

Homework 1: Face Detection Report

Part I. Implementation (6%):

- Screenshots and explanations of my codes:

Part 1:

```
18     1.In following two for-loops, load the training images by OpenCV from two folders,
19     'face' and 'non-face' respectively.
20     2.Let each image and a label meaning the image is face or not (1 or 0)
21     form a tuple and add it into the list of dataset.
22     '''
23     dataset = []
24     subPath = dataPath + '/face'
25     for filename in os.listdir(subPath):
26         img = cv2.imread(subPath + '/' + filename, cv2.IMREAD_GRAYSCALE)
27         dataset.append((img, 1))
28     subPath = dataPath + '/non-face'
29     for filename in os.listdir(subPath):
30         img = cv2.imread(subPath + '/' + filename, cv2.IMREAD_GRAYSCALE)
31         dataset.append((img, 0))
32     # raise NotImplementedError("To be implemented")
33     # End your code (Part 1)
```

Part 2:

```
158     # Begin your code (Part 2)
159     '''
160     1.Create an array and an empty list, 'Error' and 'Weakclf' respectively:
161     'Error': store errors of each weakClassifier.
162     'Weakclf': store weakClassifiers for every feature.
163     2.Update the error by the formula  $e[j] += w[i] * |h(x[i]) - y[i]|$  for every
164     weakClassifiers and images.
165     3.Choose the best classifier which has the smallest error around weakClassifiers
166     and return them.
167     '''
168     Error = np.zeros(len(features))
169     Weakclf = []
170     for i in range(len(features)):
171         Weakclf.append(WeakClassifier(
172             features[i], threshold=0, polarity=1))
173     for j in range(len(features)):
174         for i in range(len(iis)):
175             Error[j] += weights[i] * \
176                 abs(Weakclf[j].classify(iis[i]) - labels[i])
177     bestError = np.min(Error)
178     for i in range(len(Error)):
179         if Error[i] == bestError:
180             bestClf = Weakclf[i]
181             break
182     # raise NotImplementedError("To be implemented")
183     # End your code (Part 2)
```

Part 4:

```
19 # Begin your code (Part 4)
20 '''
21 1.Load the name of image and the information of its divided areas
22 | (left-top point (x,y), range) from 'detectData.txt'.
23 2.Load image twice by grayscale and default mode.
24 3.For each divided area, save it as a new smaller image,
25 | and then turn it into 19 x 19
26 4.Detect smaller image from 3. by the classifier, if it is classified as a face,
27 | add a green rectangular frame onto the original image;
28 | if not, add a red rectangular frame.
29 '''
30 file = open(dataPath, "r")
31 subPath = 'data/detect/'
32 num_testZone = 0
33 dic = {}
34 fileNames = []
35 for line in file:
36     len = 0
37     idx = 0
38     if num_testZone == 0:
39         for ch in line:
40             if ch == ' ':
41                 fileName = line[idx - len: idx]
42                 fileNames.append(fileName)
43                 dic[fileName] = []
44                 len = 0
45
46             elif line[idx] == line[-1]:
47                 num_testZone = int(line[idx - len: idx])
48             else:
49                 len += 1
50                 idx += 1
51         else:
52             range = []
53             for ch in line:
54                 if ch == ' ':
55                     range.append(int(line[idx - len: idx]))
56                     len = 0
57                 elif line[idx] == line[-1]:
58                     range.append(int(line[idx - len: idx]))
59                 else:
60                     len += 1
61                     idx += 1
62             num_testZone -= 1
63             dic[fileName].append(tuple(range))
64         file.close()
65     for fileName in fileNames:
66         img = cv2.imread(subPath + fileName, cv2.IMREAD_GRAYSCALE)
67         img_colorful = cv2.imread(subPath + fileName)
68         for zone in dic[fileName]:
69             copy_img = img[zone[1]:zone[1] +
70 |             |             |             | zone[3], zone[0]:zone[0] + zone[2]]
71             res_img = cv2.resize(copy_img, (19, 19))
```

```

71         if clf.classify(res_img) == 1:
72             BGR_form = (0, 255, 0)
73         else:
74             BGR_form = (0, 0, 255)
75         cv2.rectangle(
76             img_colorful, (zone[0], zone[1]), (zone[0] + zone[2], zone[1] + zone[3]),
77             BGR_form, 3)
78     cv2.imshow(fileName, img_colorful)
79     cv2.waitKey(0)
80     cv2.destroyAllWindows()
81     # raise NotImplementedError("To be implemented")
82     # End your code (Part 4)

