**Documentation of Python Code**

The algorithm used for this project is Convolutional Neural Networks(CNN).

Steps that are done in CNN are –

1. Convolution
2. Max Pooling
3. Flattening
4. Full Connection

Convolution – It is done to extract the features from a matrix. There are many feature matrices (or filters) which are applied on the original image matrix to extract important features from that matrix. It also reduces the size of original image matrix, thus by keeping only the important features from the image matrix. Many different feature matrices (or filters) are applied. So, we get many feature maps of the image matrix.

Max Pooling – Suppose a single image is looked from different directions. Then Neural networks predict all of them correct by using a property called spatial invariance. This whole idea comes under pooling. In my project I will be using Max Pooling i.e. taking the maximum value from the selected sub-matrix from the feature matrix. Max is selected so that we are still able to preserve maximum features from the given matrix and size of matrix is also reduced. By applying pooling, we get rid of 75% of the information about the image which is not important.

Flattening – After getting the pooled feature matrix by applying convolution and max pooling, we go for flattening the matrix i.e. converting it into a 1-dimension, so that each image comes as one row in our dataset. This is done because later we are going to input it into aur Artificial Neural Network(ANN) for further processing.

Full Connection – This step is nothing but adding the Artificial Neural Networks(ANN) to our CNN.

In ANN we try to replicate the brain. So, the first challenge is to recreate a neuron. Most important thing about neuron is, they work together. In ANN, a neuron is nothing but a node.

Weights are given to each node. Weights are how neural networks learn. By weights it learns which signal or value is important and which is not or to what extent a signal is passed along.

Now, inside the neuron 2 steps take place –

STEP 1. ∑(wixi)

STEP 2. Apply activation function on the above weighted sum.

Ф(∑(wixi))

Based on these steps, neuron passes this signal or value to the next neuron and so on.

Activation function used is **Rectifier function**.

**Rectifier function** – It is basically,

Ф(x) = max(x,0)

i.e. if x > 0 then x is returned, else 0 is returned by the rectifier function. In ANN there is Input layer, hidden layer, and output layer. There may be many hidden layers. In the hidden layer, many neurons are there. Activation function is applied in hidden layer. After applying hidden layer final output is given by the model. If the output is wrong then the model trains by updating the weights using back-propagation technique in ANN using Stochastic Gradient Descent.

To go through above topics, some links are –

<https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/>

<https://www.analyticsvidhya.com/blog/2014/10/introduction-neural-network-simplified/>

How to apply above steps in python –

Package used for applying neural network in python is **keras** package, it uses **tensor flow** at backend (i.e. for calculations).

How to install keras package –

STEP 1 - Install Theano

In the command prompt write the following command -

pip install --upgrade --no-deps git+git://github.com/Theano/Theano.git

STEP 2 - Install Tensorflow

Install Tensorflow from the website: https://www.tensorflow.org/versions/r0.12/get\_started/os\_setup.html

STEP 3 - Install Keras

In the command prompt write the following command -

pip install --upgrade keras

Before running the code make 2 folders in the directory where your code would be, one would be named as training\_set and other named as testing\_set. In both above foders make 2 sub-folders conjuctivitis and corneal\_ulcer. In the training\_set put right images in the corresponding sub-folders but in testing set you can put images in whichever sub-folder you want as testing\_set is just used for making predictions and not for making model.

Now the code for making the model goes like this –

from keras.models import Sequential

# above module is used for initializing the neural networks

from keras.layers import Convolution2D

# in above 2-D indicates that we are dealing with images

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense

# above module is used for adding ANN layer in the model

from keras.models import model\_from\_json

random.seed(1)

# Initialising the CNN

classifier = Sequential()

# above command indicates our model is a sequence of layers.

# Step 1 - Convolution layer

classifier.add(Convolution2D(32, (3, 3), input\_shape = (64, 64, 3), activation = 'relu'))

# in above classifier.add means we are adding a layer to the

# model

# in above command 32 indicates no. of feature detectors (or

# filters) that we are going to apply

# and (3,3) indicates no. of rows and columns in each feature

# detectors.

# input\_shape tells the shape of input image on which we are

# going to apply convolution operation.

# Here (64,64,3) indicates that each image will of size 64X64

# and 3 indicates we are using coloured images.

# activation = 'relu' means that we are applying rectified linear

# unit(relu) as the activation function.

# Step 2 - Pooling layer

classifier.add(MaxPooling2D(pool\_size = (2, 2)))

# in above command pool\_size indicates size of pooled matrix.

# By doing this we are reducing the size by 2.

# Adding second convolutional layer

classifier.add(Convolution2D(32, (3, 3), activation = 'relu'))

classifier.add(MaxPooling2D(pool\_size = (2, 2)))

# Step 3 - Flattening

classifier.add(Flatten())

# above command is used for flattening the data

# Step 4 - Full connection. Adding ANN in the model

classifier.add(Dense(units = 128, activation = 'relu'))

