Computer Vision – Project 2
Human Detection

Name: Anmol Bora
Net id: arb790
N Number: N11599372

1. Source code: HumanDetection.ipynb;
   HoG files: hog_crop001045b.txt; hog_crop001278a.txt

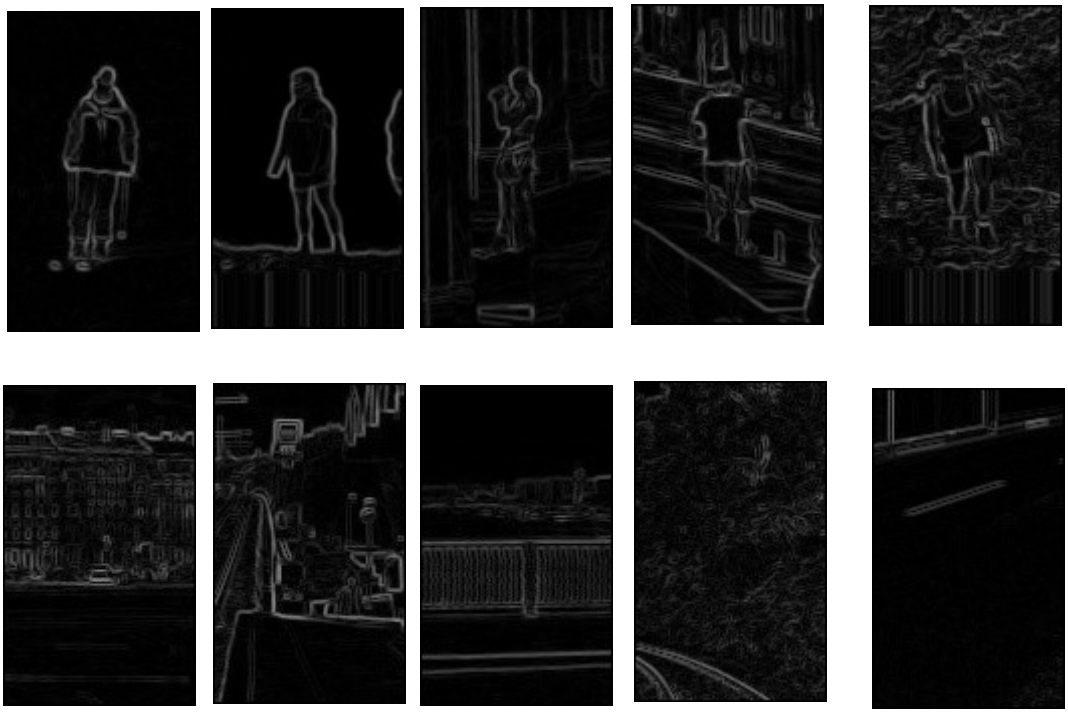
2. Run the source code in Jupyter Notebook. Make sure the path to the images is correctly set. It is currently set according to my location.

3.

    1. How did you initialize the weight values of the network? (Initialized using random.randn function according to the required shapes.)
    2. How many iterations (or epochs) through the training data did you peform? (Performed 500 iterations and saved weights gradually. Correctly predicted 9/10 test images.)
    3. How did you decide when to stop training? (Once the error change was extremely small and the output values were very firm (Close to 1 for postive test image and Close to 0 for negative test image)
    4. 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? (If the value is greater than 0.5, then human. If less than 0.5, then not-human)

Test Image	Output Value	Classification
crop_000010b	0.91245621	Human
crop001008b	0.97907213	Human
crop001028a	0.39695278	Not-Human
crop001045b	0.91555294	Human
crop001047b	0.99262075	Human
00000053a_cut	0.15154657	Not-Human
00000062a_cut	0.15897057	Not-Human
00000093a_cut	0.0093559	Not-Human
no_person__no_bike_213_cut	0.25399822	Not-Human
no_person__no_bike_247_cut	0.11755951	Not-Human

Magnitude Images:



Comments:

- The training function is not called because I have already trained the model. If you wish to train it yourself then you need to uncomment the weight and bias initialisation part (which is currently commented) and uncomment the part where the weights and biases are read from the respective .csv files. And then call the NN.begin() function which will train the model.

```
# coding: utf-8

# In[79]:



import matplotlib.pyplot as plt
import numpy as np
import imageio as i
import math
import cv2
import glob

def GradientOperator(img, op):
    ans = np.zeros_like(img,dtype=float) #gx and gy output
    image_padded = np.zeros((img.shape[0]+2, img.shape[1]+2)) #Add zero padding to the input image
    image_padded[1:-1, 1:-1] = img
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            #element-wise multiplication of respective horizontal and vertical operator and the image
            if (x<1 or x>=img.shape[0]-1) or (y<1 or y>=img.shape[1]-1):
                ans[x,y]=0
                #pixel values of first 4 rows 4 columns and last 4 rows 4 columns will be undefined
            else:
                ans[x,y]=(np.sum(op*image_padded[x:x+3,y:y+3]))/3
                #normalised by dividing by 3
    return ans

def histogram(angles, magnitudes):
    # [0, 20, 40, 60, 80, 100, 120, 140, 160]
    h = np.zeros(9, dtype=np.float32)

    for i in range(angles.shape[0]):
        for j in range(angles.shape[1]):
            if int(angles[i,j])<0:
                index_1 = 8
                index_2 = 0

                proportion = (index_2 * 20 - angles[i, j]) / 20
```

```
            value_1 = proportion * magnitudes[i, j]
            value_2 = (1 - proportion) * magnitudes[i, j]
            h[index_1] += value_1
            h[index_2] += value_2

        elif int(angles[i,j])>=160:
            index_1 = 0
            index_2 = 8

            proportion = (angles[i, j]-index_2 * 20) / 20

            value_1 = proportion * magnitudes[i, j]
            value_2 = (1 - proportion) * magnitudes[i, j]
            h[index_1] += value_1
            h[index_2] += value_2
        else:
            index_1 = int(angles[i, j] // 20)
            index_2 = int(angles[i, j] // 20 + 1)

            proportion = (index_2 * 20 - angles[i, j]) / 20

            value_1 = proportion * magnitudes[i, j]
            value_2 = (1 - proportion) * magnitudes[i, j]
            h[index_1] += value_1
            h[index_2] += value_2
    return h

def cells(trya,g):x
    #creating cells of size 8x8
    cells = []
    cell_size=8
    for i in range(0, np.shape(trya)[0],cell_size):
        row = []
        for j in range(0, np.shape(trya)[1],cell_size):
            row.append(np.array(
                histogram(trya[i:i + cell_size, j:j + cell_size], g[i:i + cell_size, j:j + cell_size]),
                dtype=np.float32))
        cells.append(row)
    return cells
```

```
def hog_descriptor(cells):
    #creating final hog vector
    hog_vector = []
    for i in range(0,np.shape(cells)[0]-1):
        for j in range(0,np.shape(cells)[1]-1):
            block_vector = []
            block_vector.extend(cells[i][j])
            block_vector.extend(cells[i][j + 1])
            block_vector.extend(cells[i + 1][j])
            block_vector.extend(cells[i + 1][j + 1])
            mag = lambda vector: math.sqrt(sum(i ** 2 for i in vector))
            magnitude = mag(block_vector)
            if magnitude != 0:
                normalize = lambda block_vector, magnitude: [element / magnitude for element in block_vector]
                block_vector = normalize(block_vector, magnitude)
            hog_vector.append(block_vector)
    hog = [item for sublist in hog_vector for item in sublist]
    return hog

def p1(image):
    #lets say this is the main function for the first part of the project i.e calculating HoG
    r, g, b = image[:,:,0], image[:,:,1], image[:,:,2]
    img = 0.299 * r + 0.587 * g + 0.114 * b
    px = np.array([[-1, 0, 1],
                   [-1, 0, 1],
                   [-1, 0, 1]]) #horizontal operator
    py = np.array([[1, 1, 1],
                   [0, 0, 0],
                   [-1, -1, -1]]) #vertical operator
    gx = GradientOperator(img, px) #horizontal gradient
    gx1=abs(gx) #we take absolute values for display purpose
    gy = GradientOperator(img, py) #vertical gradient
    gy1=abs(gy)
    g = (np.sqrt((gx1 * gx1) + (gy1 * gy1))/math.sqrt(2)) #normalise the magnitude by root(2)
    prewitt = (np.arctan2(gy, gx) * (180/np.pi)) #calculate the edge angles

    trya = np.copy(prewitt)
    for i in range(trya.shape[0]):
        for j in range(trya.shape[1]):
            if trya[i,j]<-10:
```

```
            trya[i,j]=360+trya[i,j]
        if trya[i,j]>=170 and trya[i,j]<350:
            trya[i,j]=trya[i,j]-180

    cells_c = cells(trya,g)
    hog_val = hog_descriptor(cells_c)
    return hog_val


# In[80]:


#creating the dataset
training_data_pos = []
testing_data_pos = []
training_data_neg = []