```

Part 6 (Bonus):

```

187     # Begin your code (Part 6)
188     ...
189     Compared to default version, in part 6, I changed the calculation of error:
190     Error of every classifier is initialized to 1, and it will be reduced w[i]
191     when a classifier judge the image correctly; increase w[i] * 1.5 when judge wrong.
192     ( multiplier can be changed to other values, it is like a penalty mechanism. )
193     Thus, wrong detection will get bigger error and cause a classifier has
194     lower possibility of being selected.
195     ...
196     Error = np.ones(len(features))
197     Weakclf = []
198     for i in range(len(features)):
199         Weakclf.append(WeakClassifier(
200             features[i], threshold=0, polarity=1))
201     for j in range(len(features)):
202         for i in range(len(iis)):
203             if Weakclf[j].classify(iis[i]) != labels[i]:
204                 Error[j] -= weights[i]
205             else:
206                 Error[j] += weights[i] * 1.5
207     bestError = np.min(Error)
208     for i in range(len(Error)):
209         if Error[i] == bestError:
210             bestClf = Weakclf[i]
211             break
212     # raise NotImplementedError("To be implemented")
213     # End your code (Part 6)

```

Part II. Results & Analysis (12%):

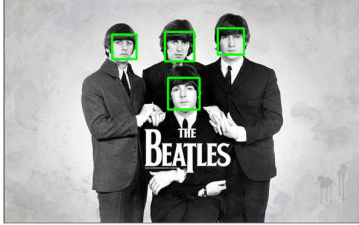
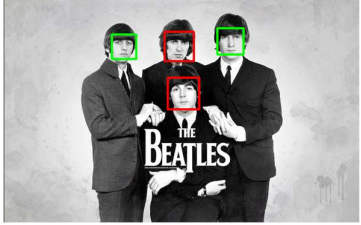
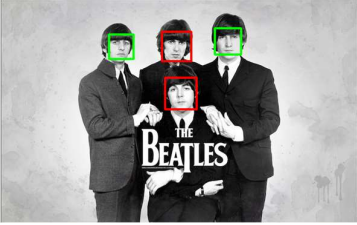



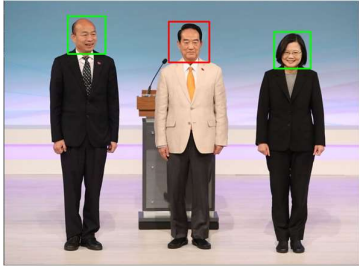
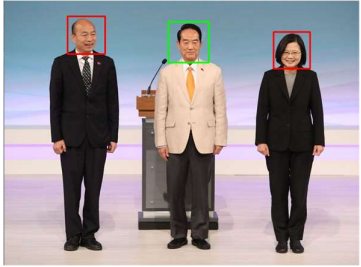
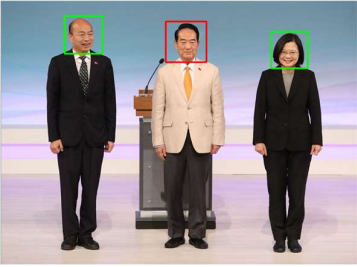
- Screenshot the results:

➤ **Method 1:**

```
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=

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)
```

Acc for each image with T = 1, T = 5 and T = 10 by method 1		
T = 1	T = 5	T = 10
 <p>100.0%</p>	 <p>50.0%</p>	 <p>50.0%</p>
 <p>86.7%</p>	 <p>20.0%</p>	 <p>20.0%</p>
 <p>66.7%</p>	 <p>33.3%</p>	 <p>66.7%</p>

