CV Project 2

Name-Karanpreet Singh Wadhwa

NID-ksw352

UnivID-N13337853

I. Source Code- Human detection Source code.txt
Python File- Human detection project.py
HOG File- *crop001278a_Descriptor.txt (For crop001278a.bmp* positive training sample*)*
crop001045b_Descriptor.txt(For *crop001045b.bmp* positive test sample.))

II. Instructions:

a) Source Code – To compile and run the source code, follow these steps:

1. Copy the source code in a python compiling platform, example: Jupyter.
2. Make sure that the system has python interpreter installed
3. Before compiling, make sure that the libraries 'numpy', 'matplotlib' and 'opencv' are linked to the project
4. The image should be in the same directory as the source code.
5. The code will run, It saves various output images and Hog descriptor text file in the same directory where source code is.

III.

a) How did you initialize the weight values of the network?

I have initialised the weight of my network randomly by using np.random.randn(). Then I have factored it according to the dimension of the data so that the weight should not cause any problem in sigmoid function. As from my experience I have seen the if I don't do this input to sigmoid goes either greater the 10 or less than -10 which result in sigmoid giving only 0 or 1.

For example:

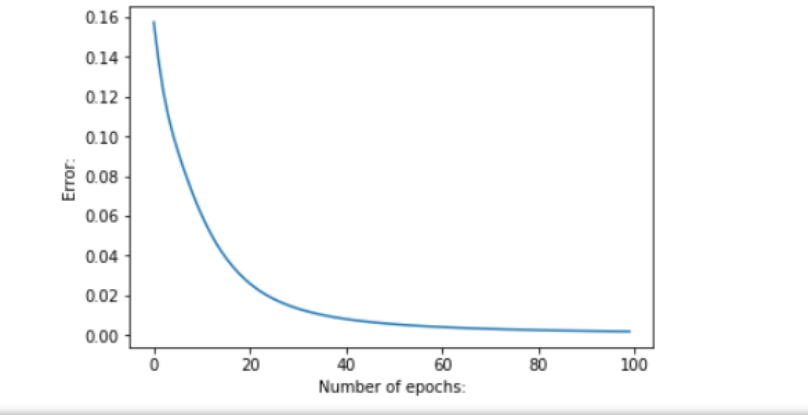
```
#This is how I declared the weights
layer1_weight=np.random.randn(n_input,n_hidden)
layer1_weight0=np.random.randn(n_hidden)
layer2_weight=np.random.randn(n_hidden,n_output)
layer2_weight0=np.random.randn(n_output)

#This is how I factorized it.
#FOR LAYER1
layer1_weight=np.multiply(layer1_weight,math.sqrt(2/int(n_input+n_hidden)))
layer1_weight0=np.multiply(layer1_weight0,math.sqrt(2/int(n_hidden)))
#FOR LAYER2
layer2_weight=np.multiply(layer2_weight,math.sqrt(1/int(n_hidden+n_output)))
layer2_weight0=np.multiply(layer2_weight0,math.sqrt(1/int(n_output)))
```

b) How many iterations (or epochs) through the training data did you perform?

I performed 100 epochs through my training set or less than 100 if error gets dropped to less then 0.01 before 100 epochs.

c) How did you decide when to stop training?
I decide to stop training when the error becomes less than 0.01 or it does not change much because this means that the weights and network outputs do not change much hence the squared errors do not change. So I compute Mean square error and used it as a measure to decide when to stop training.
Plus, I plotted the Number of epoch vs error to get more visualisation on how error is changing with Number of epoch and then set the value of epoch accordingly (If current one doesn't work).

