- Source Code: HumanClassifier.py

HOG feature files

./ImageData/Test_images_Neg_res/00000003a_cut_hog.txt
./ImageData/Test_images_Neg_res/00000118a_cut_hog.txt
./ImageData/Test_images_Neg_res/00000090a_cut_hog.txt
./ImageData/Test_images_Neg_res/no_person__no_bike_264_cut_hog.txt
./ImageData/Test_images_Neg_res/no_person__no_bike_258_Cut_hog.txt
./ImageData/Training_images_Pos_res/crop001008b_hog.txt
./ImageData/Training_images_Pos_res/crop001028a_hog.txt
./ImageData/Training_images_Pos_res/crop001030c_hog.txt
./ImageData/Training_images_Pos_res/crop001063b_hog.txt
./ImageData/Training_images_Pos_res/crop001045b_hog.txt
./ImageData/Training_images_Pos_res/crop001047b_hog.txt
./ImageData/Training_images_Pos_res/person_and_bike_026a_hog.txt
./ImageData/Training_images_Pos_res/crop001672b_hog.txt
./ImageData/Training_images_Pos_res/crop_000010b_hog.txt
./ImageData/Training_images_Pos_res/crop001275b_hog.txt
./ImageData/Training_images_Neg_res/no_person__no_bike_213_cut_hog.txt
./ImageData/Training_images_Neg_res/01-03e_cut_hog.txt
./ImageData/Training_images_Neg_res/no_person__no_bike_259_cut_hog.txt
./ImageData/Training_images_Neg_res/00000053a_cut_hog.txt
./ImageData/Training_images_Neg_res/no_person__no_bike_219_cut_hog.txt
./ImageData/Training_images_Neg_res/00000091a_cut_hog.txt
./ImageData/Training_images_Neg_res/00000093a_cut_hog.txt
./ImageData/Training_images_Neg_res/no_person__no_bike_247_cut_hog.txt
./ImageData/Training_images_Neg_res/00000062a_cut_hog.txt
./ImageData/Training_images_Neg_res/00000057a_cut_hog.txt
./ImageData/Test_images_Pos_res/crop001500b_hog.txt
./ImageData/Test_images_Pos_res/crop001278a_hog.txt
./ImageData/Test_images_Pos_res/crop001070a_hog.txt
./ImageData/Test_images_Pos_res/person_and_bike_151a_hog.txt
./ImageData/Test_images_Pos_res/crop001034b_hog.txt

LBP feature files

./ImageData/Test_images_Neg_res/00000118a_cut_lbp.txt
./ImageData/Test_images_Neg_res/no_person__no_bike_264_cut_lbp.txt
./ImageData/Test_images_Neg_res/00000003a_cut_lbp.txt
./ImageData/Test_images_Neg_res/no_person__no_bike_258_Cut_lbp.txt
./ImageData/Test_images_Neg_res/00000090a_cut_lbp.txt
./ImageData/Training_images_Pos_res/crop_000010b_lbp.txt
./ImageData/Training_images_Pos_res/crop001672b_lbp.txt
./ImageData/Training_images_Pos_res/crop001030c_lbp.txt
./ImageData/Training_images_Pos_res/crop001047b_lbp.txt
./ImageData/Training_images_Pos_res/crop001275b_lbp.txt
./ImageData/Training_images_Pos_res/crop001028a_lbp.txt
./ImageData/Training_images_Pos_res/person_and_bike_026a_lbp.txt

