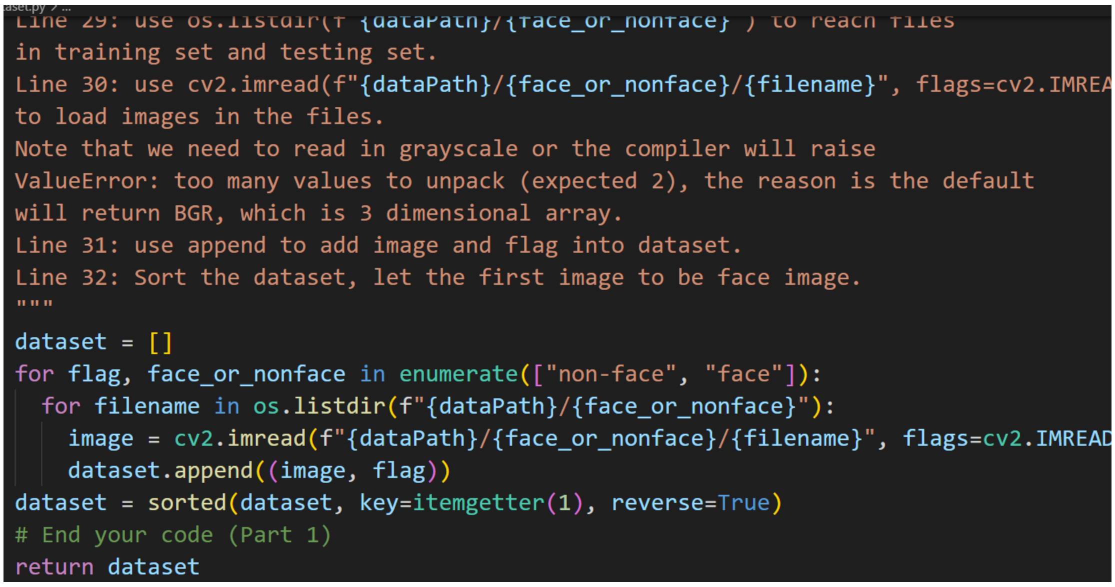
Homework 1: Face Detection

Part I. Implementation (6%):

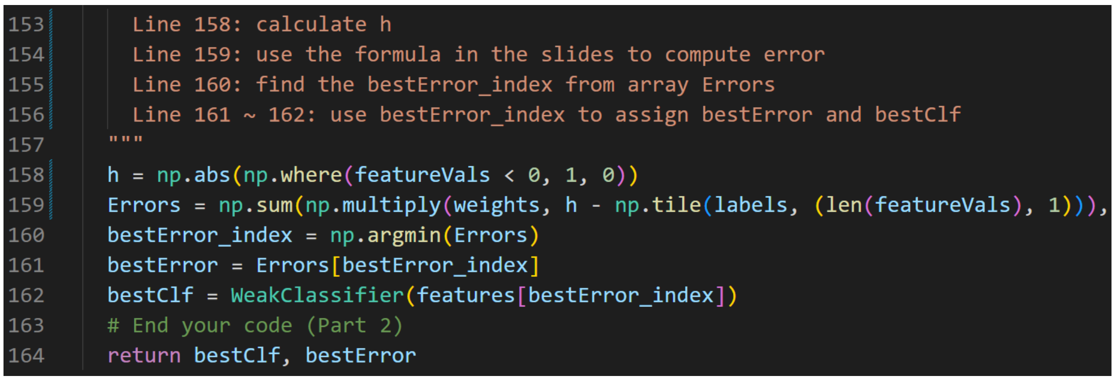
- Part 1.



```
Line 29: use os.listdir(f"{dataPath}/{face_or_nonface}") to reach files
in training set and testing set.
Line 30: use cv2.imread(f"{dataPath}/{face_or_nonface}/{filename}", flags=cv2.IMREA
to load images in the files.
Note that we need to read in grayscale or the compiler will raise
ValueError: too many values to unpack (expected 2), the reason is the default
will return BGR, which is 3 dimensional array.
Line 31: use append to add image and flag into dataset.
Line 32: Sort the dataset, let the first image to be face image.
"""
dataset = []
for flag, face_or_nonface in enumerate(["non-face", "face"]):
    for filename in os.listdir(f"{dataPath}/{face_or_nonface}"):
        image = cv2.imread(f"{dataPath}/{face_or_nonface}/{filename}", flags=cv2.IMREAD
        dataset.append((image, flag))
dataset = sorted(dataset, key=itemgetter(1), reverse=True)
# End your code (Part 1)
return dataset
```