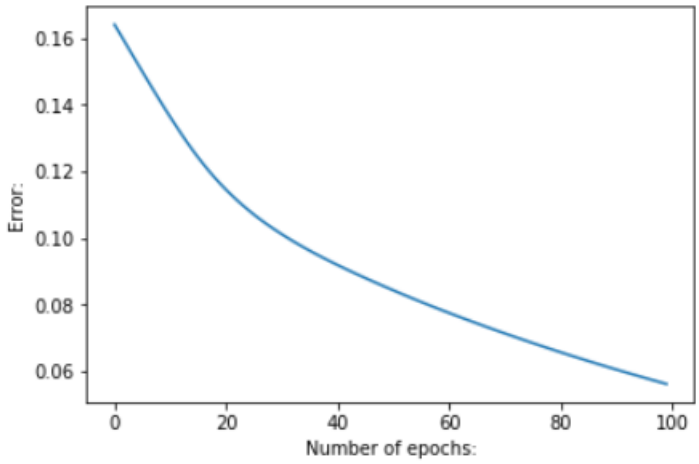


d) Based on the output value of the output neuron, how did you decide on how to classify the input image into human or not-human?
As output Neuron has sigmoid as activation function in it. So, if the output of this Neuron is greater than 0.5 then it should be classified as human otherwise Not-Human.
If y>0.5: classify as Human
Else: classify as Non-Human.

IV. Table that contains the output value of the output neuron and the classification result (human or not-human) for all 10 test images.
For Number of Hidden nodes=250
eta=0.003
Epochs=100

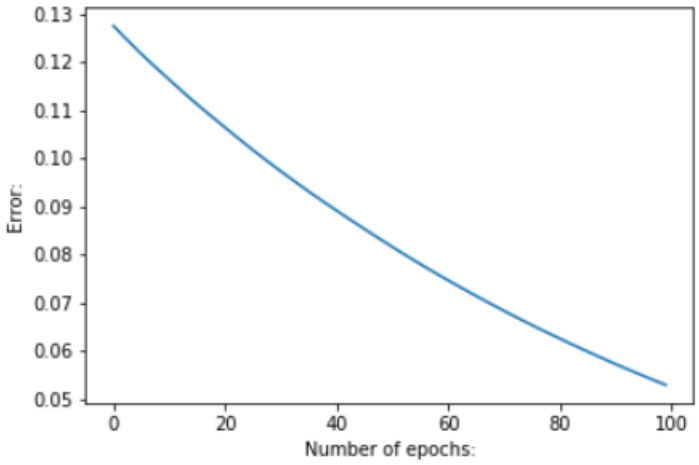
Test Image	Output value	Classification
crop_000010b	0.71120761	Human
crop001008b	0.73142849	Human
crop001028a	0.8858805	Human
crop001045b	0.47241023	**Not-Human**
crop001047b	0.76148739	Human
00000053a_cut	0.39186545	Not- Human
00000062a_cut	0.47467639	Not- Human
00000093a_cut	0.26071888	Not- Human
no_person__no_bike_213_cut	0.42763258	Not- Human

no_person__no_bike_247_cut	0.34104796	Not- Human



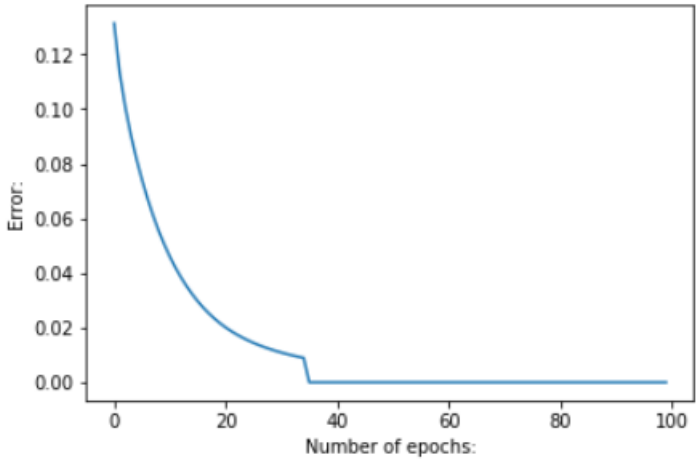
For Number of Hidden nodes=500
eta=0.003
Epochs=100

Test Image	Output value	Classification
crop_000010b	0.68333046	Human
crop001008b	0.72762083	Human
crop001028a	0.8296356	Human
crop001045b	0.50374031	**Human**
crop001047b	0.74443425	Human
00000053a_cut	0.39640844	Not- Human
00000062a_cut	0.46162108	Not- Human
00000093a_cut	0.22570673	Not- Human
no_person__no_bike_213_cut	0.43240517	Not- Human
no_person__no_bike_247_cut	0.38116479	Not- Human



For Number of Hidden nodes=1000
eta=0.01
Epochs=100

Test Image	Output value	Classification
crop_000010b	0.75799276	Human
crop001008b	0.76376089	Human
crop001028a	0.89771939	Human
crop001045b	0.44901226	**Not-Human**
crop001047b	0.79620869	Human
00000053a_cut	0.40982141	Not- Human
00000062a_cut	0.48140301	Not- Human
00000093a_cut	0.29536874	Not- Human
no_person__no_bike_213_cut	0.40031863	Not- Human
no_person__no_bike_247_cut	0.42567117	Not- Human



V.