./ImageData/Training_images_Pos_res/crop001045b_lbp.txt
./ImageData/Training_images_Pos_res/crop001063b_lbp.txt
./ImageData/Training_images_Pos_res/crop001008b_lbp.txt
./ImageData/Training_images_Neg_res/no_person__no_bike_213_cut_lbp.txt
./ImageData/Training_images_Neg_res/00000091a_cut_lbp.txt
./ImageData/Training_images_Neg_res/01-03e_cut_lbp.txt
./ImageData/Training_images_Neg_res/00000062a_cut_lbp.txt
./ImageData/Training_images_Neg_res/00000093a_cut_lbp.txt
./ImageData/Training_images_Neg_res/00000057a_cut_lbp.txt
./ImageData/Training_images_Neg_res/no_person__no_bike_219_cut_lbp.txt
./ImageData/Training_images_Neg_res/no_person__no_bike_247_cut_lbp.txt
./ImageData/Training_images_Neg_res/no_person__no_bike_259_cut_lbp.txt
./ImageData/Training_images_Neg_res/00000053a_cut_lbp.txt
./ImageData/Test_images_Pos_res/crop001070a_lbp.txt
./ImageData/Test_images_Pos_res/crop001034b_lbp.txt
./ImageData/Test_images_Pos_res/crop001278a_lbp.txt
./ImageData/Test_images_Pos_res/person_and_bike_151a_lbp.txt
./ImageData/Test_images_Pos_res/crop001500b_lbp.txt

- Instruction on how to run your program, and instruction on how to compile your program if your program requires compilation.

Step 1: install dependencies numpy, open-cv
Step 2: extract submission folder
Step 3: move into directory containing image folder **ImageData** and python file **HumanClassifier.py**
Step 4: execute python file with command
python HumanClassifier.py >> output.txt
Step 5: check neural network training error output and predictions in output.txt
Else:
Execute **python HumanClassifier.py** to view output on the command line`

- Method you used to initialize the weight values of your perceptron

w = np.random.randn(total_hidden,hidden)
w = np.multiply(w1,math.sqrt(2/int(total_hidden+hidden)))

- Criteria you used to stop training

If abs(prev_err - ep_err) < 0.0001 then stop training

- The number of iterations (or epochs) required to train your perceptron. Report for each of the four experiments: hidden layer sizes of 200 and 400 -- HOG only and combined HOG-LBP.

HOG (200)	66
HOG (400)	62
HOG-LBP (200)	63
HOG-LBP(400)	67

- For hidden layer sizes 200 and 400, create separate tables (see below) that contain the output values of the output neuron and the classification results (human, borderline or no-human) for HOG feature only and for the combined HOG-LBP feature. Use the rules above (in Two-layer perceptron section) for classification. Report results for all 10 test images in the table. Also, compute and report the average error for the 10 test images. The error for a test sample is computed as . o

HOG-HOGLBP for 200 neurons

ImageName	CorrectClass	HOG only		HOG-LBP	
		Output	Classification	Output	Classification
crop001034b.bmp	Human	0.3735689745	no-human		no-human
crop001278a.bmp	Human	0.9387916074	human		human
person_and_bike_151a.bmp	Human	0.9084456793	human		human
crop001070a.bmp	Human	0.7994003004	human		human
crop001500b.bmp	Human	0.5871574337	borderline		borderline
00000003a_cut.bmp	No-human	0.1049343418	no-human		no-human
no_person__no_bike_258_Cut.bmp	No-human	0.5701941437	borderline		human
00000090a_cut.bmp	No-human	0.01799956511	no-human		no-human
no_person__no_bike_264_cut.bmp	No-human	0.1210229736	no-human		no-human
00000118a_cut.bmp	No-human	0.1241838004	no-human		no-human

HOG 400

ImageData/Test_images_Neg/no_person__no_bike_264_cut.bmp = class (no-human) with predicted value as :: 0.Output
ImageData/Test_images_Neg/00000118a_cut.bmp = class (no-human) with predicted value as :: 0.15296791102655427

ImageData/Test_images_Neg/00000003a_cut.bmp = class (no-human) with predicted value as :: 0.14833132963023984
ImageData/Test_images_Neg/00000090a_cut.bmp = class (no-human) with predicted value as :: 0.052532847349917734
ImageData/Test_images_Neg/no_person__no_bike_258_Cut.bmp = class (human) with predicted value as :: 0.6733152887288638
ImageData/Test_images_Pos/crop001034b.bmp = class (no-human) with predicted value as :: 0.3243464760862844
ImageData/Test_images_Pos/person_and_bike_151a.bmp = class (human) with predicted value as :: 0.9235645831642586
ImageData/Test_images_Pos/crop001278a.bmp = class (human) with predicted value as :: 0.9645462573213118
ImageData/Test_images_Pos/crop001500b.bmp = class (human) with predicted value as :: 0.624489593741132
ImageData/Test_images_Pos/crop001070a.bmp = class (human) with predicted value as :: 0.8403137031468967

