Project 2 Human Detection

Jiaying Li
Jl10919

1.
File names for your source code : human_detection.py
the HOG feature files for image *crop001034b*. : crop001034b_HOG.txt
the LBP feature files for image *crop001034b*. : crop001034b_LBP.txt

2. how to run the program:
(1) place negative test images in ./ Test images (Neg),
place positive test images in ./Test images (Pos),
place negative training images in ./Training images (Neg),
place positive training images in ./Training images (Pos)
(2) open human_detection.py
Set all hyper-parameters in main function (line 432). For example:

```
hidden_layer_size = 200
```

```
epoch = 200
```

```
epoch_maximum = 1000
```

```
learning_rate = 0.1
```

```
monitor_threshold = 0.001
```

(3) run human_detection.py

3. method used to initialize the weight values:
random initialization with values within range [-0.5, 0.5]

4. Criteria you used to stop training
when change in average error between consecutive epochs is less than *monitor_threshold*, and average error < 0.01
or
when number of epochs reaches *epoch_maximum,*
stop training.
Here I set monitor_threshold = 0.001 and *epoch_maximum = 1000*

5. 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.
(1) hidden layer sizes of 200, HOG only : 90
(2) hidden layer sizes of 200, combined HOG-LBP: 90

(1) hidden layer sizes of 400, HOG only : 90
(2) hidden layer sizes of 400, combined HOG-LBP:90

6.
(1) hidden layer sizes of 200, learning rate=0.1, epoch = 90

Test Image	**Correct Class**	**HOG only**		**HOG-LBP**	
		Output	**Classification**	**Output**	**Classification**
crop001034b	Human	0.7388763750277293	human	0.5716416887754283	borderline
crop001070a	Human	0.9024570306736606	human	0.884048260831161	human
crop001278a	Human	0.7488217308902032	human	0.8923183220810181	human
crop001500b	Human	0.8461982426422734	human	0.9703786335755803	human
person_and_bike_151a	Human	0.8393576362989061	human	0.7166216559927834	human
00000003a_cut	No-human	0.3387542842262108	no-human	0.5485467737602782	borderline
00000090a_cut	No-human	0.25320373387803824	no-human	0.12421126832974055	no-human
00000118a_cut	No-human	0.4657475064774292	borderline	0.0757134517887644	no-human
no_person_no_bike_258_cut	No-human	0.6184807369499219	human	0.49715763336850005	borderline

no_person_no_bike_264_cut	No-human	0.20144858333939825	no-human	0.5647534774283219	borderline

the average error for the 10 test images using HOG: 0.294147
The SGD mean error on train data using HOG: 0.082774
the average error for the 10 test images using HOG-LBP: 0.277537
The SGD mean error on train data using HOG-LBP: 0.039238

(2) hidden layer sizes of 400,learning rate: 0.1,epoch= 90

Test Image	**Correct Class**	**HOG only**		**HOG-LBP**	
		Output	**Classification**	**Output**	**Classification**
crop001034b	Human	human	0.8781320087268393	0.8926056753951697	human
crop001070a	Human	human	0.9828356303701845	0.9877787571622693	human
crop001278a	Human	human	0.9684103178339539	0.9656537954640676	human
crop001500b	Human	human	0.981166062366835	0.9947033928722296	human
person_and_bike_151a	Human	human	0.9350926443042668	0.9814301924589768	human
00000003a_cut	No-human	no-human	0.19049509277506232	0.1322044559151823	no-human
00000090a_cut	No-human	no-human	0.022511053222529692	0.017798732973392503	no-human
00000118a_cut	No-human	no-human	0.12302180220032548	0.24676944103918	no-human

				08	
no_person_no_bike_258_cut	No-human	no-human	0.04168597260854564	0.177160562628222957	no-human
no_person_no_bike_264_cut	No-human	no-human	0.35873891882757825	0.96218273192382 65	human

the average error for the 10 test images using HOG: 0.099082
The SGD mean error on train data using HOG: 0.021979
the average error for the 10 test images using HOG-LBP: 0.171394
The SGD mean error on train data using HOG-LBP: 0.007789

7.
(1) When hidden layer has size 200, HOG-LBP performs better. When hidden layer has size 400, HOG performs better. (2)When the number of training epoch is too big, the over-fitting problem comes.
For example, when set 400 epochs for "hidden layer size of 200, hog only", the SGD mean error on train data: 0. 027403. However, the average error on test data using only HOG is 0.474342, and all test images are classified as no-human.
(3)When learning rate is too big, it is hard for mode to find the precise local optimal point to stay, but possibly to jump out of the global optimal to find the "global" optimal point. When learning rate is too small, it takes much more time to reach the optimal point but can locate at the more precise local optimal point.
After trying several combination of different learning rate and epoch, learning rate = 0.1, epoch = 90 performs better.
(4) from the tables above, we can see that model with hidden layer size of 400 perform better than model with hidden layer size of 400, with lower average error and more specific prediction(less borderline class).