➤ **Method 2 (Bonus):**

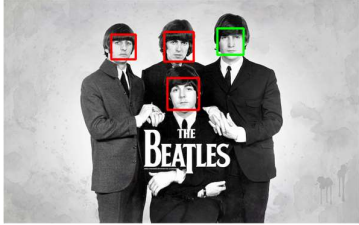

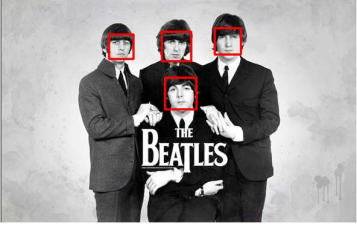
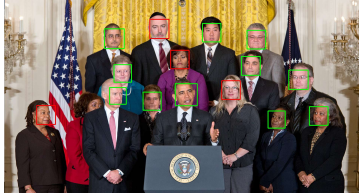


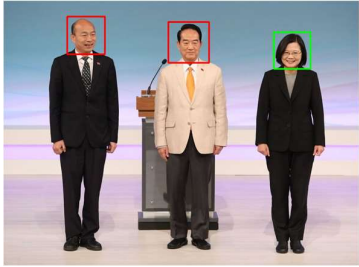
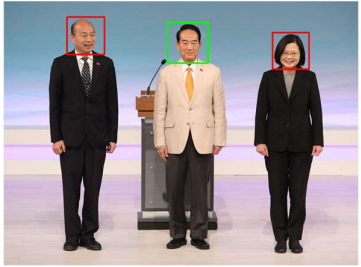
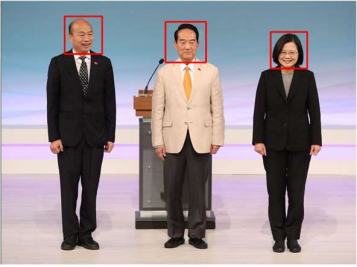
```

Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=

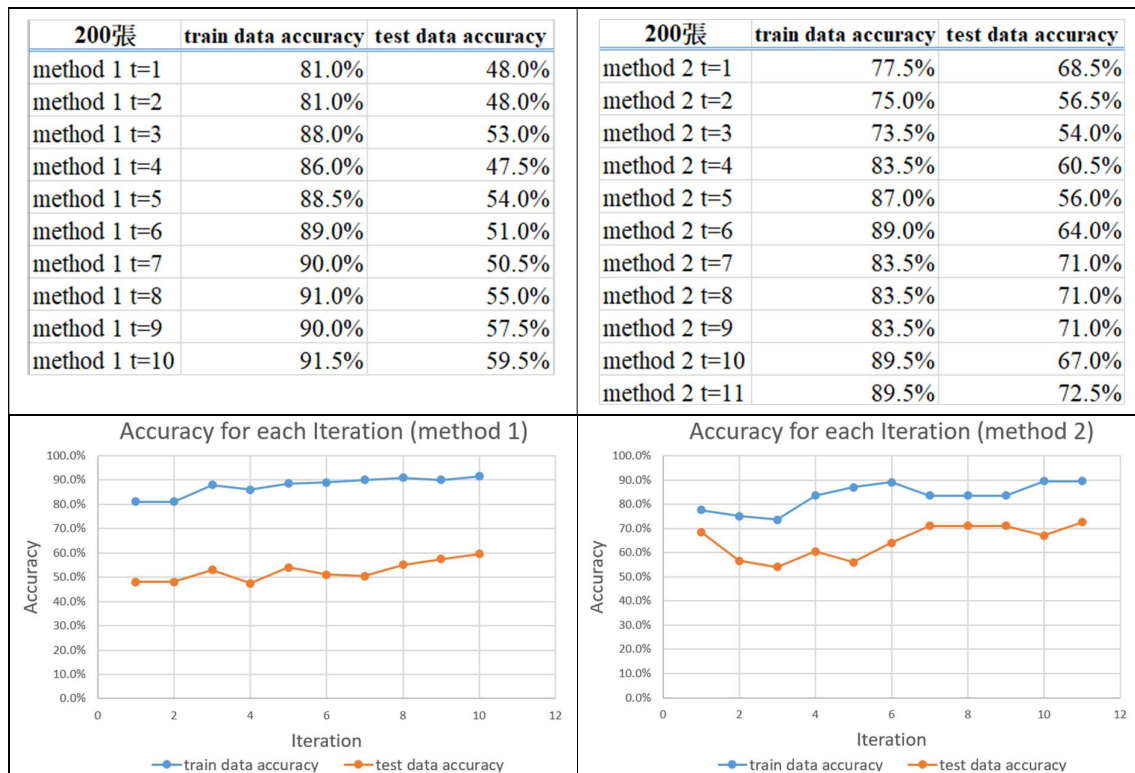
Evaluate your classifier with training dataset
False Positive Rate: 15/100 (0.150000)
False Negative Rate: 6/100 (0.060000)
Accuracy: 179/200 (0.895000)

Evaluate your classifier with test dataset
False Positive Rate: 30/100 (0.300000)
False Negative Rate: 36/100 (0.360000)
Accuracy: 134/200 (0.670000)