HOG-LBP 200

ImageData/Test_images_Neg/no_person__no_bike_264_cut.bmp = class (no-human) with predicted value as :: 0.3229632164270737
ImageData/Test_images_Neg/00000118a_cut.bmp = class (no-human) with predicted value as :: 0.11275191701850437
ImageData/Test_images_Neg/00000003a_cut.bmp = class (no-human) with predicted value as :: 0.15079312231231573
ImageData/Test_images_Neg/00000090a_cut.bmp = class (no-human) with predicted value as :: 0.02002604001521951
ImageData/Test_images_Neg/no_person__no_bike_258_Cut.bmp = class (human) with predicted value as :: 0.6962562333986343
ImageData/Test_images_Pos/crop001034b.bmp = class (no-human) with predicted value as :: 0.3041314029710677
ImageData/Test_images_Pos/person_and_bike_151a.bmp = class (human) with predicted value as :: 0.9620384569010505
ImageData/Test_images_Pos/crop001278a.bmp = class (human) with predicted value as :: 0.954780444306016
ImageData/Test_images_Pos/crop001500b.bmp = class (borderline) with predicted value as :: 0.5301325874014972
ImageData/Test_images_Pos/crop001070a.bmp = class (human) with predicted value as :: 0.8989272527219223

HOG-LBP 400

ImageData/Test_images_Neg/no_person__no_bike_264_cut.bmp = class (no-human) with predicted value as :: 0.18883294951093296
ImageData/Test_images_Neg/00000118a_cut.bmp = class (no-human) with predicted value as

:: 0.14593827575969928
ImageData/Test_images_Neg/00000003a_cut.bmp = class (no-human) with predicted value as :: 0.1336468544944816
ImageData/Test_images_Neg/00000090a_cut.bmp = class (no-human) with predicted value as :: 0.021509678911560273
ImageData/Test_images_Neg/no_person__no_bike_258_Cut.bmp = class (human) with predicted value as :: 0.6246223539818803
ImageData/Test_images_Pos/crop001034b.bmp = class (no-human) with predicted value as :: 0.33236572849529405
ImageData/Test_images_Pos/person_and_bike_151a.bmp = class (human) with predicted value as :: 0.921072216823859
ImageData/Test_images_Pos/crop001278a.bmp = class (human) with predicted value as :: 0.949909677089038
ImageData/Test_images_Pos/crop001500b.bmp = class (human) with predicted value as :: 0.6017396673228734
ImageData/Test_images_Pos/crop001070a.bmp = class (human) with predicted value as :: 0.8164627664446198