  - Line 28: use enumerate to put flag 1 on face and flag 0 on non-face.
  - Line 29: use os.listdir(f"{dataPath}/{face_or_nonface}") to reach files in training set and testing set.
  - Line 30: use cv2.imread(f"{dataPath}/{face_or_nonface}/{filename}", flags=cv2.IMREAD_GRAYSCALE) to load images in the files. Note that we need to read in grayscale or the compiler will raise ValueError: too many values to unpack (expected 2), the reason is the default will return BGR, which is 3 dimensional array.
  - Line 31: use append to add image and flag into dataset.
  - Line 32: sort the dataset, and let the first image to be face image.
  - I found that if I changed the flag values to 1 on face and 0 on non-face we , the result will be quite good. But due to the lecture slides, we had to put flag = 1 on face and flag = 0 on non-face. As a result, the following results are all conducted under 1 on faces, 0 on non-faces. However, I still tried to put flag value to 0 on faces and put flag value to 1 on non-faces. Moreover, I had record the accuracy value in the discussion section.

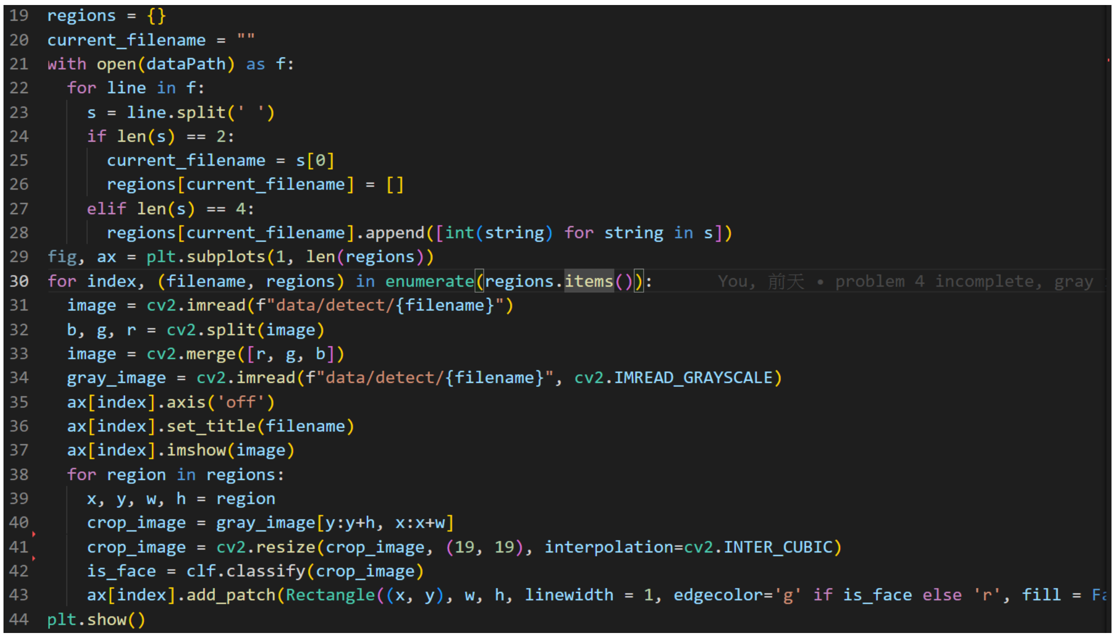
- Part 2.



```
153 |     Line 158: calculate h
154 |     Line 159: use the formula in the slides to compute error
155 |     Line 160: find the bestError_index from array Errors
156 |     Line 161 ~ 162: use bestError_index to assign bestError and bestClf
157   """
158   h = np.abs(np.where(featureVals < 0, 1, 0))
159   Errors = np.sum(np.multiply(weights, h - np.tile(labels, (len(featureVals), 1))),
160   bestError_index = np.argmin(Errors)
161   bestError = Errors[bestError_index]
162   bestClf = WeakClassifier(features[bestError_index])
163   # End your code (Part 2)
164   return bestClf, bestError
```

- Line 158: calculate h
- Line 159: use the formula in the slides to compute error
- Line 160: find the bestError_index from array Errors
- Line 161 ~ 162: use bestError_index to assign bestError and bestClf

- Part 4.

