Alexander Gao
awg297@nyu.edu
N17435149
Project 2: Human Detector Neural Network

(A)

Source Code: human-detector-awg297.py

HOG Printout Files:
crop001045b_HOG.txt
crop001278a_HOG.txt

(B)

Instructions:

Compile and run the provided .py Python Script.

Upon running the program:

- All of the training and test images will load in
- The neural network will train
- The test images will be classified
- The classification results are printed out

The number of training epochs can be modified by adjusting the parameter given to the function train_network that is called in the main part of the program. The training rate can be be adjusted by adjusting the global variable "train_rate".

Alexander Gao
awg297@nyu.edu
N17435149

(C)

How did you initialize the weight values of the network?

I initialized each weight as a unique random float in the range [0,1), using a numpy random number generator (numpy.random.rand). In this specific neural network, there are [500][7524] weights connecting the input and hidden layer, and [500] weights connecting the hidden layer and output layer.

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

I performed 150 epochs of training on the neural network, with a training rate of 0.05.

How did you decide when to stop training?

I decided to stop training when the **average error** for a given epoch changed minimally from one iteration to the next. This was a trial and error based approach, adjusting number of epochs, as well as experimenting with training rate, to find the optimal results. When I classified test images on the trained network, obviously a high accuracy rate further validated that my network had received sufficient training.

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 output value for a given test image was >= 0.5, I classified it as a human. If the output value < 0.5, I classified it as no-human.

(D)

Output neuron results:

Test Image	Output value	Classification
crop_000010b	0.5128826521843262	Human (1)
crop001008b	0.9730568125790874	Human(1)
crop001028a	0.6174900652193422	Human(1)
crop001045b	0.5200295952958314	Human (1)
crop001047b	0.07970313572184756	No Human (0)
00000053a_cut	0.224412223875482	No Human (0)
00000062a_cut	0.14147547474238112	No Human (0)
00000093a_cut	0.03259658278426859	No Human (0)
no_person__no_bike_213_cut	0.9688568795203464	Human(1)
no_person__no_bike_247_cut	0.010114113644258946	No Human (0)

Accuracy: 80%

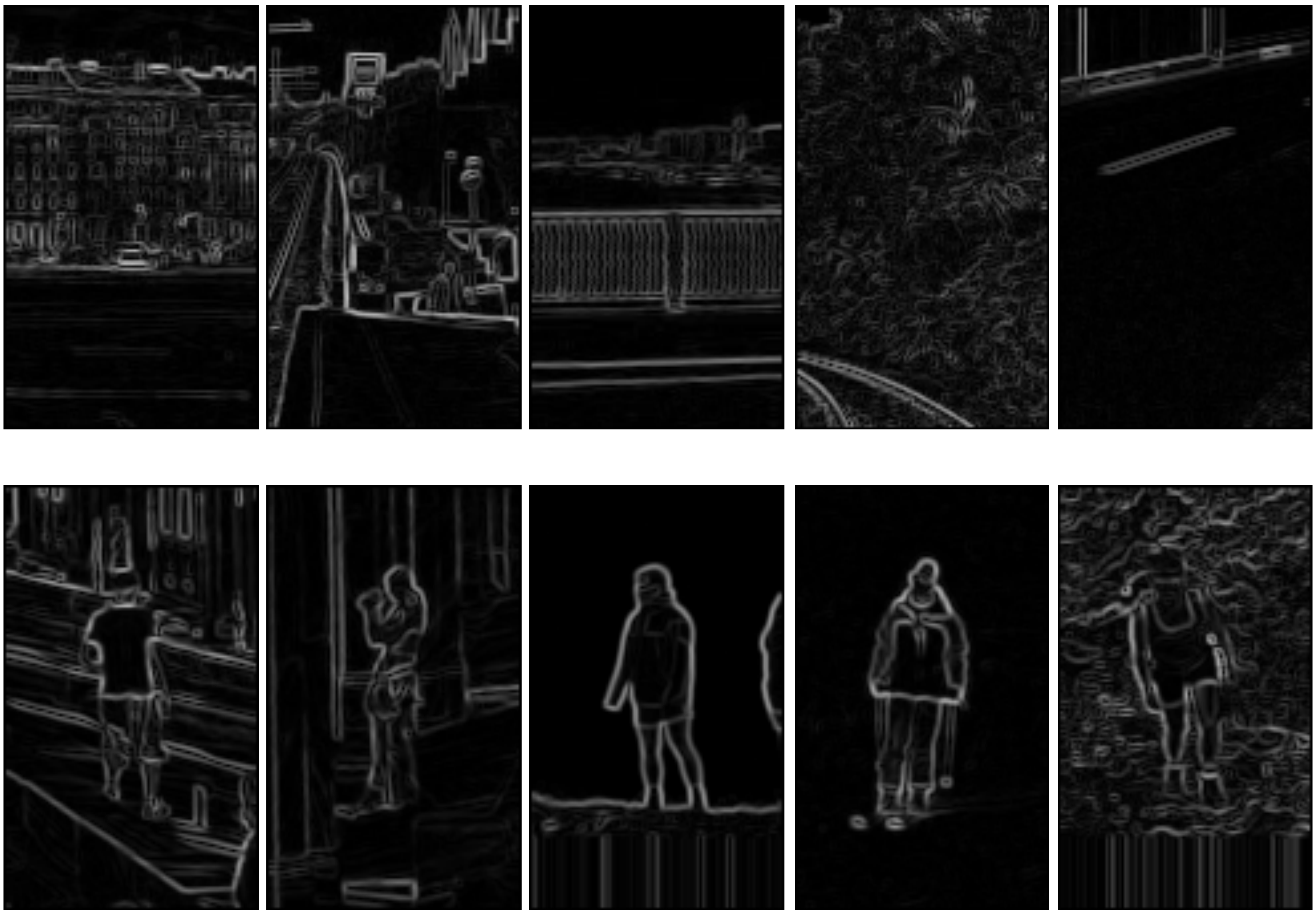
(E)