```

Acc for each image with T = 1, T = 5 and T = 10 by method 2		
T = 1	T = 5	T = 10
 <p>25.0%</p>	 <p>0.0%</p>	 <p>0.0%</p>
 <p>73.3%</p>	 <p>33.3%</p>	 <p>13.3%</p>
 <p>33.3%</p>	 <p>33.3%</p>	 <p>0.0%</p>

● **My analysis and observation:**



- a. In method 1 and 2, it can be observed that the accuracy of detecting train data and test data will be higher with the increasing of iteration times (T). By the line charts as shown above, lines of method 1 rises more smoothly than that of method 2. Though the line of method 2 shows an overall upward trend, there still exist some obvious oscillations.
- b. It can be inferred from repeated training and testing that the accuracy of detecting test data will converge even though under the condition of large parameter T (About 60% in method 1 and 70% in method 2). Besides, too large T will likely lead to “Overfitting” which may cause dropping of accuracy for real face detection.
- c. Although method 2 has better accuracy on detecting test data than method 1 based on same training dataset, it has worse performance on real image detection. So, we can infer that there likely exist more potential factors which haven’t been learned by the classifiers and cause bad accuracy.

Part III. Answer the questions (12%):

1. Please describe a problem you encountered and how you solved it.

My problem: In HW1, the problem I met is the usages of 'os' package. I haven't used it before, so I totally don't know how to use its function to access the data from the folders in the beginning.

How did I solve the problem: I looked up related documentations about 'os' on Internet and asked my friends.

2. What are the limitations of the Viola-Jones' algorithm?

- (1) This algorithm only can be used to detect frontal face.
- (2) For an image, if the brightness of its background is too bright or that of its face is too dark, face will become harder to be identified based on Viola-Jones' algorithm.
- (3) Although the face in the image is frontal and has obvious brightness distribution, if the face is partially blocked by anything such as mask, hat....., it will also be hard to be detected.

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

To improve the accuracy, I think we can modify the classifier function ($h(x)$), that is, design or find a more appropriate function to recognize an image is face or not.

In HW1, we define the function by default: If the value of Haar-like feature < 0 , return 1; Else, return 0. By the result of testing, accuracy is about 60% merely. Thus, we can try to assume that there exists a better function which may not linear, but polynomial or others and get the function via repetitive assumptions and experiments.

4. 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.

First, prepare the dataset of face which have same resolution in the beginning. Next, for every single pixel at the same coordinate in each image, statistically analyze their grayscale values, then remove outliers in analysis of each pixel. Finally, get the means of every single pixel with same coordinate and combine them into new "average" face image as our classifier.

For detecting face in an image, divide the image into many smaller images whose resolution are same as the classifier first. And then, we slide the classifier

on the image, for once sliding, Classifier can map to a small image. If the grayscale value fluctuation or contour is alike (has similar grayscale difference for near pixel), the classifier will say that it is a face.

Pros: If an image includes faces which are alike to classifier, they will be detected rapidly.

Cons: (1) Low accuracy. (2) It's time-consuming to analyze every single pixel. (3) It's easy to be misleading by the object whose contour is like the classifier. (4) If a real face whose features are very different from classifier or training dataset, it will likely not be detected

Compared to Adaboost algorithm, both are quick way for detection, but in training classifier, my idea would take more time than Adaboost. Besides, the performance of Adaboost algorithm will be more steady and better than mine.