8. Normalized gradient magnitude images for the 10 test images (Copy-and-paste from output image files.)
(1)crop001034b

(2) crop001070a

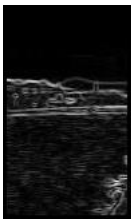
(3) crop001278a

(4) crop001500b

(5) person_and_bike_151a

(6) 00000003a_cut

(7) 00000090a_cut



(8) 00000118a_cut

(9) no_person_no_bike_258_cut

(10) no_person_no_bike_264_cut

10. source code of your program

```
import random
import cv2
import numpy as np
import os
import matplotlib.image as mpimg
import io


# wrap up the training and testing data to be used in following training.
def data_wrapper():
    training_data_HOG = []
    testing_data_HOG = []
    training_data_LBP = []
    testing_data_LBP = []
    training_label = []
    testing_label = []
    test_data_name = []
    train_data_name = []
    # get all the images under the path ./Training images (Pos)
    # and wrap them up as training data.
    # store the every image 's HOG, LBP,label(=1) and name
    for a in os.listdir('./Training images (Pos)'):
```

```python
        hog, lbp = output('Training images (Pos)', a)
        training_data_HOG.append(hog)
        training_data_LBP.append(lbp)
        training_label.append(1)
        train_data_name.append(a)
    # get all the images under the path ./Training images (Neg)
    # and wrap them up as training data.
    # store the every image 's HOG, LBP,label(=0) and name
    for b in os.listdir('./Training images (Neg)'):
        hog, lbp = output('Training images (Neg)', b)
        training_data_HOG.append(hog)
        training_data_LBP.append(lbp)
        training_label.append(0)
        train_data_name.append(b)
    # get all the images under the path ./Test images (Pos)
    # and wrap them up as training data.
    # store the every image 's HOG, LBP,label(=1) and name
    for c in os.listdir('./Test images (Pos)'):
        hog, lbp = output('Test images (Pos)', c, output_image=True)
        testing_data_HOG.append(hog)
        testing_data_LBP.append(lbp)
        testing_label.append(1)
        test_data_name.append(c)

    # get all the images under the path ./Test images (Neg)
    # and wrap them up as training data.
    # store the every image 's HOG, LBP,label(=0) and name
    for d in os.listdir('./Test images (Neg)'):
        hog, lbp = output('Test images (Neg)', d, output_image=True)
        testing_data_HOG.append(hog)
        testing_data_LBP.append(lbp)
        testing_label.append(0)
        test_data_name.append(d)
    return training_data_HOG, testing_data_HOG, training_data_LBP, testing_data_LBP, training_label, testing_label, test_data_name, train_data_name


# compute the HOG and LBP of every input image (dir = ./a/b)
def output(a, b, output_image=False):
    addr = './' + a + '/' + b
    origin = mpimg.imread(addr)
    gsr = grayscale_round(origin)
    h_grad, v_grad = sobel_operation(gsr, len(gsr), len(gsr[0]))
    mag = magnitude(h_grad, v_grad)
```

```
    if output_image == True:
        cv2.imwrite('./image_magnitude/' + b, mag)
    ang = gradient_angle(h_grad, v_grad)
    cells = cell(mag, ang)
    block = blocks(cells)
    return HOG(block), LBP(gsr)


# normalize the image matrix
def normalization(img):
    min_val = np.min(img.ravel())
    max_val = np.max(img.ravel())
    output = (img.astype('float') - min_val) / (max_val - min_val) * 255

    return output


# convert a RGB image to Grayscale image
def grayscale_round(list):
    gsr = np.zeros([len(list), len(list[0])], dtype=int)
    for i in range(len(gsr)):
        for j in range(len(gsr[0])):
            gsr[i][j] = np.around(0.299 * list[i][j][0] + 0.587 * list[i][j][1] + 0.114 * list[i][j][2])
    return gsr


# do sobel operation
def sobel_operation(gaussian_out, ga_height, ga_width):
    # horizontal sobel operator
    sobel_operator_x = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
    ])

    # vertical sobel operator
    sobel_operator_y = np.array([
        [1, 2, 1],
        [0, 0, 0],
        [-1, -2, -1]
    ])

    # initialize sobel-operation output
```

```
    sobel_xout = np.zeros([ga_height, ga_width], dtype=float)
    sobel_yout = np.zeros([ga_height, ga_width], dtype=float)

    # do cross-correlation operation
    resx = 0
    resy = 0
    for i in range(3, ga_height - 3):
        for j in range(3, ga_width - 3):
            resx = 0.0
            resy = 0.0
            for m in range(3):
                for n in range(3):
                    resx += gaussian_out[i + m - 1, j + n - 1] * sobel_operator_x[m, n]
                    resy += gaussian_out[i + m - 1, j + n - 1] * sobel_operator_y[m, n]
            sobel_xout[i, j] = resx
            sobel_yout[i, j] = resy

    return sobel_xout, sobel_yout


# calculate the magnitude
def magnitude(sobel_xout, sobel_yout):
    # so_height, so_width = sobel_yout.shape
    magnitude = np.sqrt(sobel_xout ** 2 + sobel_yout ** 2)
    # normorlize the magnitude
    magnitude = normalization(magnitude)

    return magnitude


def gradient_angle(sobel_xout, sobel_yout):
    # compute the angle of gradient (the output is in the range of [-pi,pi])
    angle = np.arctan2(sobel_yout, sobel_xout)

    return angle


# compute the cell data of HOG
def cell(mag, ang):
    # cell size : height/8, width/8, 9bins
