COMPUTER VISION AND SCENE ANALYSIS

PROJECT 2: HUMAN DETECTION

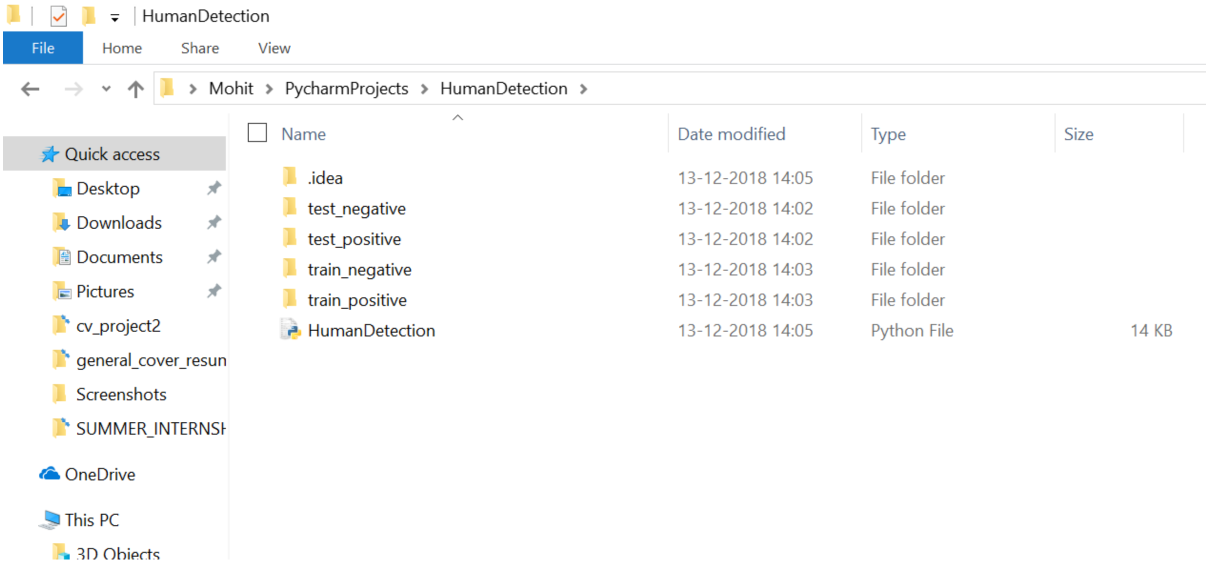
NAME: MOHIT SURESH PATEL

NETID: MSP552

N#: N17217391

EMAIL ID: MSP552@NYU.EDU

FILE AND FOLDER NAMES:



FOLDERS CONTAINING IMAGES:

Images are as per provided in nyu classes by Professor. Edward Wong as per their relavance

test_negative – negative images for testing

test_positive - positive images for testing

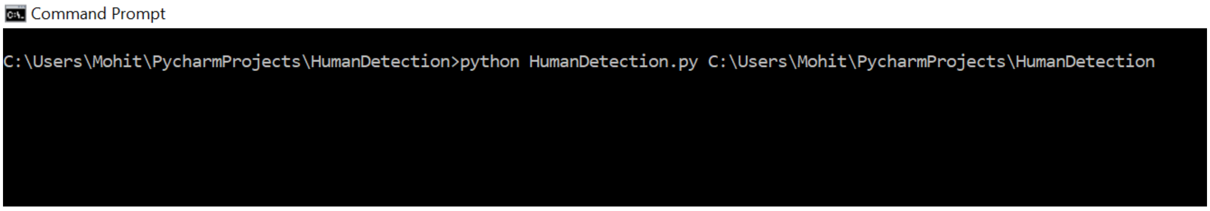
train_negative - negative images for training

train_positive - positive images for training

MAIN FILE TO BE EXECUTED IS:

HumanDetection.py

EXECUTING THE MAIN FILE:



Execute the following command as per the relevant path

C:\Users\Mohit\PycharmProjects\HumanDetection>python HumanDetection.py C:\Users\Mohit\PycharmProjects\HumanDetection

REQUIRED LIBRARIES:

Numpy, matplotlib, cv2, opencv, pillow, os, glob, math

OUTPUT RESULTS:

EPOCHS USED: 100

LEARNING RATE USED: 0.01

LABEL 1 IS HUMAN

LABEL 0 IS NON HUMAN

NEURONS IN HIDDEN LAYER: 250

```
Epoch Count: 95 Average Error:  0.003232432648371327
Epoch Count: 96 Average Error:  0.0031523684028436235
Epoch Count: 97 Average Error:  0.003076184960525245
Epoch Count: 98 Average Error:  0.0030018696873219453
Epoch Count: 99 Average Error:  0.0029314512556101935
Predicted Probability: [[0.5252004]] Actual Probability Value: [1]
Predicted Probability: [[0.67746966]] Actual Probability Value: [1]
Predicted Probability: [[0.5393744]] Actual Probability Value: [1]
Predicted Probability: [[0.59820027]] Actual Probability Value: [1]
Predicted Probability: [[0.60794597]] Actual Probability Value: [1]
Predicted Probability: [[0.48591294]] Actual Probability Value: [0]
Predicted Probability: [[0.45406646]] Actual Probability Value: [0]
Predicted Probability: [[0.34158047]] Actual Probability Value: [0]
Predicted Probability: [[0.47491354]] Actual Probability Value: [0]
Predicted Probability: [[0.52182046]] Actual Probability Value: [0]
90.0 % Prediction Accuracy
```

Sr no.	Image Name	Probability	Label(0 or 1)	Human or Non-Human
1	crop_000010b.bmp	0.525200	1	Human
2	crop001008b.bmp	0.677469	1	Human
3	crop001028a.bmp	0.539374	1	Human
4	crop001045b.bmp	0.598200	1	Human
5	crop001047b.bmp	0.607945	1	Human
6	00000053a_cut.bmp	0.485912	0	Non
7	00000062a_cut.bmp	0.454066	0	Non
8	00000093a_cut.bmp	0.341580	0	Non
9	no_person__no_bike_213_cut.bmp	0.474913	0	Non
10	no_person__no_bike_247_cut.bmp	0.521820	0	Non

NEURONS IN THE HIDDEN LAYER: 500

```
Epoch Count: 96 Average Error:  0.00264317937640224
Epoch Count: 97 Average Error:  0.002587587640787036
Epoch Count: 98 Average Error:  0.002533962124107029 6
Epoch Count: 99 Average Error:  0.0024819896494077745
Predicted Probability: [[0.52732442]] Actual Probability Value: [1]
Predicted Probability: [[0.63696358]] Actual Probability Value: [1]
Predicted Probability: [[0.56852544]] Actual Probability Value: [1]
Predicted Probability: [[0.59832571]] Actual Probability Value: [1]
Predicted Probability: [[0.54005113]] Actual Probability Value: [1]
Predicted Probability: [[0.52651848]] Actual Probability Value: [0]
Predicted Probability: [[0.42386218]] Actual Probability Value: [0]
Predicted Probability: [[0.31120401]] Actual Probability Value: [0]
Predicted Probability: [[0.4984429]] Actual Probability Value: [0]
Predicted Probability: [[0.50797678]] Actual Probability Value: [0]
80.0 % Prediction Accuracy
```

Sr no.	Image Name	Probability	Label(0 or 1)	Human or Non-Human
1	crop_000010b.bmp	0. 527324	1	Human
2	crop001008b.bmp	0.636963	1	Human
3	crop001028a.bmp	0.568525	1	Human
4	crop001045b.bmp	0.598325	1	Human
5	crop001047b.bmp	0.526518	1	Human
6	00000053a_cut.bmp	0.423862	0	Non
7	00000062a_cut.bmp	0.311204	0	Non
8	00000093a_cut.bmp	0.498844	0	Non
9	no_person__no_bike_213_cut.bmp	0.498442	0	Non
10	no_person__no_bike_247_cut.bmp	0.507976	0	Non

