Project 2: Human Detection
Computer Vision (CS 6643)
Bhushan Manohar Newalkar (bmn258)

a) Source code file names:
   1. create_data.py
   2. HoG.py
   3. neural_net.py
   4. run_NN.py

b) Instructions on how to compile and run the code:
   1. First place above four files in a folder.
   2. Inside this folder create four folders as ‘positive_train’, ‘negative_train’, ‘positive_test’, and ‘negative_test’.
   3. Place all the positive train images in positive train folder and likewise place the other images in the respective folders.
   4. First run the create_data.py file. This file will create two csv files ‘train.csv’ and ‘test.csv’.
   5. Then run the ‘run_NN.py’ file. It will display the accuracy and the predictions as outputs.
   6. Or you can run the CV_Project2.bat batch file which will do steps 4 and 5 for you.

c) Answers:
   1. How did you initialize the weight values of the network?
      → Random weights from normal distribution, 250 hidden neurons, and learning rate of 0.01
   2. How many iterations (or epochs) through the training data did you perform
      → After trying various combination, epoch was decided to be 400.
   3. How did you decide when to stop training?
      → When either number of iterations (epochs) are completed or the difference between successive errors is less than 10^-6
   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?
      → The output neuron uses sigmoid as activation function so if the value is less than 0.5 the image is non-human else it is human.

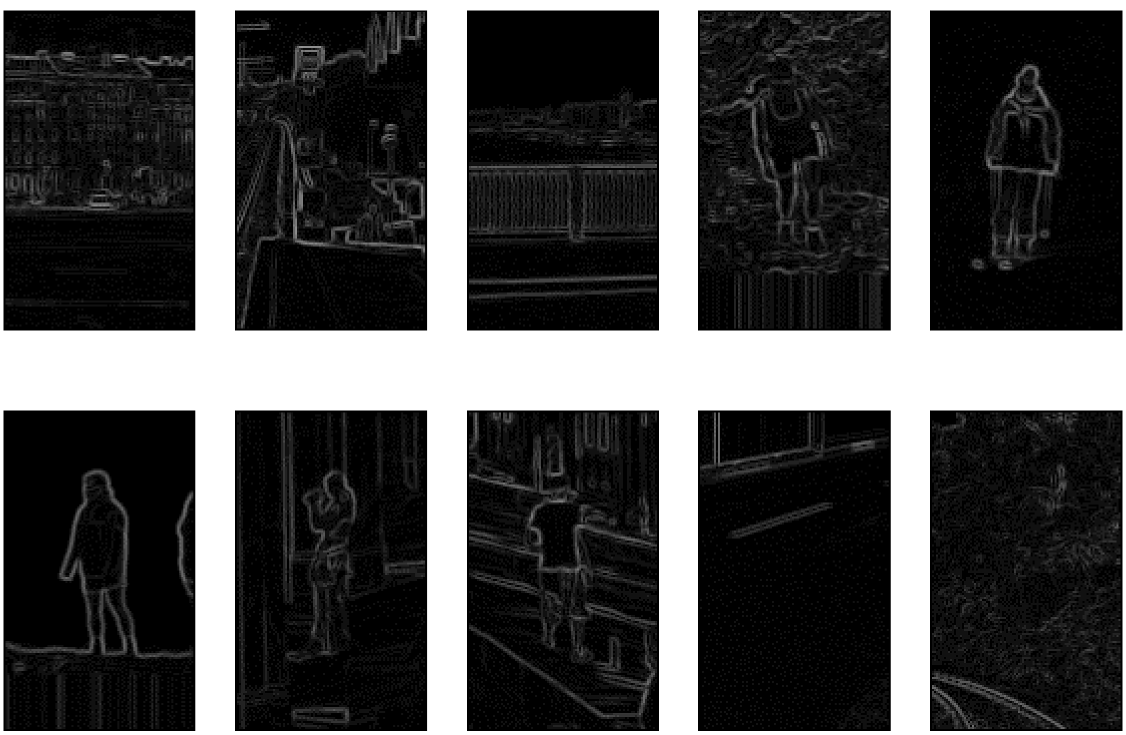
Project 2: Human Detection
Computer Vision (CS 6643)
Bhushan Manohar Newalkar (bmn258)

d) Classification results on test images:

Test Image	Output value	Classification
crop_000010b	0.97051742	1
crop001008b	0.98690437	1
crop001028a	0.78469989	1
crop001045b	0.88945977	1
crop001047b	0.91908398	1
00000053a_cut	0.41748846	0
00000062a_cut	0.22092658	0
00000093a_cut	0.03816118	0
no_person__no_bike_213_cut	0.47318405	0
no_person__no_bike_247_cut	0.08421976	0

e) Comments:
Training the neural network may take some time depending upon the size of data and the terminating conditions.
Above is the best accuracy I got while trying different hyper parameters with random initial weights.

f) Normalized test images:



Project 2: Human Detection
Computer Vision (CS 6643)
Bhushan Manohar Newalkar (bmn258)

g) Code:

1. HoG.py:

```
# importing required libraries
import numpy as np
from skimage.io import imread
from PIL import Image
import math
import sys

class HOG:
  def __init__(self, image_name):
    self.image_name = image_name

  # function to convert color image to greyscale
  def greyscale_operation(self, ip_img):
    op_img = np.zeros((ip_img.shape[0], ip_img.shape[1]))
    for i in range(ip_img.shape[0]):
      for j in range(ip_img.shape[1]):
        op_img[i][j] = np.round_(0.299*ip_img[i][j][0] + 0.587*ip_img[i][j][1] +
0.114*ip_img[i][j][2])
    return op_img

  # function for implementing gradient operation using prewitt's edge detector
  def gradient_operation(self, ip_img):
    # prewitt's vertical and horizontal kernels
    Gx_kernel = ([[-1, 0, 1]] * 3)
    Gy_kernel = ([1, 1, 1], [0, 0, 0], [-1, -1, -1])

    Gx = np.zeros((ip_img.shape[0], ip_img.shape[1]))
    Gy = np.zeros((ip_img.shape[0], ip_img.shape[1]))
    G = np.zeros((ip_img.shape[0], ip_img.shape[1]))
    Theta = np.zeros((ip_img.shape[0], ip_img.shape[1]))

    # calculating Gx and Gy using prewitt's edge detector
    for i in range(ip_img.shape[0]):
      for j in range(ip_img.shape[1]):
        # pixels for which part of the prewitt's mask goes outside of the image border
        if i < 1 or j < 1 or ip_img.shape[0] - i <= 1 or ip_img.shape[1] - j <= 1:
          continue
        else:
          arr1 = ip_img[i - 1:i + 2, j - 1:j + 2]
```

```
                Gx[i][j] = np.sum(np.multiply(arr1, Gx_kernel))/3
                Gy[i][j] = np.sum(np.multiply(arr1, Gy_kernel))/3
                G[i][j] = round(math.sqrt(Gx[i][j]**2 + Gy[i][j]**2)/math.sqrt(2))
                if Gx[i][j] == 0 and Gy[i][j] == 0:
                    Theta[i][j] = 0
                else:
                    theta = np.arctan2(Gy[i][j], Gx[i][j])*180/np.pi
                    if theta < 0:
                        theta += 180
                    if theta >= 170:
                        theta -= 180
                    Theta[i][j] = theta

        return G, Theta

    # HoG implementation
    def hog_operation(self, ip_img, ip_theta):
        # HoG bins table
        bins_table = {0:[-10,10], 20:[10,30], 40:[30,50], 60:[50,70], 80:[70,90], 100:[90,110],
120:[110,130], 140:[130, 150], 160:[150,170]}
        histo_center = np.zeros((ip_img.shape[0], ip_img.shape[1]))

        # First the bin centers for each pixel are identified
        for i in range(ip_theta.shape[0]):
            for j in range(ip_theta.shape[1]):
                for x in bins_table.keys():
                    if bins_table[x][0] <= ip_theta[i][j] < bins_table[x][1]:
                        histo_center[i][j] = x

        # hog histograms for each cell are created
        histo_list = np.zeros((int(ip_img.shape[0]/8), int(ip_img.shape[1]/8), 9))

        a = 0
        k = 0
        for x in range(int(ip_theta.shape[0]/8)):
            b = 0
            c = 0
            for y in range(int(ip_theta.shape[1]/8)):
                for i in range(a, a+8):
                    for j in range(b, b+8):
                        bin_no = int(histo_center[i][j]/20)
                        if ip_theta[i][j] == histo_center[i][j]:
                            histo_list[k][c][bin_no] += ip_img[i][j]
```

```
                else:
                    diff1 = abs(bins_table[histo_center[i][j]][0] - ip_theta[i][j])
                    diff2 = abs(bins_table[histo_center[i][j]][1] - ip_theta[i][j])
                    if ip_theta[i][j] < histo_center[i][j]:
                        histo_list[k][c][bin_no] += (diff2/20)*ip_img[i][j]
                        histo_list[k][c][bin_no - 1] += (diff1/20)*ip_img[i][j]
                    else:
                        histo_list[k][c][bin_no] += (diff1/20)*ip_img[i][j]
                        if bin_no == 8:
                            histo_list[k][c][0] += (diff2/20)*ip_img[i][j]
                        else:
                            histo_list[k][c][bin_no + 1] += (diff2/20)*ip_img[i][j]
            b += 8
            c += 1
        a += 8
        k += 1

    # block_array will store the normalized histogram block wise
    block_array = np.zeros((histo_list.shape[0] - 1, histo_list.shape[1] - 1, 36))

    # hog_discriptor will create a one dimensional array of all the normalized hirtograms
    hog_discriptor_op = []
    for i in range(histo_list.shape[0] - 1):
        for j in range(histo_list.shape[1] - 1):
            l1 = histo_list[i][j]
            l2 = histo_list[i+1][j]
            l3 = histo_list[i][j+1]
            l4 = histo_list[i+1][j+1]

            norm_factor = math.sqrt(np.sum(np.square(l1)) +  np.sum(np.square(l2)) + 
np.sum(np.square(l3)) + np.sum(np.square(l4)))
            np.seterr(divide='ignore', invalid='ignore')
            l1_, l2_, l3_, l4_ = np.true_divide(l1, norm_factor), np.true_divide(l2, norm_factor), 
np.true_divide(l3, norm_factor), np.true_divide(l4, norm_factor)
            block_array[i][j] = np.concatenate((l1_, l2_, l3_, l4_))
            hog_discriptor_op.extend(l1_.tolist())
            hog_discriptor_op.extend(l2_.tolist())
            hog_discriptor_op.extend(l3_.tolist())
            hog_discriptor_op.extend(l4_.tolist())

    return np.array(hog_discriptor_op).reshape((1, len(hog_discriptor_op)))
```

```
    # main function
    def hog(self):
        image_name = self.image_name
        img = imread(image_name)
        gs_img = self.greyscale_operation(img)
        grad_img, theta = self.gradient_operation(gs_img)
        hog_discriptor = self.hog_operation(grad_img, theta)
        return hog_discriptor
```