- The source code of your program (Copy-and-paste from source code file. This is in addition to the source code file that you need to hand in.)

```
import numpy as np
import sys
import cv2
import math
import os
import random

def getImagePathsWithOutput(path_dir_output):
    image_paths = []
    output = []
    for folder in path_dir_output.keys():
        for dirt, subdirt, fileList in os.walk(folder):
            for file in fileList:
                image_paths.append(folder + file)
                output.append(path_dir_output[folder])
    return image_paths, output

class HOGFeature:

    def rgb2gray(self, img):
        '''
        function to convert image from color image to grayscale image
```

```
    img = the image matrix
    '''
    gray = np.zeros((img.shape[0],img.shape[1]))
    for i in range(0,img.shape[0]):
        for j in range(0,img.shape[1]):
            bgr = img[i,j]
            b = bgr[0]
            g = bgr[1]
            r = bgr[2]
            gray[i,j] = round(0.299*r+0.587*g+0.114*b)
    return gray

def apply_sobel(self, img):
    '''
    Function to compute Normalized Gradient Magnitude and Gradient Angle
    '''
    #Sobel Filter Mask to calculate Horizontal(x) gradient
    sobel_x = (1/4)*np.array([(-1,0,1),
                (-2,0,2),
                (-1,0,1)])
    #Sobel Filter Mask to calculate Vertical(y) gradient
    sobel_y = (1/4)*np.array([(1,2,1),
                (0,0,0),
                (-1,-2,-1)])

    #initialize matrices to store the value of horizontal and vertical gradient,
    #normalized horizontal and vertical gradient, normalized gradient magnitude
    #and gradient angle
    gradient_magnitude = np.zeros(shape=img.shape)
    gradient_angle = np.zeros(shape=img.shape)
    #find the gradient values by perfoeming convolution
    for row in range(0,img.shape[0]-2):
        for col in range(0,img[row].size-2):
            #calculate Value at current pixel (row,col)
            #after applying sobel operator
            gx, gy = 0, 0
            for i in range (0,3):
                for j in range (0,3):
                    gx = gx + img[row+i][col+j] * sobel_x[i][j]
                    gy = gy + img[row+i][col+j] * sobel_y[i][j]
            #normalize gradient magnitude by dividing by sqrt(2)
            gradient_magnitude[row+1][col+1]=((gx**2+gy**2)**(0.5))/(1.4142)
            #calculate gradient angle based on sobel horizontal gradient and vertical gradient
            angle = 0
```

```
            if(gx == 0):
                if( gy > 0):
                    angle = 90
                else:
                    angle = -90
            else:
                angle = math.degrees(math.atan(gy/gx))
            if (angle < 0):
                angle = angle + 180
            gradient_angle[row+1,col+1]  = angle
        return [gradient_magnitude, gradient_angle]

    def get_cell_histogram(self, gradient_magnitude, gradient_angle):
        '''
        This function is to get histogram for each 8x8 cell
        gradient_magnitude = gradient magnitude for each pixel
        gradient_angle = gradient angle for each pixel
        '''
        cell_shape = gradient_magnitude.shape
        #initialize the number of cell rows and cell columns
        cell_rows = round(cell_shape[0]/8)
        cell_cols = round(cell_shape[1]/8)
        histogram_cell = np.zeros((cell_rows,cell_cols,9))
        for r in range (0,cell_rows-1):
            for c in range (0,cell_cols-1):
                for row in range (r*8,r*8+8):
                    for col in range (c*8,c*8+8):
                        angle = gradient_angle[row][col]
                        grad_mag = gradient_magnitude[row][col]
                        if angle%20 == 0:
                            if angle == 180:
                                histogram_cell[r][c][0] += grad_mag
                                continue
                            bin_no = int(angle/20)
                            histogram_cell[r][c][bin_no] += grad_mag
                            continue
                        bin_no_l = int(angle/20)
                        #calculate the vote for left and right bins.
                        if bin_no_l < 8:
                            bin_no_r = bin_no_l + 1