    cell = np.zeros([int(len(mag) / 8), int(len(mag[0]) / 8), 9])
```

```
    for i in range(len(mag)):
        for j in range(len(mag[0])):
            ## make angle in (0,180)
            if ang[i][j] >= 170 and ang[i][j] < 350:
                ang[i][j] -= 180

            # add votes to each bin
            if ang[i][j] >= -20 and ang[i][j] < 0:
                tmp = (ang[i][j] + 20) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][0] += tmp
                cell[int(i / 8)][int(j / 8)][8] += mag[i][j] - tmp

            if ang[i][j] >= 0 and ang[i][j] < 20:
                tmp = (ang[i][j] - 0) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][1] += tmp
                cell[int(i / 8)][int(j / 8)][0] += mag[i][j] - tmp
            if ang[i][j] >= 20 and ang[i][j] < 40:
                tmp = (ang[i][j] - 20) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][2] += tmp
                cell[int(i / 8)][int(j / 8)][1] += mag[i][j] - tmp
            if ang[i][j] >= 40 and ang[i][j] < 60:
                tmp = (ang[i][j] - 40) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][3] += tmp
                cell[int(i / 8)][int(j / 8)][2] += mag[i][j] - tmp
            if ang[i][j] >= 60 and ang[i][j] < 80:
                tmp = (ang[i][j] - 60) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][4] += tmp
                cell[int(i / 8)][int(j / 8)][3] += mag[i][j] - tmp
            if ang[i][j] >= 80 and ang[i][j] < 100:
                tmp = (ang[i][j] - 80) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][5] += tmp
                cell[int(i / 8)][int(j / 8)][4] += mag[i][j] - tmp
            if ang[i][j] >= 100 and ang[i][j] < 120:
                tmp = (ang[i][j] - 100) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][6] += tmp
                cell[int(i / 8)][int(j / 8)][5] += mag[i][j] - tmp
            if ang[i][j] >= 120 and ang[i][j] < 140:
                tmp = (ang[i][j] - 120) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][7] += tmp
                cell[int(i / 8)][int(j / 8)][6] += mag[i][j] - tmp
            if ang[i][j] >= 140 and ang[i][j] < 160:
                tmp = (ang[i][j] - 140) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][8] += tmp
                cell[int(i / 8)][int(j / 8)][7] += mag[i][j] - tmp
```

```
            if ang[i][j] >= 160 and ang[i][j] < 180:
                tmp = (ang[i][j] - 160) / 20 * mag[i][j]
                cell[int(i / 8)][int(j / 8)][0] += tmp
                cell[int(i / 8)][int(j / 8)][8] += mag[i][j] - tmp
    return cell


# computer the block data of HOG, blocks[i][j][k]
# i means row, j means column, and k means 4*9 bins,
# cell1:k=0-8, cell2:k=9-17,cell3:k=18-26,cell4:k=27-35
# here the blocks are stored without overlap.
def blocks(cells):
    blocks = np.zeros([len(cells) - 1, len(cells[0]) - 1, 36])
    for i in range(len(blocks)):
        for j in range(len(blocks[0])):
            for a in range(9):
                blocks[i][j][a] = cells[i][j][a]
            for b in range(9, 18):
                blocks[i][j][b] = cells[i][j + 1][b - 9]
            for c in range(18, 27):
                blocks[i][j][c] = cells[i + 1][j][c - 18]
            for d in range(27, 36):
                blocks[i][j][d] = cells[i + 1][j + 1][d - 27]
    return blocks


# convert the blocks to HOG
# and do l2 - normalization
def HOG(a):
    b = np.zeros([len(a), len(a[0]), 36])
    for i in range(len(b)):
        for j in range(len(b[0])):
            v = np.linalg.norm(a[i][j])
            # when the l2-norm is too small, like 0, will cause error
            # (invalid value encountered in double_scalars), so set as 1.
            if v < 1:
                v = 1
            # l2 - normalize
            for k in range(36):
                b[i][j][k] = a[i][j][k] / v
        # final input will be 1-D
    return np.reshape(b, [len(b) * len(b[0]) * 36, 1])
```

```python
# convert the image to LBP
def LBP(image):
    # build the bin mapping dictionary
    bin_dict = {}
    v = 0
    for k in [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]:
        bin_dict[k] = v
        v += 1

    # len(image)/16 * len(a[0])/16 blocks and each blocks have 59 bins
    lbp = np.zeros([len(image) / 16, len(image[0]) / 16, 59])
    for i in range(len(image)):
        for j in range(len(image[0])):
            bit8 = []
            dec = 0
            if i == 0 or j == 0 or i == 159 or j == 95:
                lbp[i / 16][j / 16][58] += 1
                continue
            for x, y in [(-1, -1), (-1, 0), (-1, 1), (0, 1), (1, 1), (1, 0), (1, -1), (0, -1)]:
                if image[i + x][j + y] <= image[i][j]:
                    bit8.append(0)
                else:
                    bit8.append(1)
            bit8 = [str(x) for x in bit8]
            bit8 = ''.join(bit8)
            dec = int(bit8, 2)
            # add one to the bin that this decimal number corresponds to
            if dec in bin_dict.keys():
                bin = bin_dict[dec]
            else:
                bin = 58
            lbp[i / 16][j / 16][bin] += 1
    # normalize
    lbp = lbp / 256
    # final input will be 1-D
    return np.reshape(lbp, [len(lbp) * len(lbp[0]) * 59, 1])

