train\_dir='C:\\Users\\91906\\Images\\train'

test\_dir='C:\\Users\\91906\\Images\\test'

max\_count=100

reg\_val=[]

lr\_val=[]

test\_loss=[]

test\_acc=[]

for i in range(max\_count):

print ("\*"\*30)

print (str(i+1)+"/"+str(max\_count))

print ("\*"\*30)

# Sampling learning rate and regularization from a uniform distribution

reg=10\*\*(np.random.uniform(-4,0))

lr=10\*\*(np.random.uniform(-3,-4))

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

#Defining the architechture

model=models.Sequential()

model.add(layers.Conv2D(32,(3,3),activation='relu',input\_shape=(60,60,3)))

model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Conv2D(64,(3,3),activation='relu'))

model.add(layers.MaxPooling2D(2,2))

model.add(layers.Conv2D(128,(3,3),activation='relu'))

model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Conv2D(128,(3,3),activation='relu'))

model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Flatten())

model.add(layers.Dense(512,activation='relu',kernel\_regularizer=regularizers.l2(reg)))

model.add(layers.Dense(1,activation='sigmoid',kernel\_regularizer=regularizers.l2(reg)))

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Summazing the model

#model.summary()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Configuring the model for training

model.compile(loss='binary\_crossentropy',

optimizer=optimizers.RMSprop(lr=lr),

metrics=['acc'])

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Using the ImageDataGenerator class to read the..

# images from the directories

#Rescale all the images by 1/255

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

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

train\_generator=train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(60,60),

batch\_size=20,

class\_mode='binary')

test\_generator=test\_datagen.flow\_from\_directory(

test\_dir,

target\_size=(60,60),

batch\_size=20,

class\_mode='binary'

)

#Fit the model using batch generator

history=model.fit\_generator(

train\_generator,

steps\_per\_epoch=100,

epochs=5,

validation\_data=test\_generator,

validation\_steps=50)

reg\_val.append(reg)

lr\_val.append(lr)

test\_loss.append(history.history['val\_loss'])

test\_acc.append(history.history['val\_acc'])

#Save the model

#model.save('face\_vs\_nonface.h5')

#Plotting accuracy and loss

'''

acc=history.history['acc']

test\_acc=history.history['val\_acc']

loss=history.history['loss']

test\_loss=history.history['val\_loss']

epochs=range(1,len(acc)+1)

plt.plot(epochs,acc,'bo',label='TRAINING ACCURACY')

plt.plot(epochs,test\_acc,'b',label='TEST ACCURACY')

plt.title('TRAINING AND TEST ACCURACY')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.figure()

plt.plot(epochs,loss,'bo',label='TRAINING LOSS')

plt.plot(epochs,test\_loss,'b',label='TEST LOSS')

plt.title('TRAINING AND TESTING LOSS')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

'''

print ("\*"\*30)

print ("Finding the highest Test Accuracy and lowest Test Loss...")

index1=0

index2=0

max\_test\_acc=max(test\_acc[0])

min\_test\_loss=min(test\_loss[0])

for i in range(max\_count):

temp1=max(test\_acc[i])

if(temp1>=max\_test\_acc):

max\_test\_acc=temp1

index1=i

temp2=min(test\_loss[i])

if(temp2<min\_test\_loss):

min\_test\_loss=temp2

index2=i

print ('Maximum Testing Accuracy:',max\_test\_acc)

print ('Minimum Testing Loss:',min\_test\_loss)

print ('Value of optimum learning rate :',lr\_val[index1])

print ('Value of optimum regularization:',reg\_val[index2])