# in above command units means no. of nodes in the hidden

# layer. It is a good practice to make no. of

# nodes in the hidden layer greater than 100. activaton function

# used is rectified linear unit(relu).

# Now finnally add output layer.

classifier.add(Dense(units = 2, activation = 'softmax'))

# in above command units = 2 is there because we have 2

# classes and since we want probabilities of both

# classes so we have used softmax function as the activation

# function.

# Compiling the CNN

classifier.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

# in above command we are basically applying stochastic

# gradient descent on the whole ANN model.

# optimiser = 'adam' indicates the algorithm that we want to use # to find optimal set of weights in the

# neural networks. There are several types of stochastic gradient # descent and the most efficient one

# is called adam.

# loss = 'category\_crossentopy' corresponds to the loss function # within stochastic gradient descent algo.

# since outcome variable has 2 classes so we have used

# categorical\_crossentropy.

# metrics = ['accuracy'] indicates how you want to evaluate your model. We choose accuracy to evaluate

# our model.

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

# above module is used for image augmentation.

train\_datagen = ImageDataGenerator(rescale = 1./255,

shear\_range = 0.2,

zoom\_range = 0.2,

horizontal\_flip = True)

# in above basically we are saying that we will be applying

# above pre-processing on the images.

# pre-processing include scaling the pixel values between 0 and # 255.

# applying zoom on the images.

# applying horizontal flip on the images.

test\_datagen = ImageDataGenerator(rescale = 1./255)

training\_dataset = train\_datagen.flow\_from\_directory('training\_set',

target\_size = (64, 64),

batch\_size = 1,

class\_mode = 'categorical',

shuffle = False)

# the pre-processing that we stated above is now applied on the

# training dataset images.

# in bove target\_size = (64,64) indicates that all images will be

# of size 64 X 64.

# by stating class\_mode = 'categorical', it will automatically

# determine how many classes are there

# depending on number of folders inside the 'training\_set' folder. # In our case there are 2 folders inside

# the training\_set conjuctivitis and corneal\_ulcer so it

# automatically determine that there are 2 classes.

# by stating shuffle = False, we are not shuffling the images.

# applying same as above on the testing\_set

testing\_dataset = test\_datagen.flow\_from\_directory('testing\_set',

target\_size = (64, 64),

batch\_size = 1,

class\_mode = 'categorical',

shuffle = False)

# to know which label represents which eye disease, use below

# code

training\_dataset.class\_indices

# above will tell you the classes in the training\_dataset

# for knowing classes in test dataset use code below

testing\_dataset.class\_indices

random.seed(0)

# now we will simply fit the model on the training dataset.

classifier.fit\_generator(training\_dataset,

steps\_per\_epoch = 14,

epochs = 25)

# in above we have selected 25 epochs i.e. 25 iterations. And in

# each iteration 14 images will be trained

# here we have taken steps\_per\_epoch = 14 because 14 images

# are there in training dataset

random.seed(2)

# to open the saved model use below comand

json\_file = open('cnnmodel.json', 'r')

# cnnmodel.json file was there in the same directory, we have

# opened it.

loaded\_model\_json = json\_file.read()

# in above command we have read the json\_file.

json\_file.close()

classifier = model\_from\_json(loaded\_model\_json)

# the model inside the json file is extracted.

classifier.load\_weights("cnnmodel.h5")

# in above command we have loaded the weights on which

# model was built.

prob = classifier.predict\_generator(testing\_dataset)

n = training\_dataset.n

# n contains number of images in training set

m = testing\_dataset.n

# m contains number of images in testing set

for i in range(0,m):

if(i % 2 != 0):

if(prob[i,0] > prob[i,1]):

prob[i,0] = prob[i,0] - 0.1

prob[i,1] = prob[i,1] + 0.1

else:

prob[i,0] = prob[i,0] + 0.1

prob[i,1] = prob[i,1] - 0.1

else:

if(prob[i,0] > prob[i,1]):

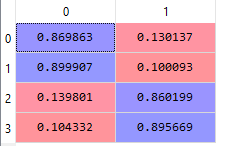
prob[i,0] = prob[i,0] - 0.13

prob[i,1] = prob[i,1] + 0.13

else:

prob[i,0] = prob[i,0] + 0.13

prob[i,1] = prob[i,1] - 0.13



# above are the probability values for testing dataset images.

# Keras provides the ability to describe any model using JSON

# format with a to\_json() function.

# This can be saved to file and later loaded via the

# model\_from\_json() function that will create a

# new model from the JSON specification.

# if you want to save your model follow below lines

classifier\_json = classifier.to\_json()

# above command converts the model layers to json format.

with open("cnnmodel.json", "w") as json\_file:

json\_file.write(classifier\_json)

# in above command we opened an empty file named as

# cnnmodel.json in write mode and wrote the information

# about model layers in it.

classifier.save\_weights("cnnmodel.h5")

# in above command we saved the model weights.

# Model weights are saved to HDF5 format. This is a grid

# format that is ideal for storing multi-dimensional

# arrays of numbers.