- When I have taken larger number of Hidden node I tried to take a bigger value of eta like for Hidden Node =1000 I took Atta =0.01 instead of 0.003 so that it converge fast.
- I have tried with multiple value of MSE to stop neural net in training and choose 0.01 as the value to stop training of neural net. Because I didn't want to overfit or underfit the training data as this value looked intuitively best to me after seeing results.
- I have normalized the random weights according to the dimension of the matrix so that it does not cause problem in sigmoid function.

VI. Normalized gradient magnitude images for all 10 test images.

Fig.1 00000053a_cut Image and Normalized Gradient Image

Fig.2 00000062a_cut Image and Normalized Gradient Image

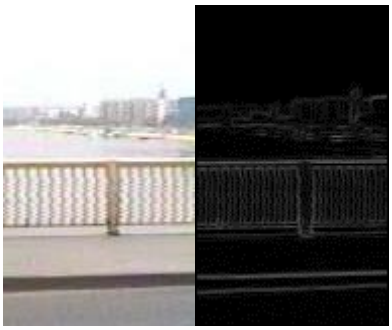


Fig.3 00000093a_cut Image and Normalized Gradient Image

Fig.4 no_person__no_bike_213_cut Image and Normalized Gradient Image

Fig.5 no_person__no_bike_247_cut Image and Normalized Gradient Image

Fig.6 crop_000010b Image and Normalized Gradient Image

Fig.7 crop001008b Image and Normalized Gradient Image

Fig.8 crop001028a Image and Normalized Gradient Image

Fig.9 crop001045b Image and Normalized Gradient Image

Fig.10 crop001047b Image and Normalized Gradient Image

VII. Source code

```
import cv2
```

```
from matplotlib import pyplot as plt
import numpy as np
import math

# Prewitt GX and GY
def Compute_GX_GY(IMAGE):
    Image_n,Image_m=IMAGE.shape
    start_n=1
    start_m=1
    end_n=Image_n-1#it should be -2 but range would not take last value of it so will give it one more
    end_m=Image_m-1##it should be -2 but range would not take last value of it so will give it one more

    Prewitt_gx=np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
    Gx_n,Gx_m=Prewitt_gx.shape
    GX=np.zeros((Image_n,Image_m),dtype=np.float)

    Prewitt_gy=np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
    Gy_n,Gy_m=Prewitt_gy.shape
    GY=np.zeros((Image_n,Image_m),dtype=np.float)

    for II in range(start_n,end_n):
        for JJ in range(start_m,end_m):
            SetX=0
            SetY=0

            I_Temp=II-1
            J_Temp=JJ-1
            for I1 in range(Gx_n):
                J_Temp=JJ-1
                for J1 in range(Gx_m):
                    # print(II,JJ,I1,J1,I_Temp,J_Temp)
                    SetX+=(Prewitt_gx[I1][J1])*(IMAGE[I_Temp][J_Temp])
                    SetY+=(Prewitt_gy[I1][J1])*(IMAGE[I_Temp][J_Temp])
```

```
                J_Temp+=1
            I_Temp+=1
        GX[II][JJ]=SetX
        GY[II][JJ]=SetY
    GX_NORMALIZED=np.abs(GX)/3
    GY_NORMALIZED=np.abs(GY)/3



    return GX,GY,GX_NORMALIZED,GY_NORMALIZED

# Function for computing Prewitt's MAGNITUTE
def COMPUTE_MAGNITUDE(IMAGE,GX_NORMALIZED,GY_NORMALIZED,o):
    IMAGE_N,IMAGE_M=IMAGE.shape
    MAGNITUDE=np.zeros((IMAGE_N,IMAGE_M),dtype=np.float)
    NORMALIZED_MAGNITUDE=np.zeros((IMAGE_N,IMAGE_M),dtype=np.int)
    for i in range(IMAGE_N):
        for j in range(IMAGE_M):
            MAGNITUDE[i][j]=np.hypot(GX_NORMALIZED[i][j],GY_NORMALIZED[i][j]) # To calculate magintude

    NORMALIZED_MAGNITUDE=(np.round(MAGNITUDE/(np.sqrt(2)))) # Hold normalized magnitue after
applying prewitt operation

    #print("--------Normalized Magnitude Image after Prewitt operation--------")
    plt.imshow(NORMALIZED_MAGNITUDE)
    plt.show()

    cv2.imwrite('NORMALIZED_MAGNITUDE_Image_%d.bmp'%(o),NORMALIZED_MAGNITUDE)

    return NORMALIZED_MAGNITUDE

# Function for computing Gradient Angle
def COMPUTE_ANGLE(IMAGE,GX,GY):
    IMAGE_N,IMAGE_M=IMAGE.shape
```

```
    ANGLE=np.zeros((IMAGE_N,IMAGE_M),dtype=np.float)
    ANGLE=np.arctan2(GY,GX)* 180 / np.pi # arctan2 returns in radians so to convert it into degree we did this
    ANGLE=ANGLE+360
