Homework 1: Face Detection Report

110550168

Part I. Implementation (6%):

● Please screenshot your code snippets of Part 1, Part 2, Part 4, and explain your implementation.

Part 1:

```python
# Begin your code (Part 1)
# raise NotImplementedError("To be implemented")
i=os.getcwd() # get the absolute path of the current execution place
dp=os.path.normpath(dataPath) #  normalize the datapath, make " / " become " \ "
path = os.path.join(i,dp) # join current path and the given path
dataset = [] # to store the input image
for filename in os.listdir(path): # go through all folder in train or test (face or non-face)
    for second_file in os.listdir(os.path.join(path,filename)): # go through all data in face or non-face
        img = cv2.imread(os.path.join(os.path.join(path,filename),second_file)) # read image
        if img is not None: # check if img is none or not
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # convert the color to grayscale
            if filename == 'face': # if the folder is face, give a label 1
                dataset.append([img,1])
            else : # else, give a label 0
                dataset.append([img,0])

# End your code (Part 1)
```

I get the absolute path and store it in i. Then I normalize the datapath, , and join with i, so the path now is 'data\test' or 'data\train'. Then go through all folders in test/train by the first for loop, and join it to the path, which becomes 'data\test(train)\face' or 'data\test(train)\non-face'. The second for loop is to read through all data in face or non-face, and read the data out. Then I convert its color to grayscale and check whether the img needs to go by the index of the first for loop.

Part 2:

```python
        # Begin your code (Part 2)
        # raise NotImplementedError("To be implemented")
        cur = [] # to store WeakClassifier(feature)
        for i in range(len(features)): # go through all data in feature
            cur.append(WeakClassifier(features[i])) # change feature to weakclassifier
        bestClf = None # define best classifier
        bestError = float('inf') # let the current best error be the max of float
        for i in range(len(cur)): # go through all weakclassifier
            errorTMP = 0 # the sum of current weakclassifier's error
            for j in range(len(iis)): # go through all images
                if cur[i].classify(iis[j]) != labels[j]:
                    # check if the image's classify is same as the real label
                    errorTMP += weights[j] # if not, add weight to the sum
            if errorTMP < bestError: # check if current error is smaller
                bestError = errorTMP # change the current best error num
                bestClf = cur[i] # change the current best classifier

        # End your code (Part 2)
```

I change all features to weak classifiers and store them in cur. Then go through all cur to check which one is the best classifier.