NEURONS IN THE HIDDEN LAYER: 1000

```
Epoch Count: 94 Average Error:  0.0025054554343919935
Epoch Count: 95 Average Error:  0.0024552317020875153
Epoch Count: 96 Average Error:  0.0024069993700766715
Epoch Count: 97 Average Error:  0.0023603145786003804
Epoch Count: 98 Average Error:  0.00231514219858027
Epoch Count: 99 Average Error:  0.0022714946418210978
Predicted Probability: [[0.52550838]] Actual Probability Value: [1]
Predicted Probability: [[0.67252383]] Actual Probability Value: [1]
Predicted Probability: [[0.5124421]] Actual Probability Value: [1]
Predicted Probability: [[0.56536176]] Actual Probability Value: [1]
Predicted Probability: [[0.53311671]] Actual Probability Value: [1]
Predicted Probability: [[0.55430926]] Actual Probability Value: [0]
Predicted Probability: [[0.38829187]] Actual Probability Value: [0]
Predicted Probability: [[0.31259869]] Actual Probability Value: [0]
Predicted Probability: [[0.478354]] Actual Probability Value: [0]
Predicted Probability: [[0.49395163]] Actual Probability Value: [0]
90.0 % Prediction Accuracy

C:\Users\Mohit\PycharmProjects\HumanDetection>
```

Sr no.	Image Name	Probability	Label(0 or 1)	Human or Non-Human
1	crop_000010b.bmp	0. 525508	1	Human
2	crop001008b.bmp	0.672523	1	Human
3	crop001028a.bmp	0.512442	1	Human
4	crop001045b.bmp	0.565361	1	Human
5	crop001047b.bmp	0.533116	1	Human
6	00000053a_cut.bmp	0.554309	0	Non
7	00000062a_cut.bmp	0.388291	0	Non
8	00000093a_cut.bmp	0.312598	0	Non
9	no_person__no_bike_213_cut.bmp	0.478354	0	Non
10	no_person__no_bike_247_cut.bmp	0.493951	0	Non