    ANGLE=np.fmod(ANGLE,360)# To deal with negative values of angle

    return ANGLE


#make angle between 0-180
def Compute_scaled_angle(Angle):
    IMAGE_N,IMAGE_M=Angle.shape
    Scaled_angle=np.zeros((IMAGE_N,IMAGE_M),dtype=np.float)
    for i in range(0,IMAGE_N):
        for j in range(0,IMAGE_M):
            if(Angle[i][j]>=170 and Angle[i][j]<350):
                Scaled_angle[i][j]=Angle[i][j]-180
            elif(Angle[i][j]>=350 and Angle[i][j]<360):
                Scaled_angle[i][j]=Angle[i][j]-360
            else:
                Scaled_angle[i][j]=Angle[i][j]

    return Scaled_angle


# 8*8 Cell  and 16 * 16 pixel block! we move a cell for new block
def Compute_Histogram1(NORMALIZED_MAGNITUDE,Scaled_Angle):
    I_n,I_m=NORMALIZED_MAGNITUDE.shape
    start_n=0
    start_m=0
    Block_move_size=8
    i=0
    j=0
    Descriptor=[]
```

```
while(i+16<=I_n):
    j=0
    while(j+16<=I_m):
        Block=[]
        Num_cell=0
        Cell_i=i
        Cell_j=j
        for Num_cell in range(4):
            Histogram_for_cell=np.zeros((9,1),dtype=np.float)

            for i1 in range(Cell_i,Cell_i+8):
                for j1 in range(Cell_j,Cell_j+8):
                    #print(i,j,i1,j1,Cell_i,Cell_j)
                    if(Scaled_Angle[i1][j1]==0):
                        #do
                        Histogram_for_cell[0][0]+=NORMALIZED_MAGNITUDE[i1][j1]
                    elif(Scaled_Angle[i1][j1]==20):
                        #do
                        Histogram_for_cell[1][0]+=NORMALIZED_MAGNITUDE[i1][j1]
                    elif(Scaled_Angle[i1][j1]==40):
                        #do
                        Histogram_for_cell[2][0]+=NORMALIZED_MAGNITUDE[i1][j1]
                    elif(Scaled_Angle[i1][j1]==60):
                        #do
                        Histogram_for_cell[3][0]+=NORMALIZED_MAGNITUDE[i1][j1]
                    elif(Scaled_Angle[i1][j1]==80):
                        #do
                        Histogram_for_cell[4][0]+=NORMALIZED_MAGNITUDE[i1][j1]
                    elif(Scaled_Angle[i1][j1]==100):
                        #do
                        Histogram_for_cell[5][0]+=NORMALIZED_MAGNITUDE[i1][j1]
                    elif(Scaled_Angle[i1][j1]==120):
                        #do
                        Histogram_for_cell[6][0]+=NORMALIZED_MAGNITUDE[i1][j1]
```

```
elif(Scaled_Angle[i1][j1]==140):
   #do
   Histogram_for_cell[7][0]+=NORMALIZED_MAGNITUDE[i1][j1]
elif(Scaled_Angle[i1][j1]==160):
   #do
   Histogram_for_cell[8][0]+=NORMALIZED_MAGNITUDE[i1][j1]
else:
   c1=-111
   c2=-111
   if(Scaled_Angle[i1][j1]>=-10 and Scaled_Angle[i1][j1]<0):
      dis=0-Scaled_Angle[i1][j]
      Factor_c1=(20-(np.abs(0+Scaled_Angle[i1][j1])))/20
      Factor_c2=(20-(np.abs(20+Scaled_Angle[i1][j1])))/20

      Histogram_for_cell[0][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
      Histogram_for_cell[8][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
   if(Scaled_Angle[i1][j1]>0 and Scaled_Angle[i1][j1]<20):
      # do for center 0 and 20
      c1=0
      c2=1
      Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-0)))/20
      Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-20)))/20

      Histogram_for_cell[0][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
      Histogram_for_cell[1][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
   elif(Scaled_Angle[i1][j1]>20 and Scaled_Angle[i1][j1]<40):
      #do for center 20 and 40
      c1=1
      c2=2
      Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-20)))/20
      Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-40)))/20

      Histogram_for_cell[1][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
      Histogram_for_cell[2][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
```

```
elif(Scaled_Angle[i1][j1]>40 and Scaled_Angle[i1][j1]<60):