2. create_data.py:

```
import HoG as HoG
import numpy as np
import os

# function to create data
def create_data():
    files = os.listdir('./positive_train')
    for i in range(len(files)):
        hog_obj = HoG.HOG('./positive_train/'+files[i])
        hog = hog_obj.hog()
        if i == 0:
            train_X_1 = hog
        else:
            train_X_1 = np.vstack((train_X_1, hog))

    train_y_1 = np.full((train_X_1.shape[0],1), 1)

    files = os.listdir('./negative_train')
    for i in range(len(files)):
        hog_obj = HoG.HOG('./negative_train/'+files[i])
        hog = hog_obj.hog()
        if i == 0:
            train_X_0 = hog
        else:
            train_X_0 = np.vstack((train_X_0, hog))

    train_y_0 = np.full((train_X_0.shape[0],1), 0)

    train_x = np.vstack((train_X_1, train_X_0))
    train_x = np.nan_to_num(train_x)
    train_y = np.vstack((train_y_1, train_y_0))
```

```
    train = np.append(train_x, train_y, 1)

    # saving training data to csv file
    np.savetxt("train.csv",train,delimiter=",")

    files = os.listdir('./positive_test')
    for i in range(len(files)):
        hog_obj = HoG.HOG('./positive_test/'+files[i])
        hog = hog_obj.hog()
        if i == 0:
            test_X_1 = hog
        else:
            test_X_1 = np.vstack((test_X_1, hog))

    test_y_1 = np.full((test_X_1.shape[0],1), 1)

    files = os.listdir('./negative_test')
    for i in range(len(files)):
        hog_obj = HoG.HOG('./negative_test/'+files[i])
        hog = hog_obj.hog()
        if i == 0:
            test_X_0 = hog
        else:
            test_X_0 = np.vstack((test_X_0, hog))

    test_y_0 = np.full((test_X_0.shape[0],1), 0)

    test_x = np.vstack((test_X_1, test_X_0))
    test_x = np.nan_to_num(test_x)
    test_y = np.vstack((test_y_1, test_y_0))
    test = np.append(test_x, test_y, 1)

    # saving testing data to csv file
    np.savetxt("test.csv",test,delimiter=",")

create_data()
```

Project 2: Human Detection
Computer Vision (CS 6643)
Bhushan Manohar Newalkar (bmn258)