                            histogram_cell[r][c][bin_no_r] += grad_mag*((angle - (bin_no_l * 20))/20)
                            histogram_cell[r][c][bin_no_l] += grad_mag*(((bin_no_r * 20) - angle)/20)
                        else:
                            bin_no_r = 0
```

```
                    histogram_cell[r][c][bin_no_r] += grad_mag*((angle - 160)/20)
                    histogram_cell[r][c][bin_no_l] += grad_mag*((180 - angle)/20)
        squared_histogram_cell = np.square(histogram_cell)
        return [histogram_cell, squared_histogram_cell]


    def get_hog_descriptor(self, histogram_cell, histogram_cell_squared):
        cell_histogram_shape = histogram_cell.shape
        descriptor = np.array([])
        for row in range(0,cell_histogram_shape[0]-1):
            for col in range(0,cell_histogram_shape[1]-1):
                block = np.array([])
                block_squared = np.array([])
                block = np.append(block,histogram_cell[row,col])
                block = np.append(block,histogram_cell[row,col+1])
                block = np.append(block,histogram_cell[row+1,col])
                block = np.append(block,histogram_cell[row+1,col+1])
                block_squared = np.append(block_squared,histogram_cell_squared[row,col])
                block_squared = np.append(block_squared,histogram_cell_squared[row,col+1])
                block_squared = np.append(block_squared,histogram_cell_squared[row+1,col])
                block_squared = np.append(block_squared,histogram_cell_squared[row+1,col+1])
                block_squared = np.sum(block_squared)
                if(block_squared>0):
                    #normalize the block descriptor
                    normalized = np.sqrt(block_squared)
                    block = (1/normalized)*block
                descriptor = np.append(descriptor, block)
        return descriptor


    def get_LBP_descriptor(self, magnitude):
        '''
        This function is to get histogram for each 8x8 cell
        gradient_magnitude = gradient magnitude for each pixel
        gradient_angle = gradient angle for each pixel
        '''
        pattern = [(-1, -1),(-1, 0),(-1, 1),(0, -1),(0, 1),(1, -1),(1, 0),(1, 1)]
        uniform_patterns = np.array([0, 1, 2, 3, 4, 6, 7, 8, 12, 14, 15, 16, 24, 28, 30, 31, 32, 48, 56,
        60, 62, 63, 64, 96, 112, 120, 124, 126, 127, 128, 129, 131, 135, 143, 159, 191, 192, 193,
195, 199,
        207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254, 255])
        non_uniform_patterns = np.array(list(set(np.arange(256))-set(uniform_patterns)))
        cell_shape = magnitude.shape
        #initialize the number of cell rows and cell columns
```

```
        cell_rows = round(cell_shape[0]/16)
        cell_cols = round(cell_shape[1]/16)
        histogram_block = np.zeros((cell_rows,cell_cols,59))
        lbp_mask = np.zeros(magnitude.shape)
        lbp_descriptor = np.array([])
        for r in range (0,cell_rows):
            for c in range (0,cell_cols):
                histogram_block = np.zeros(256)
                for row in range (r*16,r*16+16):
                    for col in range (c*16,c*16+16):
                        if row == 0 or col == 0 or row == magnitude.shape[0]-1 or col ==
magnitude.shape[1]-1:
                            lbp_mask[row][col] = 5
                            continue
                        bin_val = ''
                        mag = magnitude[row][col]
                        for i,j in pattern:
                            if magnitude[row+i][col+j] >= mag:
                                bin_val += '1'
                            else:
                                bin_val += '0'
                        histogram_block[int(bin_val,2)] += 1
                normalized_histogram = histogram_block/256
                bin_59 = sum(normalized_histogram[non_uniform_patterns])
                bin_1_58 = normalized_histogram[uniform_patterns]
                bin_1_59 = np.append(bin_1_58, bin_59)
                lbp_descriptor = np.append(lbp_descriptor,bin_1_59)
        return lbp_descriptor


    def saveImageAndHog(self, mag, hog, lbp, path):
        """
        function to save the image and hog descriptor
        """