    #do for center 40 and 60
    c1=2
    c2=3
    Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-40)))/20
    Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-60)))/20

    Histogram_for_cell[2][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
    Histogram_for_cell[3][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
elif(Scaled_Angle[i1][j1]>60 and Scaled_Angle[i1][j1]<80):
    #do for center 60 and 80
    c1=3
    c2=4
    Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-60)))/20
    Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-80)))/20

    Histogram_for_cell[3][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
    Histogram_for_cell[4][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
elif(Scaled_Angle[i1][j1]>80 and Scaled_Angle[i1][j1]<100):
    #do for center 80 and 100
    c1=4
    c2=5
    Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-80)))/20
    Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-100)))/20

    Histogram_for_cell[4][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
    Histogram_for_cell[5][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
elif(Scaled_Angle[i1][j1]>100 and Scaled_Angle[i1][j1]<120):
    #do for center 100 and 120
    c1=5
    c2=6
    Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-100)))/20
    Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-120)))/20
```

```
            Histogram_for_cell[5][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
            Histogram_for_cell[6][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
        elif(Scaled_Angle[i1][j1]>120 and Scaled_Angle[i1][j1]<140):
            #do for center 120 and 140
            c1=6
            c2=7
            Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-120)))/20
            Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-140)))/20

            Histogram_for_cell[6][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
            Histogram_for_cell[7][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
        elif(Scaled_Angle[i1][j1]>140 and Scaled_Angle[i1][j1]<160):
            #do for center 140 and 160
            c1=7
            c2=8
            Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-140)))/20
            Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-160)))/20

            Histogram_for_cell[7][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
            Histogram_for_cell[8][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
        elif(Scaled_Angle[i1][j1]>160 and Scaled_Angle[i1][j1]<170 ):
            #do for center 160 and 0
            c1=0
            c2=8
            Factor_c1=(20-(np.abs(Scaled_Angle[i1][j1]-180)))/20
            Factor_c2=(20-(np.abs(Scaled_Angle[i1][j1]-160)))/20

            Histogram_for_cell[0][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c1)
            Histogram_for_cell[8][0]+=(NORMALIZED_MAGNITUDE[i1][j1]*Factor_c2)
if(Num_cell==0):
    Cell_j+=8
elif(Num_cell==1):
    Cell_j=j
```

```
                Cell_i+=8
            elif(Num_cell==2):
                Cell_j+=8


            for xxx in Histogram_for_cell:
                Block.append(xxx[0])

        j=j+8


        Normalized=0
        for xx in Block:
            Normalized+=np.square(xx)

        Normalized=np.sqrt(Normalized)

        for o in range(len(Block)):
            if(Normalized==0):
                Block[o]=0
            else:
                Block[o]=Block[o]/Normalized

        for xx in Block:
            Descriptor.append(xx)

    i=i+8

  return Descriptor

def Compute_descriptor(IMAGE,i):
  b,g,r = cv2.split(IMAGE)
  New_Gray_Image=np.round(0.299*r+0.587*g+0.114*b)
  GX,GY,GX_NORMALIZED,GY_NORMALIZED=Compute_GX_GY(New_Gray_Image)
```

```