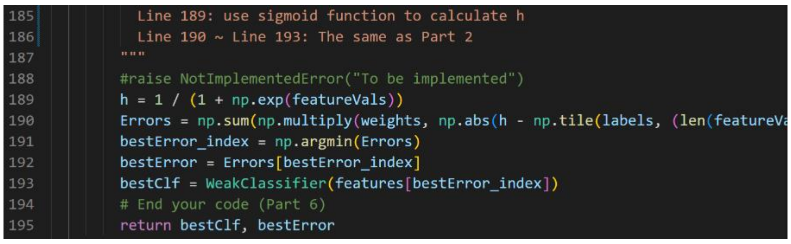


```
19 regions = {}
20 current_filename = ""
21 with open(dataPath) as f:
22   for line in f:
23     s = line.split(' ')
24     if len(s) == 2:
25       current_filename = s[0]
26       regions[current_filename] = []
27     elif len(s) == 4:
28       regions[current_filename].append([int(string) for string in s])
29 fig, ax = plt.subplots(1, len(regions))
30 for index, (filename, regions) in enumerate(regions.items()):        You, 前天 • problem 4 incomplete, gray
31   image = cv2.imread(f"data/detect/{filename}")
32   b, g, r = cv2.split(image)
33   image = cv2.merge([r, g, b])
34   gray_image = cv2.imread(f"data/detect/{filename}", cv2.IMREAD_GRAYSCALE)
35   ax[index].axis('off')
36   ax[index].set_title(filename)
37   ax[index].imshow(image)
38   for region in regions:
39     x, y, w, h = region
40     crop_image = gray_image[y:y+h, x:x+w]
41     crop_image = cv2.resize(crop_image, (19, 19), interpolation=cv2.INTER_CUBIC)
42     is_face = clf.classify(crop_image)
43     ax[index].add_patch(Rectangle((x, y), w, h, linewidth = 1, edgecolor='g' if is_face else 'r', fill = Fa
44 plt.show()
```

- Line 29 ~ Line 38: read the file and use regions to store each image's selected region
- Line 39: use plt.subplots to show image, where row is 1 and columns are len(regions)
- Line 40: use enumerate(regions.items()) to iterate all the selected regions.
- Line 41 ~ Line 46: read images and initialize axes.
- Line 47 ~ Line 52: iterate all the regions in the selected image,
- use classify to check whether the region is face or not, and use add_patch to put green or red
- rectangles on the regions.
- Line 53: show the result.

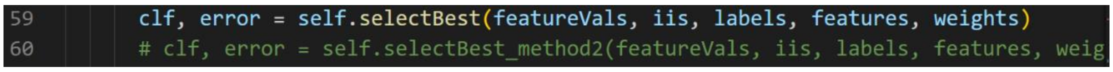
- Part 6.
  - I used the function Sigmoid mentioned in this article: Activation function 之群雄亂舞. However, I changed the sign of-z into z like the function at right.

$$g(x) = \frac{1}{1+e^x}$$



```
185               Line 189: use sigmoid function to calculate h
186               Line 190 ~ Line 193: The same as Part 2
187             """
188             #raise NotImplementedError("To be implemented")
189             h = 1 / (1 + np.exp(featureVals))
190             Errors = np.sum(np.multiply(weights, np.abs(h - np.tile(labels, (len(featureVa
191             bestError_index = np.argmin(Errors)
192             bestError = Errors[bestError_index]
193             bestClf = WeakClassifier(features[bestError_index])
194             # End your code (Part 6)
195             return bestClf, bestError
```

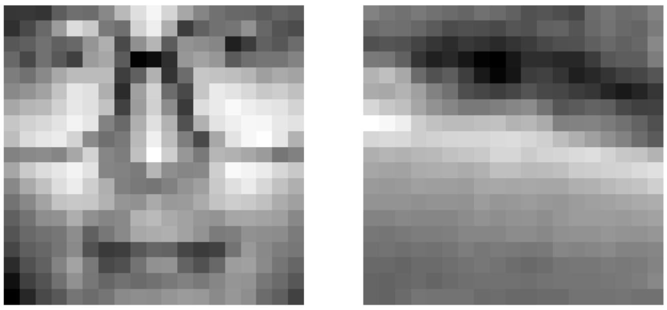
  - Line 189: use sigmoid function to calculate h
  - Line 190 ~ Line 193: The same as Part 2
  - To change the method, please go to the line 60 in adaboost.py



