data Visualization

In [67]:
```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [68]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [70]:
```
%matplotlib inline
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
```

In [71]:
```
for i in Y_pred:
    print(i)    # data prediction for output
```

In [72]:
```
Y_pred_rounded = classifier.predict_classes(X_test, batch_size=10, verbose=0)
```

In [73]:
```
for i in Y_pred_rounded:
    print(i)
```

In [74]:
```
confusion_matrix(y_test, Y_pred_rounded) # Confusion matrix
```

Out[74]:
```
array([[6, 1],
       [6, 5]], dtype=int64)
```

X. The final set is to create a function for representation.

0 = Negative (Healthy), 1=Positive (Glaucoma)

```
In [83]: def converter(value):
             if value == 0:
                 return "Negative"
             else:
                 return "Positive"
```

Making prediction from the model

```
In [87]: ex_measures=np.array([0.07722,
         0.962,
         0.2792,
         0.9639,
         1.6119])
         ex_measures=ex_measures.reshape(1,-1)

         prediction = classifier.predict(ex_measures)
         print(converter(np.argmax(prediction)))

         Positive
```

It can be said the neural network gives an accurate answear

ANOTHER TECHNIQUE THAT WOULD BE USED IS CONVOLUTIONAL NEURAL NETWORK (CNN), PRE-TRAINED (TRANSFER LEARNING) MODEL.

ALSO WE CAN INCREASE THE ACCURACY BY INCREASING THE NUMBER OF FEATURES.

FUTHER RESEARCH WOULD BE CONDUCTED IN FUTURE TO KNOW HOW GOOD OUR MODEL CAN BE.

BIG APPRECIATION TO GOOGLE AND YOUTUBE!!! LOL