NORMALIZED_MAGNITUDE=COMPUTE_MAGNITUDE(New_Gray_Image,GX_NORMALIZED,GY_NORMALIZE
D,i)
   Angle=COMPUTE_ANGLE(NORMALIZED_MAGNITUDE,GX,GY)
   Scaled_Angle=Compute_scaled_angle(Angle)
   Descriptor=Compute_Histogram1(NORMALIZED_MAGNITUDE,Scaled_Angle)
   return Descriptor


def RELU(x):
   n,m=x.shape
   y=np.zeros((n,m),dtype=float)
   for i in range(n):
      for j in range(m):
         if(x[i][j]>0):
            y[i][j]=x[i][j]
      else:
            y[i][j]=0
   return y



def sigmoid(data):
   yy=1/(1+np.exp(-data))
   return yy

# I is matrix of input examples
# D is matrix of output examples
# n_hidden is number of nodes in hidden layer
# n_max is number of training epochs

def Net(I,D,n_hidden,eta,n_max):

   row_inp,col_inp=I.shape
   no_example=row_inp
   n_input=col_inp
   row_out,col_out=D.shape
```

```
n_output=col_out

#declared the weights for layer 1 and normalize it according to the dimensions
layer1_weight=np.random.randn(n_input,n_hidden)
layer1_weight=np.multiply(layer1_weight,math.sqrt(2/int(n_input+n_hidden)))

layer1_weight0=np.random.randn(n_hidden)
layer1_weight0=np.multiply(layer1_weight0,math.sqrt(2/int(n_hidden)))

#declared the weights for layer 2 and normalize it according to the dimensions
layer2_weight=np.random.randn(n_hidden,n_output)
layer2_weight=np.multiply(layer2_weight,math.sqrt(1/int(n_hidden+n_output)))

layer2_weight0=np.random.randn(n_output)
layer2_weight0=np.multiply(layer2_weight0,math.sqrt(1/int(n_output)))

err_curve=np.zeros((n_max,col_out))
#print(layer1_weight,layer1_weight0)
for n in range(n_max):

   sq_err_sum= np.zeros((1,n_output))

   for k in range(no_example):
      x=I[k,:].reshape([1,-1])
      z=RELU((x.dot(layer1_weight)+layer1_weight0))
      y=sigmoid(z.dot(layer2_weight)+layer2_weight0)



      err=(D[k,0]-y)

      sq_err_sum+= 0.5*np.square(err)

      # Calculated delta for layer 2 weights
      Delta_output=(-1*err)*(1-y)*y
```

```
        Delta_layer2=z.T.dot(Delta_output)
        Delta_layer20=np.sum(Delta_output,axis=0)

        #Did differential of z which is the output of Relu
        zz=np.zeros_like(z)
        for xyz in range(n_hidden):

          if(z[0][xyz]>0):
            zz[0][xyz]=1
          else:
            zz[0][xyz]=0

        # Calculated delta for layer 1 weights
        Delta_hidden= Delta_output.dot(layer2_weight.T)*zz
        Delta_layer1=x.T.dot(Delta_hidden)
        Delta_layer10=np.sum(Delta_hidden,axis=0)

        #Updated weights for layer1 and layer2
        layer2_weight-= eta*Delta_layer2
        layer2_weight0-= eta*Delta_layer20
        layer1_weight-= eta*Delta_layer1
        layer1_weight0-= eta*Delta_layer10

        err_curve[n] = sq_err_sum/no_example
    print('Epoch %d: err %f'%(n,np.mean(sq_err_sum)/no_example))

    # if error is less than 0.01 stop training
    if(np.mean(sq_err_sum)/no_example<0.01):
        break
#For plotting Number of epoch vs error
plt.plot(np.linspace(0,n_max-1,n_max),np.mean(err_curve,axis=1))
plt.xlabel('Number of epochs:')
plt.ylabel('Error:')
plt.show()
```

```
    return layer1_weight,layer1_weight0,layer2_weight,layer2_weight0,err_curve


def predict(w,wb,v,vb,Output_descriptor):
    Number_of_test_image,number_of_attribute=Output_descriptor.shape
    predict=[]
    for k in range(Number_of_test_image):
        x=Output_descriptor[k,:].reshape([1,-1])
        z=RELU((x.dot(w)+wb))
        y=sigmoid(z.dot(v)+vb)
        predict.append(y)
    return predict


def main():
    # for training image descriptors ----------------------------------------------
    Final_descriptor=[]
    y=[]
    #for positive 1-3
    IMAGE = cv2.imread('crop001030c.bmp')
    Descriptor=Compute_descriptor(IMAGE,0)
    Final_descriptor.append(Descriptor)
    y.append(1)

    IMAGE = cv2.imread('crop001034b.bmp')
    Descriptor=Compute_descriptor(IMAGE,0)