        pathSplit = path.split('/')
        currImage = pathSplit[-1]
        imageName, imageExt = currImage.split('.')
        updatedImage = '.'.join([imageName+'_mag',imageExt])
        imageFolder = '/'.join(pathSplit[:-1])
        imageFolder += "_res"
        if not os.path.exists(imageFolder):
            os.makedirs(imageFolder)
        finalPath = '/'.join([imageFolder,updatedImage])
        cv2.imwrite(finalPath, mag)
```

```
        filename = imageFolder+"/"+imageName+"_hog.txt"
        hog_file = open(filename,'a+')
        for i in hog:
            hog_file.write(str(i[0])+'\n')
        hog_file.close()
        filename = imageFolder+"/"+imageName+"_lbp.txt"
        lbp_file = open(filename,'a+')
        for i in lbp:
            lbp_file.write(str(i[0])+'\n')
        lbp_file.close()

    def HogLbp(self, image_list):
        hog_features = []
        lbp_features = []
        for image in image_list:
            color_img = cv2.imread(image,cv2.IMREAD_COLOR)
            gray_img = self.rgb2gray(color_img)
            gradient_magnitude, gradient_angle = self.apply_sobel(gray_img)
            histogram = self.get_cell_histogram(gradient_magnitude, gradient_angle)
            histogram_cell = histogram[0]
            squared_histogram_cell = histogram[1]
            HOG_descriptor = self.get_hog_descriptor(histogram_cell, squared_histogram_cell)
            HOG_descriptor = HOG_descriptor.reshape(-1,1)
            LBP_descriptor = self.get_LBP_descriptor(gradient_magnitude)
            LBP_descriptor = LBP_descriptor.reshape(-1,1)
            self.saveImageAndHog(gradient_magnitude, HOG_descriptor, LBP_descriptor, image)
            hog_features.append(HOG_descriptor)
            lbp_features.append(LBP_descriptor)
        return hog_features,lbp_features

def ReLU(x):
    """Function to calculate RELU
    """
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i,j]<0:
                x[i,j]=0
    return x

def Sigmoid(x):
    """
    Function to calculate sigmoid
    """
    return 1/(1+np.exp(-x))
```

```
def neural_network(network,out,hidden):
    """
    Function to implament neural network, to train it.
    """
    aplha = 0.1
    # Initializing the learning rate
    total_hidden = network.shape[1]
    total_output = network.shape[2]
    # Initializing and Factoring the weight for hidden layer
    w1 = np.random.randn(total_hidden,hidden)
    w1 = np.multiply(w1,math.sqrt(2/int(total_hidden+hidden)))
    # Initializing and Factoring the weight for output layer
    w2 = np.random.randn(hidden,total_output)
    w2 = np.multiply(w2,math.sqrt(2/int(hidden+total_output)))
    # Bias for hidden and output layer
    b1 = np.random.randn(hidden)
    b1 = np.multiply(b1,math.sqrt(2/int(hidden)))
    b2 = np.random.randn(total_output)
    b2 = np.multiply(b2,math.sqrt(2/int(total_output)))
    epoch = 0
    err_sq = 0
    prev_err = sys.maxsize
    while True:
        # FeedForward and Backpropagation for each epoch of all vectors
        for i in range(network.shape[0]):
            x = network[i,:].reshape([1,-1])
            # Computing values for hidden layer and output layer in feedforward
            layer1_output = ReLU((x.dot(w1)+b1))
            layer2_output = Sigmoid((layer1_output.dot(w2)+b2))
            err = out[i]-layer2_output
            err_sq += 0.5*err*err
            #Doing BackPropagation
            del_output = (-1*err)*(1-layer2_output)*layer2_output
            del_layer2 = layer1_output.T.dot(del_output)
            del_bias_layer2 = np.sum(del_output,axis = 0)
            layer1_output_like = np.zeros_like(layer1_output)
            for k in range(hidden):
                if(layer1_output[0][k]>0):
                    layer1_output_like[0][k] = 1
                else:
                    layer1_output_like[0][k] = 0
            del_hidden = del_output.dot(w2.T)*layer1_output_like
            del_layer1 = x.T.dot(del_hidden)
```

```
            del_bias_layer1 = np.sum(del_hidden,axis=0)

            w2 -= aplha*del_layer2
            b2 -= aplha*del_bias_layer2
            w1 -= aplha*del_layer1
            b1 -= aplha*del_bias_layer1
        ep_err = np.mean(err_sq)/network.shape[0]
        print("Epoch Count: " + str(epoch), "Average Error: ", ep_err)