Additional comments: In order to find the optimal number of training epochs and training rate, I used a trial and error approach, eventually choosing those values to maximize my accuracy on classifying test images. By the end of training, all of the training images were generating output safely 100% within the correct classification, but this still did not guarantee a 100% accuracy rate when it came to then classifying test images.

Alexander Gao
awg297@nyu.edu
N17435149
Project 2: Human Detector Neural Network

(F)

Normalized Gradient Images



Alexander Gao
awg297@nyu.edu
N17435149
Project 2: Human Detector Neural Network

(G)

Source Code:

```
import math
import numpy as np
from PIL import Image
from random import seed
from random import random
import glob

#CREATE DIRECTORIES OF FILE PATHS TO ALL TRAINING AND TEST IMAGES
training_negative_paths = [img for img in glob.glob("Human/Train_Negative/*.bmp")]
training_positive_paths = [img for img in glob.glob("Human/Train_Positive/*.bmp")]
test_positive_paths = [img for img in glob.glob("Human/Test_Positive/*.bmp")]
test_negative_paths = [img for img in glob.glob("Human/Test_Neg/*.bmp")]

#HOG-------------------------------------------------------------------------------------------------------------------------------------------

#CONVERT RGB IMAGE TO GRAYSCALE
def make_grayscale(input_image_1):
    rgb_indexed = np.array(input_image_1)
    bw_image = np.zeros((rgb_indexed.shape[0],rgb_indexed.shape[1]), dtype = 'int')
    rgb_to_gray = np.array([0.299, 0.587, 0.114])                #RGB CHANNEL WEIGHTS
    for i in range(160):
        for j in range(96):
            bw_image[i,j] = round(np.sum(rgb_to_gray*rgb_indexed[i,j]))
    return bw_image

#PRODUCE IMAGE GRADIENT MAGNITUDE AND GRADIENT ANGLE
def gradient(input_image_2):
    dim = input_image_2.shape
    #CREATE NEW, SEPARATE MATRICES FOR GX, GY, GRADIENT MAGNITUDE, and GRADIENT ANGLE
    gx_arr = np.zeros(dim, dtype=np.float)
    gy_arr = np.zeros(dim, dtype=np.float)
    gradient_magnitude = np.zeros(dim, dtype='float')
    gradient_angle = np.zeros(dim, dtype = 'float')
    # PREWITT OPERATOR USED TO CALCULATE GX AND GY
    prewitt_gx = np.array([[-1,0,1],
                            [-1,0,1],
                            [-1,0,1]])
    prewitt_gy = np.array([[1,1,1],
                            [0,0,0],
                            [-1,-1,-1]])

    # 3X3 SUBMATRIX THAT WILL BE TAKEN AT EACH IMAGE PIXEL, AND PREWITT OPERATOR APPLIED TO
    img_submatrix = np.zeros((3,3))
    for i in range(1,(dim[0]-1)):
        for j in range(1,(dim[1]-1)):
            img_submatrix = input_image_2[i-1:i+2, j-1:j+2]
            gx_sum = 0
            gy_sum = 0
            gx_sum += np.sum(img_submatrix * prewitt_gx)
            gy_sum += np.sum(img_submatrix * prewitt_gy)
            gx_arr[i,j] = abs(gx_sum)/3
            gy_arr[i,j] = abs(gy_sum)/3
            gradient_magnitude[i,j] = math.sqrt(gx_arr[i,j]**2 + gy_arr[i,j]**2)
            gradient_angle[i,j] = math.degrees(math.atan2(gy_arr[i,j],gx_arr[i,j]))
    return (gradient_magnitude, gradient_angle)

#L2 NORMALIZATION PERFORMED ON EACH 36-DEGREE BLOCK VECTOR
def l2_normalize(block_input):
    l2_norm = math.sqrt(np.sum(block_input**2))
    if l2_norm == 0:
        return block_input
    else:
        normalized = block_input / l2_norm
    return normalized

#MAIN FUNCTION FOR PRODUCING HOG DESCRIPTOR
def hog_feature(image_path):
    #OPEN IMAGE
    new_image = Image.open(image_path)

    #PREPROCESS IMAGE
    image_bw = make_grayscale(new_image)
    (gradient_magnitude, gradient_angle) = gradient(image_bw)

    height = gradient_magnitude.shape[0]
    width = gradient_magnitude.shape[1]