Part 4:

```
    # Begin your code (Part 4)
    #raise NotImplementedError("To be implemented")
    i=os.getcwd() # get the absolute path of the current execution place
    dp=os.path.normpath(dataPath) # normalize the path
    imagePath = os.path.join(i,os.path.normpath('data\detect')) # get detect's path
    txtpath = os.path.join(i,dataPath) # get .txt path
    with open(txtpath,'r') as txt: # open .txt
      line = txt.readlines() # read all line and store in line
    txt.close() # close .txt

    colorimg = [] # image with RGB color
    face = [] # the face area
    result = [] # is face or not
    img_num = [] # the image has how many faces
    image = None # the variable to store image that is read
    count = 0 # which image is now
    curlinenum = 0 # counting the current line
    tmp_face = [] # to store the face in one image
    tmp_result = [] # to store the result in one image
    for i in line: # go through all line
      num = i.split(' ') # split the space between the line
      if num[0][-4:] == ".jpg": # if the first word in the line is image path
        image = cv2.imread(os.path.join(imagePath,num[0])) # read the image by the path
        rgbcolor = cv2.cvtColor(image,cv2.COLOR_BGR2RGB) # convert to RGB
        colorimg.append(rgbcolor) # store RGB image
        image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY) # convert to gray
        curlinenum = int(num[1]) # get the number of the faces in the image
        img_num.append(int(num[1])) # store the number of the faces in the image
        tmp_face = [] # clear the array
        tmp_result = [] # clear the array
        continue
    else:
      tmp = np.zeros((int(num[2]),int(num[3]))) # create an empty array, size = face size
      for pixel_y in range(int(num[2])): # go through pixels in y-axis
        for pixel_x in range(int(num[3])): # go through pixels in x-axis
          tmp[pixel_y][pixel_x] = image[int(num[1])+pixel_y][int(num[0])+pixel_x] # copy face to tmp
      tmp = cv2.resize(tmp,(19,19),interpolation=cv2.INTER_AREA) # resize to train size
      tmp_face.append(num) # add face's coordinates to num
      tmp_result.append(clf.classify(tmp)) # classify the face and store result

    count+=1 # count how many faces has loaded
    if count == curlinenum: # if the number of the faces is same as the given num
      count = 0 # reset count
      face.append(tmp_face) # add the face's coordinates list of the image to array
      result.append(tmp_result) # add the result list of the image to array

  count = 0 # reset count
  for i in colorimg: # go through all color images
    for f in range(img_num[count]): # go through all faces in one image
      if(result[count][f] == 1): # draw green if result of the face is 1
        for pixel_y in range(int(face[count][f][3])): # go through pixels in y-axis
          for pixel_x in range(int(face[count][f][2])):  # go through pixels in x-axis

            # draw a 3 pixels line for green
            if pixel_y == 0 or pixel_y == int(face[count][f][3]) - 1:
              i[int(face[count][f][1]) + pixel_y - 1][int(face[count][f][0]) + pixel_x] = (0,255,0)
              i[int(face[count][f][1]) + pixel_y][int(face[count][f][0]) + pixel_x] = (0,255,0)
              i[int(face[count][f][1]) + pixel_y + 1][int(face[count][f][0]) + pixel_x] = (0,255,0)
```

```
77                  elif pixel_x == 0 or pixel_x == int(face[count][f][2]) - 1:
78                    i[int(face[count][f][1]) + pixel_y][int(face[count][f][0]) + pixel_x - 1] = (0,255,0)
79                    i[int(face[count][f][1]) + pixel_y][int(face[count][f][0]) + pixel_x] = (0,255,0)
80                    i[int(face[count][f][1]) + pixel_y][int(face[count][f][0]) + pixel_x + 1] = (0,255,0)
81
82              else : # draw red if result of the face is 0
83                for pixel_y in range(int(face[count][f][3])):
84                  for pixel_x in range(int(face[count][f][2])):
85
86                    # draw a 3 pixels line for red
87                    if pixel_y == 0 or pixel_y == int(face[count][f][3]) - 1:
88                      i[int(face[count][f][1]) + pixel_y - 1][int(face[count][f][0]) + pixel_x] = (255,0,0)
89                      i[int(face[count][f][1]) + pixel_y][int(face[count][f][0]) + pixel_x] = (255,0,0)
90                      i[int(face[count][f][1]) + pixel_y + 1][int(face[count][f][0]) + pixel_x] = (255,0,0)
91                    elif pixel_x == 0 or pixel_x == int(face[count][f][2]) - 1:
92                      i[int(face[count][f][1]) + pixel_y][int(face[count][f][0]) + pixel_x - 1] = (255,0,0)
93                      i[int(face[count][f][1]) + pixel_y][int(face[count][f][0]) + pixel_x] = (255,0,0)
94                      i[int(face[count][f][1]) + pixel_y][int(face[count][f][0]) + pixel_x + 1] = (255,0,0)
95
96              plt.imshow(i) # plot the color image
97              plt.show() # show image
98              count+=1 # add one to count, meaning that to load next image
99
100         # End your code (Part 4)
```

I use arrays to store many things. Basically, the upper layer of the array is to store the image number, and the next layer will have different types. For example, array coloring stores the RGB colored image, array face stores the faces' coordinates and the size of the corresponding image, array result stores the classification of the faces, array img_num stores the number of the faces in the corresponding image.

Part II. Results & Analysis (12%):

● Please screenshot the results:

Accuracy when T = 10:

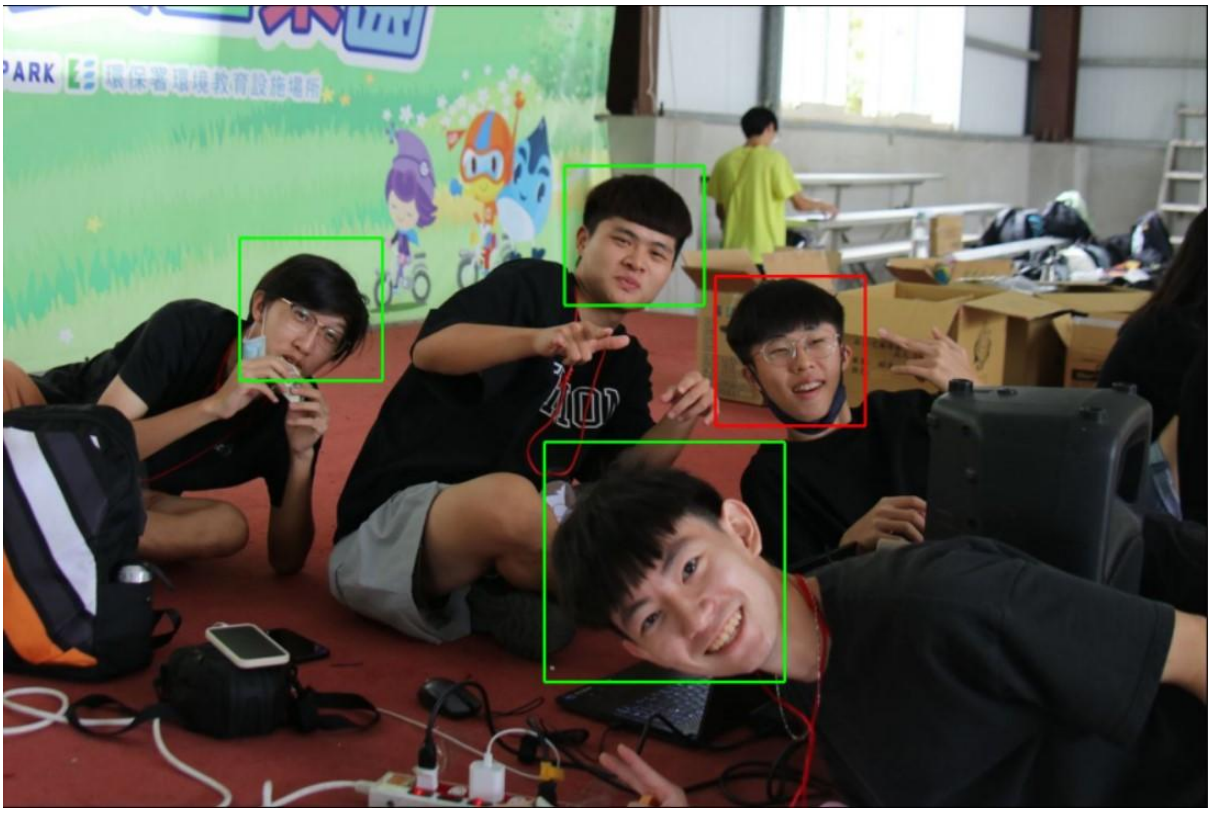
```
Evaluate your classifier with training dataset
False Positive Rate: 17/100 (0.170000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 183/200 (0.915000)

Evaluate your classifier with test dataset
False Positive Rate: 45/100 (0.450000)
False Negative Rate: 36/100 (0.360000)
Accuracy: 119/200 (0.595000)
```

Beatles:

American Congression:

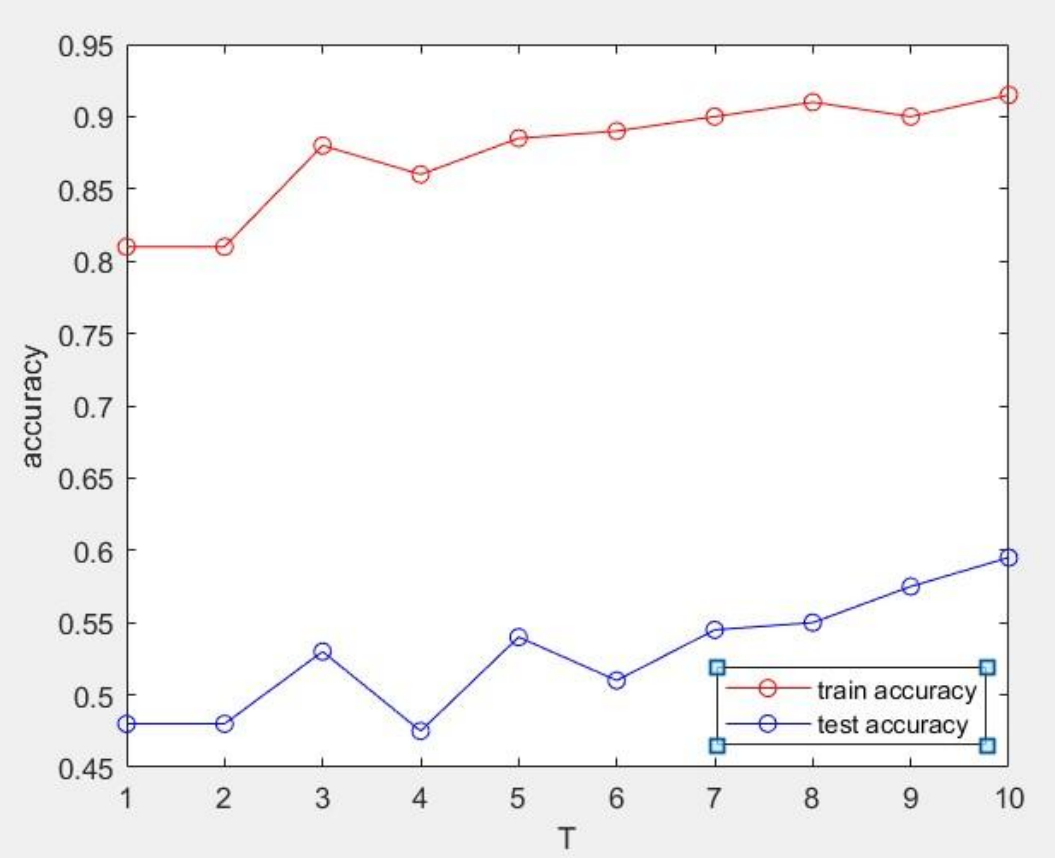
Own Picture:



Train/Test Accuracy from T = 1 ~ T = 10:

200 pic	train accuracy	test accuracy
T = 1	81%	48%
T = 2	81%	48%
T = 3	88%	53%
T = 4	86%	47.50%
T = 5	88.50%	54%
T = 6	89%	51%
T = 7	90%	54.50%
T = 8	91%	55%
T = 9	90%	57.50%
T = 10	91.50%	59.50%

Line Chart of Accuracy:



First, I figure out that the trend of training has proposition to the trend of testing. Although there are some exception, but I consider it as data differences. Next, the accuracy of training and testing will become higher when T grows. And the accuracy has a skyrocket at T = 3, so I think that we must train at least 3 times to make the accuracy more accurate.

Part III. Answer the questions (12%):

1. Please describe a problem you encountered and how you solved it
   (1) path problems, use os functions like getcwd() to get absolute path and normpath() to normalize the path, making the form of the path is same
   (2) filenotfound error, I figured out that the problem is the current directory is not the same as the file's directory, so I key in "cd folder_name" on the cmd to get to the right directory.
   (3) image showing problem, the image output was not as expected. I added " cmap = 'gray' " as the parameters of the imshow function.

2. What are the limitations of the **Viola-Jones' algorithm**?
   (1) true detection rate is high, but false detection is high also, due to the way of its detection to face
   (2) easy to detect face in frontal, but the orientation of the face will affect to the face detection easily

(3) the brightness of the image, since Viola-Jones' algorithm will uses grayscale images to detect
(4) restricted to black-white picture, but most pictures in real life are colored

3. Based on **Viola-Jones' algorithm**, how to improve the accuracy except changing the training dataset and parameter T?

(1) we can change features from rectangles to composite one, so that we can detect some slash line
(2) reduce the background of the face image, so that there will be few interruption

4. Other than **Viola-Jones' algorithm**, please propose another possible face detection method (no matter how good or bad, please come up with an idea). Please discuss the pros and cons of the idea you proposed, compared to the Adaboost algorithm.

I think that we can make a model of the contour face, eyes, nose, mouth by lines and dots, and then we overlap all models to find out the possible area of eyes, nose, and mouth.We can give the area a number to represent the probability to have eyes, nose, and mouth. When we get an image, we can compare it to the trained model and find out whether the image is face or not by using the probability. Also, I think that we can train models for different races and different ages to avoid some difference between them.

Compared to **Viola-Jones' algorithm**, I think that the one I came up with will be hard to train, will need lots of data, and may let face detection become complex, which may take more time. Also, I'm not sure whether relying on probability is possible. For the pros, I think that my idea can deal with the orientation of the face because I can train for the frontal and the profile. Besides, it is able to detect faces for images that are not bright enough because I only need the contour, but not the comparison with the surrounding.