if not(os.path.exists('C:\\Users\\Shobi\\OneDrive\\Desktop\\PROJECT\\Signature-Forgery-Detection-main\\TestFeatures')):

os.mkdir('C:\\Users\\Shobi\\OneDrive\\Desktop\\PROJECT\\Signature-Forgery-Detection-main\\TestFeatures')

with open('C:\\Users\\Shobi\\OneDrive\\Desktop\\PROJECT\\Signature-Forgery-Detection-main\\TestFeatures\\testcsv.csv', 'w') as handle:

handle.write('ratio,cent\_y,cent\_x,eccentricity,solidity,skew\_x,skew\_y,kurt\_x,kurt\_y\n')

handle.write(','.join(map(str, feature))+'\n')

makeCSV()

n\_input = 9

train\_person\_id = input("Enter person's id : ")

test\_image\_path = input("Enter path of signature image : ")

train\_path = 'C:\\Users\\Shobi\\OneDrive\\Desktop\\PROJECT\\Signature-Forgery-Detection-main\\Features\\Training\\training\_' + train\_person\_id + '.csv'

testing(test\_image\_path)

test\_path = 'C:\\Users\\Shobi\\OneDrive\\Desktop\\PROJECT\\Signature-Forgery-Detection-main\\TestFeatures\\testcsv.csv'

def readCSV(train\_path, test\_path, type2=False):

df = pd.read\_csv(train\_path, usecols=range(n\_input))

train\_input = np.array(df.values)

train\_input = train\_input.astype(np.float32, copy=False)

df = pd.read\_csv(train\_path, usecols=(n\_input,))

temp = [elem[0] for elem in df.values]

correct = np.array(temp)

corr\_train = keras.utils.to\_categorical(correct,2)

df = pd.read\_csv(test\_path, usecols=range(n\_input))

test\_input = np.array(df.values)

test\_input = test\_input.astype(np.float32, copy=False)

if not(type2):

df = pd.read\_csv(test\_path, usecols=(n\_input,))

temp = [elem[0] for elem in df.values]

correct = np.array(temp)

corr\_test = keras.utils.to\_categorical(correct,2)

if not(type2):

return train\_input, corr\_train, test\_input, corr\_test

else:

return train\_input, corr\_train, test\_input

ops.reset\_default\_graph()

learning\_rate = 0.001

training\_epochs = 1000

display\_step = 1

n\_hidden\_1 = 7

n\_hidden\_2 = 10

n\_hidden\_3 = 30

n\_classes = 2

X = tf.placeholder("float", [None, n\_input])

Y = tf.placeholder("float", [None, n\_classes])

weights = {

'h1': tf.Variable(tf.random\_normal([n\_input, n\_hidden\_1], seed=1)),

'h2': tf.Variable(tf.random\_normal([n\_hidden\_1, n\_hidden\_2])),

'h3': tf.Variable(tf.random\_normal([n\_hidden\_2, n\_hidden\_3])),

'out': tf.Variable(tf.random\_normal([n\_hidden\_1, n\_classes], seed=2))

}

biases = {

'b1': tf.Variable(tf.random\_normal([n\_hidden\_1], seed=3)),

'b2': tf.Variable(tf.random\_normal([n\_hidden\_2])),

'b3': tf.Variable(tf.random\_normal([n\_hidden\_3])),

'out': tf.Variable(tf.random\_normal([n\_classes], seed=4))

}

def multilayer\_perceptron(x):

layer\_1 = tf.tanh((tf.matmul(x, weights['h1']) + biases['b1']))

layer\_2 = tf.add(tf.matmul(layer\_1, weights['h2']), biases['b2'])

layer\_3 = tf.add(tf.matmul(layer\_2, weights['h3']), biases['b3'])

out\_layer = tf.tanh(tf.matmul(layer\_1, weights['out']) + biases['out'])

return out\_layer

logits = multilayer\_perceptron(X)

loss\_op = tf.reduce\_mean(tf.squared\_difference(logits, Y))

optimizer = tf.compat.v1.train.AdamOptimizer(learning\_rate=learning\_rate)

train\_op = optimizer.minimize(loss\_op)

pred = tf.nn.softmax(logits)

correct\_prediction = tf.equal(tf.argmax(pred,1), tf.argmax(Y,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

init = tf.global\_variables\_initializer()

#import pyfiglet

#ascii\_art = pyfiglet.figlet\_format("Genuine Image")

#print('\033[91m' + ascii\_art + '\033[0m')

def evaluate(train\_path, test\_path, type2=False):

if not(type2):

train\_input, corr\_train, test\_input, corr\_test = readCSV(train\_path, test\_path)

else:

train\_input, corr\_train, test\_input = readCSV(train\_path, test\_path, type2)

ans = 'Random'

with tf.Session() as sess:

sess.run(init)

for epoch in range(training\_epochs):

\_, cost = sess.run([train\_op, loss\_op], feed\_dict={X: train\_input, Y: corr\_train})

if cost<0.0001:

break

accuracy1 = accuracy.eval({X: train\_input, Y: corr\_train})

if type2 is False:

accuracy2 = accuracy.eval({X: test\_input, Y: corr\_test})

return accuracy1, accuracy2

else:

prediction = pred.eval({X: test\_input})

if prediction[0][1]>prediction[0][0]:

print('\033[91mGenuine Image\033[0m')

return True

else:

print('\033[91mForged Image\033[0m')

return False

def trainAndTest(rate=0.001, epochs=1700, neurons=7, display=False):

start = time()

global training\_rate, training\_epochs, n\_hidden\_1

learning\_rate = rate

training\_epochs = epochs

n\_hidden\_1 = neurons

n\_hidden\_2 = 7

n\_hidden\_3 = 30

train\_avg, test\_avg = 0, 0

n = 10

for i in range(1,n+1):

if display:

print("Running for Person id",i)

temp = ('0'+str(i))[-2:]

train\_score, test\_score = evaluate(train\_path.replace('01',temp), test\_path.replace('01',temp))

train\_avg += train\_score

test\_avg += test\_score

if display:

print("Training average-", train\_avg/n)

print("Testing average-", test\_avg/n)

print("Time taken-", time()-start)

return train\_avg/n, test\_avg/n, (time()-start)/n

evaluate(train\_path, test\_path, type2=True)