    Final_descriptor.append(Descriptor)
    y.append(1)

    IMAGE = cv2.imread('crop001063b.bmp')
    Descriptor=Compute_descriptor(IMAGE,0)
    Final_descriptor.append(Descriptor)
    y.append(1)
```

```
#1-3 negative
IMAGE = cv2.imread('01-03e_cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)

IMAGE = cv2.imread('00000003a_cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)

IMAGE = cv2.imread('00000057a_cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)

#4-6 positive
IMAGE = cv2.imread('crop001070a.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(1)

IMAGE = cv2.imread('crop001275b.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(1)

IMAGE = cv2.imread('crop001278a.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(1)

#4-6 negative
IMAGE = cv2.imread('00000090a_cut.bmp')
```

```
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)

IMAGE = cv2.imread('00000091a_cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)

IMAGE = cv2.imread('00000118a_cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)

#7-9 postive
IMAGE = cv2.imread('crop001500b.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(1)

IMAGE = cv2.imread('crop001672b.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(1)

IMAGE = cv2.imread('person_and_bike_026a.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(1)

#7-9 neagtive

IMAGE = cv2.imread('no_person__no_bike_219_cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
```

```
Final_descriptor.append(Descriptor)
y.append(0)

IMAGE = cv2.imread('no_person__no_bike_258_Cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)

IMAGE = cv2.imread('no_person__no_bike_259_cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)

#10 positive
IMAGE = cv2.imread('person_and_bike_151a.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(1)

#10 negative
IMAGE = cv2.imread('no_person__no_bike_264_cut.bmp')
Descriptor=Compute_descriptor(IMAGE,0)
Final_descriptor.append(Descriptor)
y.append(0)



final_input=np.zeros((20,len(Final_descriptor[0])),dtype=float)
for i in range(0,20):
    final_input[i]=Final_descriptor[i]

final_output=np.zeros((20,1),dtype=float)
for i in range(20):
    final_output[i][0]=y[i]
#print(final_output)
```

```
# for test Image descriptors----------------------------------------------

Output_descriptor=[]
real_output=[]



#for positive

IMAGE = cv2.imread('crop_000010b.bmp')
Descriptor_output=Compute_descriptor(IMAGE,6)
Output_descriptor.append(Descriptor_output)
real_output.append(1)

IMAGE = cv2.imread('crop001008b.bmp')
Descriptor_output=Compute_descriptor(IMAGE,7)
Output_descriptor.append(Descriptor_output)
real_output.append(1)

IMAGE = cv2.imread('crop001028a.bmp')
Descriptor_output=Compute_descriptor(IMAGE,8)
Output_descriptor.append(Descriptor_output)
real_output.append(1)

IMAGE = cv2.imread('crop001045b.bmp')
Descriptor_output=Compute_descriptor(IMAGE,9)
Output_descriptor.append(Descriptor_output)
real_output.append(1)

IMAGE = cv2.imread('crop001047b.bmp')
Descriptor_output=Compute_descriptor(IMAGE,10)
Output_descriptor.append(Descriptor_output)
real_output.append(1)
```

```
#for Negative
IMAGE = cv2.imread('00000053a_cut.bmp')
Descriptor_output=Compute_descriptor(IMAGE,1)
Output_descriptor.append(Descriptor_output)
real_output.append(0)

IMAGE = cv2.imread('00000062a_cut.bmp')
Descriptor_output=Compute_descriptor(IMAGE,2)
Output_descriptor.append(Descriptor_output)
real_output.append(0)

IMAGE = cv2.imread('00000093a_cut.bmp')
Descriptor_output=Compute_descriptor(IMAGE,3)
Output_descriptor.append(Descriptor_output)
real_output.append(0)

IMAGE = cv2.imread('no_person__no_bike_213_cut.bmp')
Descriptor_output=Compute_descriptor(IMAGE,4)
Output_descriptor.append(Descriptor_output)
real_output.append(0)