SOURCE CODE:

```
# Name: Mohit Suresh Patel
# Netid: msp552
# N#: N17217391
# Computer Vision and Scene Analysis Final Project 2
# Human Detection whether the given image contains Human or not and
generates the output 1 for Human and 0 for Non-Human

import math
import numpy as np
from PIL import Image
from sys import argv
import cv2
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow, show, subplot, figure, gray, title,
axis, savefig, imread
import sys
import os
import glob


# Class HistogramOfGradients is used for getting HOG descriptor
class HistogramOfGradients:

    # function findImages is used to get the images from the folder
    def findImages(self, dataFile, delimeter):
        List_of_path = []
        Output_Data = []

        for dataFolder in dataFile.keys():
            for directoryName, subDirectory, fileL in os.walk(dataFolder):
                for imageFile in fileL:
                    imageP = dataFolder + delimeter + imageFile
                    List_of_path.append(imageP)

                    Output_Data.append([dataFile[dataFolder]])

        return List_of_path, Output_Data

    # this function converts the given RGB image in to a Grayscale Image
```

```
    def RGBtoGray(self, image):
        return np.round(np.dot(image[..., :3], [0.299, 0.587, 0.114]))

    # the function convolveP is used for convolution of the image
    def convolveP(self, image, kernel):

        row_size = len(image)
        column_size = len(image[0])
        # initializing a matrix with float(0)
        image_return = np.zeros((row_size, column_size), dtype=np.float)
        for a in range(0, row_size - 2):
            for b in range(0, column_size - 2):
                # multiplication of two matrices for convolution
                image_return[a + 1][b + 1] = (np.sum(image[a: a + 3, b: b + 3]
* kernel)) / 3

        return image_return

    # function Gradient_using_prewitt is used for generating the horizontal
and vertical gradients of the image which are used for the further
processing
    def Gradient_using_prewitt(self, image):

        row_size = len(image)
        column_size = len(image[0])

        op1 = np.array(([-1, 0, 1],
                        [-1, 0, 1],
                        [-1, 0, 1]))  # Prewitt's Operator for horizontal gradient

        op2 = np.array(([1, 1, 1],
                        [0, 0, 0],
                        [-1, -1, -1]))  # Prewitt's Operator for horizontal gradient
        Gx = self.convolveP(image, op1)
        # Gx = Gx/3
        Gy = self.convolveP(image, op2)
        # Gy = Gy/3
        gradient = np.zeros((row_size, column_size), dtype=np.float)
        grad_angle = np.zeros((row_size, column_size), dtype=np.float)

        for m in range(row_size):
            for n in range(column_size):
                gradient[m][n] = (np.sqrt(np.square(Gx[m][n]) +
```

```
np.square(Gy[m][n]))) / np.sqrt(2)
                # computing gradient for every pixel value and normalizing it
                grad_angle[m][n] = 0
                if (Gx[m][n] == 0):
                    if (Gy[m][n] == 0):
                        grad_angle[m][n] = 0
                    #  grad_angle[m][n] = 0
                    else:
                        if (Gy[m][n] > 0):
                            grad_angle[m][n] = 90
                        else:
                            grad_angle[m][n] = -90

                else:
                    grad_angle[m][n] = math.degrees(np.arctan(Gy[m][n] /
Gx[m][n]))
                    # computing gradient angle for each pixel

                if (grad_angle[m][n] < 0):
                    grad_angle[m][n] = grad_angle[m][n] + 180

        return gradient, grad_angle

    # histogram function is used for returning the feature vector which is
used further in the neural networks for the classification of images
    def histogram(self, grad_angle, gradient):
        row_size = len(gradient)
        column_size = len(gradient[0])

        Feature_Vector = []
        cell_size = (8, 8)  # defining the cell size
        block_size = (16, 16)  # defining the block size
        step_size = (8, 8)  # defining the bin size
        grad_angle = ((grad_angle) / 20.0)  # calculating the the

        i_cell_size = cell_size[0]  # cell size assigned for row
        j_cell_size = cell_size[1]  # cell size assigned for columm

        i_cell_count = row_size // cell_size[0]  # calcualting the cell count
        j_cell_count = column_size // cell_size[1]

        i_cells_per_block, j_cells_per_block = np.array(block_size) /
np.array(