        if(ep_err < prev_err):
            print("error decreased by ", prev_err-ep_err)
        else:
            if(ep_err > prev_err):
                print("error increased by ", ep_err-prev_err)
            else:
                print("error stayed same")
        #check for the change in error if very less we can stop training
        if(abs(prev_err - ep_err) < 0.0001):
            print("training complete....")
            break
        prev_err = ep_err
        epoch += 1

    return w1,b1,w2,b2

def predict(w,wb,v,vb,output_descriptor):
    # Function to predict values for my neural network
    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

root = 'ImageData/'
train_pos_dir = root + 'Training_images_Pos/'
train_neg_dir = root + 'Training_images_Neg/'
test_pos_dir = root + 'Test_images_Pos/'
test_neg_dir = root + 'Test_images_Neg/'

train_data_dir_with_output = {
train_pos_dir:1,
train_neg_dir:0
```

```
}
test_data_dir_with_output = {
test_neg_dir:0,
test_pos_dir:1
}

training_image_paths, training_output = getImagePathsWithOutput(train_data_dir_with_output)
testing_image_paths, testing_output = getImagePathsWithOutput(test_data_dir_with_output)

h = HOGFeature()

hog_train, lbp_train = h.HogLbp(training_image_paths)

hog_test, lbp_test = h.HogLbp(testing_image_paths)

hog_lbp_train = []
for i,j in zip(hog_train, lbp_train):
    hog_lbp_train.append(np.append(i,j))

hog_lbp_test = []
for i,j in zip(hog_test, lbp_test):
    hog_lbp_test.append(np.append(i,j))

train_data_hog = np.array(hog_train)
test_data_hog = np.array(hog_test)


train_data_hoglbp = np.array(hog_lbp_train).reshape(20,11064,1)
test_data_hoglbp = np.array(hog_lbp_test).reshape(10,11064,1)

print('\nHOG only ')
for hidden in [200,400]:
    print('\n\nHIDDEN LAYER = %d \n\n'%(hidden))
    w1,w1bias,w2,w2bias = neural_network(np.array(hog_train),np.array(training_output),hidden)
    predicted_output = predict(w1,w1bias,w2,w2bias,np.array(hog_test).reshape(10,7524))
    prediction = []
    classes = []
    for predicted in predicted_output:
        if(predicted >=0.5):
            prediction.append(1)
        else:
            prediction.append(0)
        if predicted >= 0.6:
            classes.append('human')
```

```
        elif predicted <= 0.4:
            classes.append('no-human')
        else:
            classes.append('borderline')

    correct=0
    wrong=0
    error = 0
    for i in range(len(prediction)):
        error += abs(testing_output[i] - predicted_output[i])
        if(prediction[i]==testing_output[i]):
            correct+=1
        else:
            wrong+=1
        print(testing_image_paths[i] + ' = class (' + classes[i] + ') with predicted value as  :: ' +
str(predicted_output[i][0][0]))
    print('correct = %d'%(correct))
    print('wrong = %d'%(wrong))
    print('average error = %d'%(error/10))
    print(predicted_output)
    print(testing_output)


print('\nHOG and LBP ')
for hidden in [200,400]:
    print('\n\nHIDDEN LAYER = %d \n\n'%(hidden))
    w1,w1bias,w2,w2bias = neural_network(train_data_hoglbp,np.array(training_output),hidden)
    predicted_output=predict(w1,w1bias,w2,w2bias,test_data_hoglbp.reshape(10,11064))
    prediction = []
    classes = []
    for predicted in predicted_output:
        if(predicted >=0.5):
            prediction.append(1)
        else:
            prediction.append(0)
        if predicted >= 0.6:
            classes.append('human')
        elif predicted <= 0.4:
            classes.append('no-human')
        else:
            classes.append('borderline')

    correct=0
    wrong=0
```

```
    error = 0
    for i in range(len(prediction)):
        error += abs(testing_output[i] - predicted_output[i])
        if(prediction[i]==testing_output[i]):
            correct+=1
        else:
            wrong+=1
        print(testing_image_paths[i] + ' = class (' + classes[i] + ') with predicted value as  :: ' +
str(predicted_output[i][0][0]))
    print('correct = %d'%(correct))
    print('wrong = %d'%(wrong))
    print('average error = %d'%(error/10))
    print(predicted_output)
    print(testing_output)
```

- Normalized gradient magnitude images for the 10 test images (Copy-and-paste from output image files.) o