```

```python

# concatenate hog and lbp to create hog-lbp
def HOG_LBP(hog, lbp):
    hl = []
    for i in range(len(hog)):
        c = np.concatenate((hog[i], lbp[i]), axis=0)
        hl.append(c)
    return hl


# sigmoid neuron
# if deriv = false, output is forward calculation
# if deriv = True, output is backward calculation, the derivative
def sigmoid(x, deriv=False):
    if (deriv == True):
        return x * (1 - x)
    else:
        return 1 / (1 + np.exp(-x))


# relu neuron
# if deriv = false, output is forward calculation
# if deriv = True, output is backward calculation, the derivative
def relu(x, deriv=False):
    if (deriv == True):
        return np.maximum(x, 0)
    else:
        return np.greater(x, 0).astype(int)


# Stochastic Gradient Descent aka online training
def SGD_train(layer1size, layer2size, layer3size, epoch, max_epoch, train_data, train_label, studyratio, threshold):
    np.random.seed(1)
    # 初始化各层单元之间的权值，即输入层到隐藏层，隐藏层到输出层，分别是 w1，w2
    # initial the weights in [-0.5,0.5], bias = -1
    # w1 and b1 and the weights and bias between input layer and hidden layer
    # w2 and b2 and the weights and bias between hidder layer and output layer
    w1 = -0.5 + np.random.random((layer1size, layer2size))  # (7524, 200)
    b1 = np.ones((1, layer2size)) * (-1)
    w2 = -0.5 + np.random.random((layer2size, layer3size))
    b2 = np.ones((1, layer3size)) * (-1)
```

```
    # record the mean of last epoch
    last_mean = 0

    # do epoch
    for i in range(epoch):

        # shuttle the training data every time finishing one epoch
        totall_index = range(len(train_data))
        random_index = random.sample(totall_index, len(train_data))
        # store the error of each training data
        epoch_diff = []

        # training according to the shuttle order
        # update the weights and bias one time after training one training data
        for j in random_index:
            # forward propagation
            output0 = np.array(train_data[j])
            output0 = output0.T
            datapass01 = np.dot(output0, w1) + b1
            output1 = relu(datapass01)
            datapass02 = np.dot(output1, w2) + b2
            output2 = sigmoid(datapass02)

            # backward propagation
            diff = train_label[j] - output2
            error2 = diff * sigmoid(output2, deriv=True)
            error1sum = np.dot(error2, w2.T)
            error1 = error1sum * relu(output1, deriv=True)

            # since we only read one image data once, we need change it to matrix for further calculation
            output1 = np.matrix(output1)
            error2 = np.matrix(error2)
            output0 = np.matrix(output0)
            error1 = np.matrix(error1)

            # update weights and bias
            w2 += np.dot(output1.T, error2) * studyratio
            b2 += error2 * studyratio
            w1 += np.dot(output0.T, error1) * studyratio
            b1 += error1 * studyratio

            epoch_diff.append(diff)
```

```

        # mean error of this epoch
        epoch_diff = np.abs(np.array(epoch_diff))
        mean = np.mean(epoch_diff)
        # epoch monitor
        # stop training when the change in average error between consecutive epochs is less than some threshold
        # or when the number of epochs is more than max_epoch
        if (abs(mean - last_mean) < threshold and mean < 0.01) or epoch > max_epoch:
            print ('mean', mean)
            print ('last_mean', last_mean)
            break
        # print mean error every 20 epochs
        if i % 10 == 0:
            print "The SGD mean error on train data: %f " % mean
        last_mean = mean

    return w1, w2, b1, b2


# do prediction for testing data
def bp_test(test_data, w1, w2, b1, b2):
    res = []
    for j in range(len(test_data)):
        # forward propagation
        output0 = np.array(test_data[j])
        output0 = output0.T  # (1, 7524)
        datapass01 = np.dot(output0, w1) + b1
        output1 = relu(datapass01)
        datapass02 = np.dot(output1, w2) + b2
        output2 = sigmoid(datapass02)
        # store prediction
        res.append(output2)
    return res


# calculate average error on test data
def mis(result, test_label):
    test_label = np.array(test_label)
    result = np.reshape(result, [10])
    diff = np.abs(test_label - result)
    return np.mean(diff)