```

```
        cell_size)  # calculating the number of cells per block

    i_cells_per_step, j_cells_per_step = np.array(step_size) /
np.array(step_size)

    i_block_count = int(i_cell_count - i_cells_per_block) /
i_cells_per_step + 1
    j_block_count = int(j_cell_count - j_cells_per_block) /
j_cells_per_step + 1

    Histogram_of_bins = np.zeros((i_cell_count, j_cell_count, 9))

    for I in range(row_size):
      for J in range(column_size):
        c_angle = grad_angle[I][J]
        c_mag = gradient[I][J]

        l_bin = np.floor(c_angle)
        h_bin = np.ceil(c_angle)

        l_dist = c_angle - l_bin
        h_dist = c_angle - h_bin

        l_val = c_mag * h_dist
        h_val = c_mag * l_dist

        Histogram_of_bins[int(I // i_cell_size)][int(J /
j_cell_size)][int(l_bin)] += l_val
        Histogram_of_bins[int(I // i_cell_size)][int(J /
j_cell_size)][int(h_bin % 9)] += h_val

    Histogram_of_bins = Histogram_of_bins.astype(np.float)

    for iWc in range(0, int(i_block_count), int(i_cells_per_step)):
      for jWc in range(0, int(j_block_count), int(j_cells_per_step)):

        c_block = Histogram_of_bins[iWc: iWc +
int(i_cells_per_block),
                     jWc: jWc + int(j_cells_per_block)].reshape(-1)
        c_block_l2 = sum(c_block ** 2) ** 0.5

        if c_block_l2 != 0:
          c_block /= c_block_l2
```

```
            Feature_Vector.append(c_block)

        Feature_Vector = np.array(Feature_Vector).reshape(-1, 1)

        return Feature_Vector

    # HOG_descriptor function is used for returning the features of the
image
    def HOG_descriptor(self, im_path):

        features_of_image = []
        for images in im_path:
            inputImage = self.Get_the_Images(images)
            if ('png' in sys.argv[1]):
                inputImage *= 255

            grayscale = self.RGBtoGray(inputImage)
            gsResult = Image.fromarray(grayscale)

            # computing the gradient and the angle and using Prewitt's
Operator
            gradient, angle = self.Gradient_using_prewitt(grayscale)

            gradientResult = Image.fromarray(gradient)

            cellRes = self.histogram(angle, gradient)

            features_of_image.append(cellRes)
        return features_of_image

    # function Get_the_Images used for getting the images from the
folders which are specified in the main function of this file
    def Get_the_Images(self, im_path):
        return np.array(plt.imread(im_path))


# class NeuralNetwork is used for performing the other half of the project
which includes training the network and classifing the images
class NeuralNetwork:

    def __init__(self, arch=(7524, 1000, 1), epochs=100,
learning_rate=0.01):
```

```
        # arch is the array containing the size of the feature vector, number of neurons in the hidden layer and output
        # epoch is the counts for which our training should be performed
        # learning rate is the alpha which specifies the rate at which neural network should learn and train it accordingly
        self.arch = arch

        # W1 used between input layer and hidden layer
        self.W1 = np.random.randn(arch[1], arch[0]) * 0.01

        # W2 used between hidden layer and output layer
        self.W2 = np.random.randn(arch[2], arch[1]) * 0.01

        # B1 is the bias for input and hidden layer
        self.B1 = np.zeros((arch[1], 1))

        # B2 is the bias for hidden and output layer
        self.B2 = np.zeros((arch[2], 1))

        self.layer1 = self.layer2 = None
        self.d_W1 = self.d_W2 = None
        self.epochs = epochs
        self.learning_rate = learning_rate

    # function feed_forward is used for performing feed forward process of neural networks
    def feed_forward(self, training_data):
        a1 = self.W1.dot(training_data) + self.B1
        self.layer1 = self.ReLU(a1)
        self.layer2 = self.sigmoid(self.W2.dot(self.layer1) + self.B2)

    # function error is used for returning the error from the output of the output layer
    def error(self, actual_output):
        return 0.5 * np.square(self.layer2 - actual_output).sum()

    # function back_propagation is used for performing back propagation process of neural networks
    def back_propagation(self, training_data, actual_output):
        diff = self.layer2 - actual_output
        z2 = diff * self.sigmoid_derivative(self.layer2)
        self.d_W2 = np.dot(z2, self.layer1.T)
```

```
        z1 = np.dot(self.W2.T, z2) * self.RELU_derivative(self.layer1)
        self.d_W1 = np.dot(z1, training_data.T)