    rows = height/8
    columns = width/8

    #INITIZALIZE A ROWSxCOLUMNS MATRIX TO STORE 9-BIN HISTOGRAM FOR EACH CELL IN IMAGE
    cell_histogram = np.empty(shape=(rows,columns,9))
    histogram_bin = np.zeros(9)
```

```
    #CREATE CELL HISTOGRAM MATRIX
    for i_start in range(0,height,8):
        for j_start in range(0,width,8):

            i_end = i_start + 8
            j_end = j_start + 8

            row = i_start/8
            column = j_start/8

            for i in range(i_start, i_end):
                for j in range(j_start, j_end):
                    angle = gradient_angle[i,j]

                    if angle < -10:
                        angle += 180

                    if angle >= -10 and angle <= 10:
                        weight_l = (10 - angle)/20
                        bin_index = (0,1)
                    elif angle >= 10 and angle <= 30:
                        weight_l = (30 - angle)/20
                        bin_index = (1,2)
                    elif angle >= 30 and angle <= 50:
                        weight_l = (50 - angle)/20
                        bin_index = (2,3)
                    elif angle >= 50 and angle <= 70:
                        weight_l = (70 - angle)/20
                        bin_index = (3,4)
                    elif angle >= 70 and angle <= 90:
                        weight_l = (90 - angle)/20
                        bin_index = (4,5)
                    elif angle >= 90 and angle <= 110:
                        weight_l = (110 - angle)/20
                        bin_index = (5,6)
                    elif angle >= 110 and angle <= 130:
                        weight_l = (130 - angle)/20
                        bin_index = (6,7)
                    elif angle >= 130 and angle <= 150:
                        weight_l = (150 - angle)/20
                        bin_index = (7,8)
                    elif ((angle >= 150 and angle <= 170)):
                        weight_l = (170 - angle)/20
                        bin_index = (8,0)

                    weight_r = 1 - weight_l

                    #POPULATE HISTOGRAM BINS USING WEIGHTED GRADIENT MAGNITUDES
                    histogram_bin[bin_index[0]] += gradient_magnitude[i,j]*weight_l
                    histogram_bin[bin_index[1]] += gradient_magnitude[i,j]*weight_r

            for x in range(9):
                cell_histogram[row,column,x] = histogram_bin[x]

    #CREATE BLOCK HISTOGRAM MATRIX FROM CELLS
    hog_output = np.empty(shape=0)
    for i in range(rows-1):
        for j in range(columns-1):
            block = np.empty(shape=0)
            block = np.concatenate((block,cell_histogram[i,j]),axis=None)
            block = np.concatenate((block,cell_histogram[i,j+1]),axis=None)
            block = np.concatenate((block,cell_histogram[i+1,j]),axis=None)
            block = np.concatenate((block,cell_histogram[i+1,j+1]),axis=None)
            block_normal = l2_normalize(block)
            hog_output = np.concatenate((hog_output,block_normal),axis=None)
    return hog_output


#NETWORK----------------------------------------------------------------------------------------------------------------------------------------

#INITIALIZE NETWORK VARIABLES

input_size = 7524
hidden_size = 500
train_rate = 0.03

a_input = np.empty(0)
a_hidden = np.empty(hidden_size)
a_output = 0.0
w_input = (np.random.rand(hidden_size,input_size))/hidden_size     #WEIGHTS ARE INITIALIZED TO A RANDOM VALUE IN RANGE [0,1)
w_hidden  = (np.random.rand(hidden_size))/hidden_size  #WEIGHTS ARE INITIALIZED TO A RANDOM VALUE IN RANGE [0,1)

in_i = 0.0
in_j = np.empty(hidden_size)
delta_i = 0.0
delta_j = np.empty(hidden_size)


#-------------------
#ACTIVATON FUNCTIONS
```