testing_data_neg = []

tr_p_files = glob.glob ('/home/anmol/CV_Project_2/Human/Train_Positive/*.bmp')
for myFile1 in tr_p_files:
    image1 = i.imread (myFile1)
    hog1=p1(image1)
    training_data_pos.append(hog1)

te_p_files = glob.glob ('/home/anmol/CV_Project_2/Human/Test_Positive/*.bmp')
for myFile2 in te_p_files:
    image2 = i.imread (myFile2)
    hog2=p1(image2)
    testing_data_pos.append(hog2)

tr_n_files = glob.glob ('/home/anmol/CV_Project_2/Human/Train_Negative/*.bmp')
for myFile3 in tr_n_files:
    image3 = i.imread (myFile3)
    hog3=p1(image3)
    training_data_neg.append(hog3)

te_n_files = glob.glob ('/home/anmol/CV_Project_2/Human/Test_Neg/*.bmp')
for myFile4 in te_n_files:
    image4 = i.imread (myFile4)
    hog4=p1(image4)
```

```
    testing_data_neg.append(hog4)


# In[81]:


#final training and testing dataset
training = np.concatenate([training_data_pos,training_data_neg])
testing = np.concatenate([testing_data_pos,testing_data_neg])


# In[91]:


X = np.asarray(training)
X_test = np.asarray(testing)
Y = np.array(([1], [1], [1], [1], [1], [1], [1], [1], [1], [1], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0]))
Y_test = np.array(([1], [1], [1], [1], [1], [0], [0], [0], [0], [0], ))
xPredicted = np.reshape(X_test[1],(7524,1)).transpose()


# In[92]:


class Neural_Network(object):

    def __init__(self):
        #parameters
        inputSize = 7524
        hiddenSize = 1000
        outputSize = 1

        #self.W1 = np.random.randn(inputSize, hiddenSize) *0.01 # (7524x500) weight matrix from input to hidden layer
        #self.W2 = np.random.randn(hiddenSize, outputSize)*0.01 # (500x1) weight matrix from hidden to output layer
        #self.B1 = np.random.randn(hiddenSize, 1)
        #self.B2 = np.random.randn(1,1)
        self.W1 = np.array(list(csv.reader(open("W1.csv", "rb"), delimiter=","))).astype("float")
        self.W2 = np.array(list(csv.reader(open("W2.csv", "rb"), delimiter=","))).astype("float")
        self.B1 = np.array(list(csv.reader(open("B1.csv", "rb"), delimiter=","))).astype("float")
        self.B2 = np.array(list(csv.reader(open("B2.csv", "rb"), delimiter=","))).astype("float")
```

```
    #saved weights and biases after training 500 epochs

def forward(self, X):
#forward propagation through our network
    self.z = np.dot(X, self.W1) + self.B1.T # dot product of X (input) and first set of 7524x500 weights
    self.z2 = self.relu(self.z) #relu activation function
    self.z3 = np.dot(self.z2, self.W2)+self.B2 # dot product of hidden layer (z2) and second set of 500x1 weights
    self.yhat = self.sigmoid(self.z3) #final activation function i.e. sigmoid
    return self.yhat

def sigmoid(self, s):
#sigmoid activation function
    return 1/(1+np.exp(-s))

def relu(self,s):
#relu activation function
    s[s<=0] = 0
    return s

def reluPrime(self,s):
#derivative of relu
    s[s<=0] = 0
    s[s>0] = 1
    return s

def sigmoidPrime(self, s):
#derivative of sigmoid
    return self.sigmoid(s) * (1 - self.sigmoid(s))

def backward(self, X, y, o):
#backward propagate through the network
    self.o_error = (self.o-y)
    olerror = np.multiply(self.o_error, self.sigmoidPrime(self.o))
    hlerror = np.multiply(olerror, self.W2.T)*self.reluPrime(self.z2)

    dw1 = np.dot(X.T,hlerror)
    dw2 = np.dot(self.z2.T,olerror)

    self.W1+=(-0.01*dw1)
    self.W2+=(-0.01*dw2)
```

```
    self.B1+=(-0.01*hlerror.T)
    self.B2+=(-0.01*olerror)

def train(self, x, y):
    self.o = self.forward(x)
    self.backward(x, y, self.o)

def saveWeightsandBiases(self):
    #saving the weights
    np.savetxt('W1.csv', self.W1, delimiter=',')
    np.savetxt('W2.csv', self.W2, delimiter=',')
    np.savetxt('B1.csv', self.B1, delimiter=',')
    np.savetxt('B2.csv', self.B2, delimiter=',')

def predict(self):
    #img = imageio.imread("/home/anmol/CV_Project_2/Human/Test_Positive/crop001008b.bmp")
    #plt.imshow(img, cmap=plt.cm.gray)
    print "Prediction based on trained weights: "
    print "Input \n" + str(xPredicted)
    print "Output: " + str(self.forward(xPredicted))

def test(self):
    correct=0
    pred_list=[]
    l2 = self.forward(X_test)
    print "Test Images: \n",l2
    for i in range(len(l2)):
        if l2[i]>=0.5:
            pred=1
        else:
            pred=0

        if pred == Y_test[i]:
            correct+=1
        pred_list.append(pred)

    ans = (float(correct)/float(len(Y_test)))*100
    print "Accuracy: ", ans

def begin(self):
```

```
        for i in xrange(11):# trains the NN 1,000 times
            for j in range(20):
                Xt = np.reshape(X[j],(7524,1))
                Yt = np.reshape(Y[j],(1,1))
        NN.train(Xt.T, Yt)
        if i%10==0:
            print "Epochs: {}, Loss: {}".format(i,str(np.mean(np.square(Y - NN.forward(X))))) # mean sum squared loss


NN = Neural_Network()


#NN.begin() #This is the training function which is commented as the network is already trained.
#NN.saveWeightsandBiases()
NN.predict()
NN.test()
```