```

```python

# print result of training average error, classification result and test average error
def classificaiton(layer1size, layer2size, layer3size, epoch, max_epoch, train_data, train_label,
                    studyratio, threshold, test_data, testing_label, test_data_name, HOG=False, HOG_LBP=False):
    if HOG == True:
        print "------Start training with HOG------"
    if HOG_LBP == True:
        print "------Start training with HOG and LBP------"
    # train the bp-nn model with training data, only using HOG
    w1, w2, b1, b2 = SGD_train(layer1size, layer2size, layer3size, epoch, max_epoch, train_data, train_label,
                                studyratio, threshold)
    # get prediction of testing data
    result = bp_test(test_data, w1, w2, b1, b2)
    # classify every test image according to following rules
    for i in range(len(test_data_name)):
        if result[i] >= 0.6:
            print(test_data_name[i] + ': human', float(result[i]))
        elif result[i] <= 0.4:
            print(test_data_name[i] + ': no-human', float(result[i]))
        else:
            print(test_data_name[i] + ': borderline', float(result[i]))
    # calculate the average error on test data
    error = mis(result, testing_label)
    # print  average error
    if HOG == True:
        print "The average error on test data using only HOG: %f " % error

    if HOG_LBP == True:
        print "The average error on test data using HOG-LBP: %f " % error


if __name__ == "__main__":
    # initialize the hyper-parameters
    hidden_layer_size = 400
    epoch = 100
    epoch_maximum = 1000
    learning_rate = 0.07
    monitor_threshold = 0.001

    # get data
```

```
    train_data_hog, test_data_hog, train_data_lbp, test_data_lbp, training_label, testing_label, test_data_name, train_data_name = data_wrapper()

    # get HOG-LBP data of training data and testing data
    train_hl = HOG_LBP(train_data_hog, train_data_lbp)
    test_hl = HOG_LBP(test_data_hog, test_data_lbp)

    # save hog and lbp of crop001034b.bmp
    crop001034b_HOG, crop001034b_LBP = output('Test images (Pos)', 'crop001034b.bmp')
    np.savetxt("crop001034b_HOG.txt", crop001034b_HOG)
    np.savetxt("crop001034b_LBP.txt", crop001034b_LBP)

    # train bp-neural network and classify test data with hog
    classificaiton(len(train_data_hog[0]), hidden_layer_size, 1, epoch, epoch_maximum, train_data_hog, training_label,
                   learning_rate, monitor_threshold, test_data_hog, testing_label, test_data_name, HOG=True)
    # train bp-neural network and classify test data with hog-lbp
    classificaiton(len(train_hl[0]), hidden_layer_size, 1, epoch, epoch_maximum, train_hl, training_label,
                   learning_rate, monitor_threshold, test_hl, testing_label, test_data_name, HOG_LBP=True)
```