Alexander Gao
awg297@nyu.edu
N17435149
Project 2: Human Detector Neural Network

```
def relu(x):
    if x <= 0:
        return 0
    else:
        return x

def relu_derivative(x):
    if x <= 0:
        return 0
    else:
        return 1

def sigmoid(x):
    return (1/(1+ math.e**(-x) ))

def sigmoid_derivative(x):
    return sigmoid(x)*(1-sigmoid(x))

#--------
#TRAINING

def forward_propogate(hog):

    global a_input
    global a_hidden
    global w_input
    global w_hidden
    global in_i
    global in_j

    a_input = hog[0]

    #FORWARD PROPOGATION
    for j in range(hidden_size):
        in_j[j] = np.sum(a_input*w_input[j])
        a_hidden[j] = relu(in_j[j])
    in_i = np.sum(a_hidden*w_hidden)
    output = sigmoid(in_i)
    label = hog[1]
    print('output: ', output, ' / label: ', label)
    return (label, output)

#SUBROUTINE CALLED BY BACK_PROPOGATE_ERROR
def update_weights():
    global w_hidden
    global w_input
    global train_rate
    global a_hidden
    global a_input
    global delta_i
    global delta_j

    for j in range(hidden_size):
        w_hidden[j] +=      train_rate*a_hidden[j]*delta_i
    for j in range(hidden_size):
        for k in range(input_size):
            w_input[j][k] += train_rate * a_input[k] * delta_j[j]

def back_propogate_error(error):
    global delta_i
    global delta_j
    global in_i
    global in_j
    global w_hidden

    delta_i = error * sigmoid_derivative(in_i)
    for j in range(hidden_size):
        delta_j[j] = relu_derivative(in_j[j])*w_hidden[j]*delta_i
    update_weights()

#SUBROUTINE CALLED BY TRAIN-NETWORK -- RETURNS AVERAGE SQUARE ERROR OVER EACH EPOCH OF 20 IMAGES
def average_error(error_cache):
    error_sum = 0
    for i in range(20):
        error_sum += error_cache[i]**2/2
    average = error_sum/20
    return average

#MAIN DRIVING FUNCTION FOR TRAINING OUR NEURAL NETWORK
def train_network(hog_directory,epochs):
    error_cache = [0 for i in range(20)]
    for i in range(epochs):
        print('Epoch: ',i)
        #Iterate through the 20 training images
        for j in range(20):
            result = forward_propogate(hog_directory[j])
            error_cache[j] = result[0] - result[1]
            back_propogate_error(error_cache[j])
        print('average error ',average_error(error_cache))

#------------------
```

Alexander Gao
awg297@nyu.edu
N17435149
Project 2: Human Detector Neural Network

```
#ONCE NETWORK HAS BEEN TRAINED, CLASSIFY IS CALLED TO GENERATE OUTPUT FOR TEST IMAGES
def classify(hog):
    result = forward_propogate(hog)
    if result[1] >=0.5:
        return 1
    else:
        return 0


#MAIN-------------------------------------------------------------------------------------------------------------------------------------------

#MAIN ROUTINE THAT LOADS ALL IMAGES, CONVERTS THEM TO HOG, TRAINS NETWORK, AND CLASSIFIES TEST IMAGES
def main():

    hog_directory = []
    hog_test_directory = []

    for i in range(10):
        hog = hog_feature(training_positive_paths[i])
        hog_directory.append([hog,1])
        hog = hog_feature(training_negative_paths[i])
        hog_directory.append([hog,0])

    # exit()

    for i in range(5):
        # print(test_negative_paths[i],"  ",i)
        # print(test_positive_paths[i],"  ",i)
        hog = hog_feature(test_negative_paths[i])
        hog_test_directory.append([hog,0])
        hog = hog_feature(test_positive_paths[i])
        hog_test_directory.append([hog,1])


    #TRAIN NETWORK USING HOG FEATURES FROM 20 TRAINING IMAGES
    train_network(hog_directory,200)

    count_correct = 0
    for i in range(10):
        result = classify(hog_test_directory[i])
        if result == hog_test_directory[i][1]:
            count_correct += 1
    print(count_correct)

main()
```