        self.d_B2 = np.sum(z2, axis=1, keepdims=True)
        self.d_B1 = np.sum(z1, axis=1, keepdims=True)

    # function update is used for updating the weights after getting the output
    def update(self):
        self.W1 = self.W1 - self.learning_rate * self.d_W1
        self.B1 = self.B1 - self.learning_rate * self.d_B1

        self.W2 = self.W2 - self.learning_rate * self.d_W2
        self.B2 = self.B2 - self.learning_rate * self.d_B2

    # function train_Neural_Network is used for training the network in order to reduce the error which is returned from the error function
    def train_Neural_Network(self, trainImages, training_data_in, training_data_out):
        trainLen = len(training_data_in)

        for epoch in range(self.epochs):
            ep_error = 0.0
            for data_count, train_data in enumerate(training_data_in):
                self.feed_forward(train_data)
                error = self.error(training_data_out[data_count])
                ep_error += error
                self.back_propagation(train_data, training_data_out[data_count])
                self.update()

            print("Epoch Count: " + str(epoch), "Average Error: ", ep_error / trainLen)

    # function test_Neural_Network is used for testing the images whether it contains human or not on the basis of the network which is generated so far
    def test_Neural_Network(self, testImages, testing_data_in, testing_data_out):
        misclassify = 0

        positiveList = []
        negativeList = []
```

```
        for data_count, test_data in enumerate(testing_data_in):
            self.feed_forward(test_data)
            print("Predicted Probability: " + str(self.layer2),
                  "Actual Probability Value: " +
str(testing_data_out[data_count]))

            cPrediction = np.round(self.layer2.sum())

            if cPrediction:
                positiveList.append([testImages[data_count],
str(self.layer2.sum())])
            else:
                negativeList.append([testImages[data_count],
str(self.layer2.sum())])

            misclassify += (float(cPrediction - testing_data_out[data_count])
== 0)

        print(str(float(misclassify) / float(len(testing_data_out)) * 100) + " %
Prediction Accuracy")

    # function sigmoid is used between the hidden and output layer during
the feed forward process
    def sigmoid(self, t):
        return 1 / (1 + np.exp(-t))

    # function sigmoid_derivative is used between the hidden and output
layer during the back propagation process
    def sigmoid_derivative(self, p):
        return p * (1 - p)

    # function RELU is used between the input and hidden layer during
the feed forward process
    def ReLU(self, t):
        return np.maximum(t, 0)

    # function RELU_derivative is used between the input and hidden
layer during the back propagation process
    def RELU_derivative(self, t):
        return 1 * (t > 0)
```

```
if __name__ == "__main__":
    if len(sys.argv) >= 2:
        datafilepath = sys.argv[1]
    else:
        datafilepath = raw_input("data filepath was not provided, please
enter a filepath for data now:")

    TRAIN_POS_FOLDER = 'train_positive'
    TRAIN_NEG_FOLDER = 'train_negative'
    TEST_POS_FOLDER = 'test_positive'
    TEST_NEG_FOLDER = 'test_negative'
    FILE_PATH_DELIMITER = '/'

    TRAIN_POS_PATH = datafilepath + FILE_PATH_DELIMITER +
TRAIN_POS_FOLDER
    TRAIN_NEG_PATH = datafilepath + FILE_PATH_DELIMITER +
TRAIN_NEG_FOLDER
    TEST_POS_PATH = datafilepath + FILE_PATH_DELIMITER +
TEST_POS_FOLDER
    TEST_NEG_PATH = datafilepath + FILE_PATH_DELIMITER +
TEST_NEG_FOLDER

    # giving the labels 1-Human and 0-Non-Human Images
    train_data_dict = {TRAIN_POS_PATH: 1, TRAIN_NEG_PATH: 0}
    test_data_dict = {TEST_POS_PATH: 1, TEST_NEG_PATH: 0}

    # creating an object of class HistogramOfGradients
    h = HistogramOfGradients()

    train_image_path_list, train_data_output = \
        h.findImages(train_data_dict, FILE_PATH_DELIMITER)
    test_image_path_list, test_data_output = \
        h.findImages(test_data_dict, FILE_PATH_DELIMITER)

    train_data_input = np.array(h.HOG_descriptor(train_image_path_list))
    test_data_input = np.array(h.HOG_descriptor(test_image_path_list))

    # creating an object of class NeuralNetwork for performing training the
network and testing of the images
    OBJECT_OF_NN = NeuralNetwork()
    OBJECT_OF_NN.train_Neural_Network(train_image_path_list,
train_data_input, train_data_output)
```

```
    OBJECT_OF_NN.test_Neural_Network(test_image_path_list,
test_data_input, test_data_output)
```