IMAGE = cv2.imread('no_person__no_bike_247_cut.bmp')
Descriptor_output=Compute_descriptor(IMAGE,5)
Output_descriptor.append(Descriptor_output)
real_output.append(0)

# for making descriptor for test images
final_test_input=np.zeros((len(Output_descriptor),len(Output_descriptor[0])),dtype=float)

for i in range(0,len(Output_descriptor)):
    final_test_input[i]=Output_descriptor[i]
```

```
#------------------------------training and predicting for different values of eta and hidden nodes



# final_input is matrix of input examples
# final_output is matrix of output examples
# 250 is number of nodes in hidden layer
# 0.003 is eta
# 100 is number of training epochs
#training and getting weights
print('for Hidden node = %d and eta = %f'%(250,0.003))
w,wb,v,vb,err_curve=Net(final_input,final_output,250,0.003,100)
#predicting
predicted_output=predict(w,wb,v,vb,final_test_input)

#Threesholding the output of sigmoid fucntion so that we can classify as human or not
pre=[]

for check in predicted_output:
    if(check >=0.5):
        pre.append(1)
    else:
        pre.append(0)

# for calculating accuracy of our model
correct=0
wrong=0

for i in range(len(pre)):
    if(pre[i]==real_output[i]):
        correct+=1
    else:
        wrong+=1

print('correct = %d'%correct)
```

```
print('wrong = %d'%wrong)


print('Accuracy is %f Percent.'%(correct*100/(correct+wrong)))



#-------------------------------------------------------------------------------------
# final_input is matrix of input examples
# final_output is matrix of output examples
# 500 is number of nodes in hidden layer
# 0.003 is eta
# 100 is number of training epochs
#training and getting weights
print('for Hidden node = %d and eta = %f'%(500,0.003))
w,wb,v,vb,err_curve=Net(final_input,final_output,500,0.003,100)
#predicting
predicted_output=predict(w,wb,v,vb,final_test_input)

#Threesholding the output of sigmoid fucntion so that we can classify as human or not
pre=[]

for check in predicted_output:
    if(check >=0.5):
        pre.append(1)
    else:
        pre.append(0)

# for calculating accuracy of our model
correct=0
wrong=0

for i in range(len(pre)):
    if(pre[i]==real_output[i]):
        correct+=1
    else:
```

```
        wrong+=1

print('correct = %d'%correct)
print('wrong = %d'%wrong)

print('Accuracy is %f Percent.'%(correct*100/(correct+wrong)))

#------------------------------------------------------------------------------
# final_input is matrix of input examples
# final_output is matrix of output examples
# 1000 is number of nodes in hidden layer
# 0.01 is eta
# 100 is number of training epochs
#training and getting weights
print('for Hidden node = %d and eta = %f'%(1000,0.01))
w,wb,v,vb,err_curve=Net(final_input,final_output,1000,0.01,100)
#predicting
predicted_output=predict(w,wb,v,vb,final_test_input)

#Threesholding the output of sigmoid fucntion so that we can classify as human or not
pre=[]

for check in predicted_output:
    if(check >=0.5):
        pre.append(1)
    else:
        pre.append(0)

# for calculating accuracy of our model
correct=0
wrong=0

for i in range(len(pre)):
    if(pre[i]==real_output[i]):
```

```
            correct+=1
        else:
            wrong+=1

    print('correct = %d'%correct)
    print('wrong = %d'%wrong)

    print('Accuracy is %f Percent.'%(correct*100/(correct+wrong)))



    #-------------------------------------------------------------------------------
    ########################################################
    #Final output for documents
    f= open("crop001278a_Descriptor.txt","w+")
    for x in Final_descriptor[8]:
        f.write('%.17f\n'%x)
    f.close()

    f= open("crop001045b_Descriptor.txt","w+")
    for x in Output_descriptor[3]:
        f.write('%.17f\n'%x)
    f.close()



if __name__=="__main__":
    main()
```