3. neural_net.py

```
import numpy as np

class MLP:
	def __init__(self, w1, b1, w2, b2, lr):
		self.fc1 = FCLayer(w1, b1, lr)
		self.rel = ReLU()
		self.fc2 = FCLayer(w2, b2, lr)
		self.sig = Sigmoid()

	# function to calculate mean squared error
	def MSE(self, prediction, target):
		return (0.5*(target-prediction)**2).sum()

	# function to calculate the error
	def MSEGrad(self, prediction, target):
		return -(target - prediction)

	# training neural network
	def train(self, X, y, steps):
		stop = False
		prev_loss = 0.0
		s = 0

		# training will end when either epoch iterations are completed or when error is 
very low
		while stop != True and s != steps:
			i = s % y.size
			xi = np.expand_dims(X[i], axis=0)
			yi = np.expand_dims(y[i], axis=0)

			pred = self.fc1.forward(xi)
			pred = self.rel.forward(pred)
			pred = self.fc2.forward(pred)
			pred = self.sig.forward(pred)
			loss = self.MSE(pred, yi)

			if round(abs(loss - prev_loss), 6) == 0.0:
				print("Epochs:", s/y.size+1)
				stop = True
				break
```

```
                prev_loss = loss
                grad = self.MSEGrad(pred, yi)
                grad = self.sig.backward(grad)
                grad = self.fc2.backward(grad)
                grad = self.rel.backward(grad)
                grad = self.fc1.backward(grad)
                s += 1

    # prediction using trained NN
    def predict(self, X):
        pred = self.fc1.forward(X)
        pred = self.rel.forward(pred)
        pred = self.fc2.forward(pred)
        pred = self.sig.forward(pred)
        # pred = np.round(pred)
        return np.ravel(pred)

class FCLayer:

    def __init__(self, w, b, lr):
        self.lr = lr
        self.w = w
        self.b = b

    # forward pass
    def forward(self, input):
        self.input = input
        h = np.dot(input, self.w) + self.b
        return h

    # backward pass
    def backward(self, gradients):
        input = self.input
        x_ = np.dot(gradients, self.w.T)
        self.w = self.w - np.dot(input.T, gradients)*self.lr
        self.b = self.b - gradients*self.lr
        return x_

class Sigmoid:

    def __init__(self):
        None
```

```
    def sigmoid_func(self, a):
        return 1/(1+np.exp(-a))

    #forward pass
    def forward(self, input):
        self.input = input
        sig_val = self.sigmoid_func(input)
        return sig_val

    # backward pass
    def backward(self, gradients):
        input = self.input
        sig_val_back = gradients*(1 - self.sigmoid_func(input))*self.sigmoid_func(input)
        return sig_val_back

class ReLU:
    def __init__(self):
        None

    # forward pass
    def forward(self, input):
        self.input = input
        input[input<0] = 0
        return input

    # backward pass
    def backward(self, gradients):
        input = self.input
        input[input < 0] = 0
        input[input > 0] = 1
        input *= gradients
        return input
```

Project 2: Human Detection
Computer Vision (CS 6643)
Bhushan Manohar Newalkar (bmn258)

4. run_NN.py

```
import numpy as np
import neural_net as model

# function to load saved data
def load_data(path):
        data = np.genfromtxt(path, delimiter=',', dtype=float)
        return data[:,:-1], data[:,-1].astype(int)

train_x, train_y = load_data("train.csv")
test_x, test_y = load_data("test.csv")

# MLP Training
# learning rate
lr = 0.01

# random weight initialization
w1 = np.random.normal(0, .1, size=(train_x.shape[1], 250))
w2 = np.random.normal(0, .1, size=(250,1))
b1 = np.random.normal(0, .1, size=(1,250))
b2 = np.random.normal(0, .1, size=(1,1))

mlp = model.MLP(w1, b1, w2, b2, lr)

# set epoch values
epoch = 400
steps = epoch*train_y.size

# training neural network
mlp.train(train_x, train_y, steps)

# evaluation function to calculate accuracy
def evaluate(solutions, real):
        if(solutions.shape != real.shape):
                raise ValueError("Output is wrong shape.")
        predictions = np.array(solutions)
        labels = np.array(real)
        return (predictions == labels).sum() / float(labels.size)

# predicting on test data
solutions = mlp.predict(test_x)
```

```
# printing NN output
print("NN Output:")
print(solutions)

# printing predictions
print("Predictions")
solutions = np.round(solutions)
print(solutions)

# printing evaluation accuracy
print(evaluate(solutions, test_y))
```