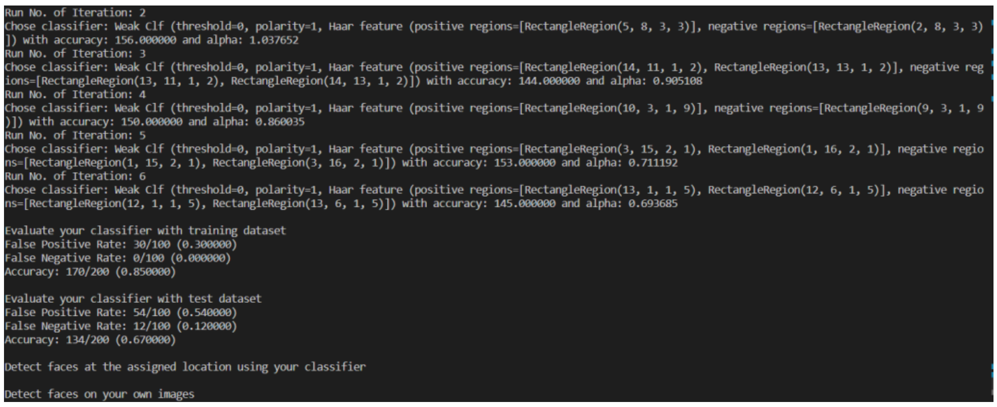
```
59          clf, error = self.selectBest(featureVals, iis, labels, features, weights)
60          # clf, error = self.selectBest_method2(featureVals, iis, labels, features, weig
```

Part II. Results & Analysis (12%):

- Screenshots for results
  - Result for Part 1:

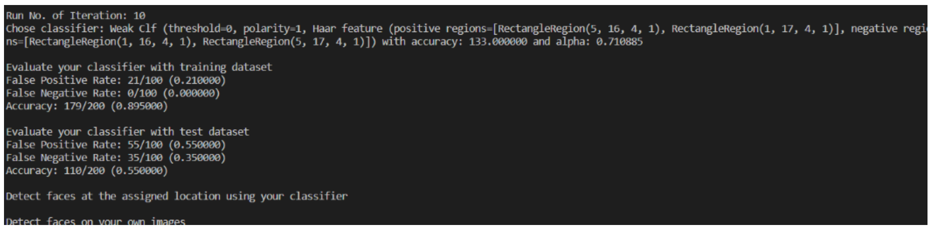


  - Result for Part 2: (I only paste the best result on testing set in each cases)
    - Method 1(face = 1, non-face = 0), T = 6:



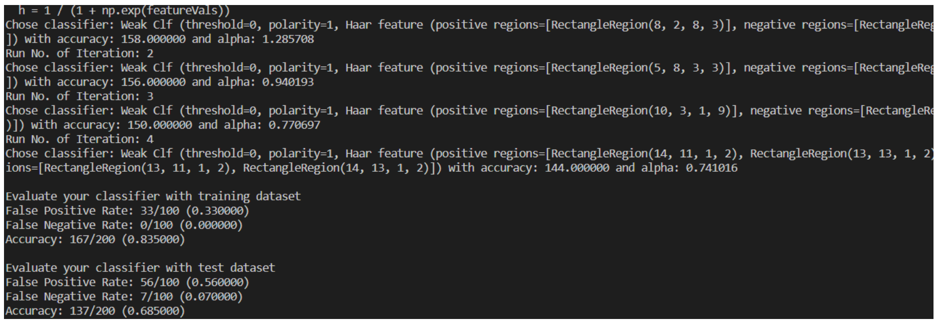
```
Run No. of Iteration: 2
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 8, 3, 3)], negative regions=[RectangleRegion(2, 8, 3, 3)
]) with accuracy: 156.000000 and alpha: 1.037652
Run No. of Iteration: 3
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(14, 11, 1, 2), RectangleRegion(13, 13, 1, 2)], negative reg
ions=[RectangleRegion(13, 11, 1, 2), RectangleRegion(14, 13, 1, 2)]) with accuracy: 144.000000 and alpha: 0.905108
Run No. of Iteration: 4
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 3, 1, 9)], negative regions=[RectangleRegion(9, 3, 1, 9
)]) with accuracy: 150.000000 and alpha: 0.860035
Run No. of Iteration: 5
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(3, 15, 2, 1), RectangleRegion(1, 16, 2, 1)], negative regio
ns=[RectangleRegion(1, 15, 2, 1), RectangleRegion(3, 16, 2, 1)]) with accuracy: 153.000000 and alpha: 0.711192
Run No. of Iteration: 6
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(13, 1, 1, 5), RectangleRegion(12, 6, 1, 5)], negative regio
ns=[RectangleRegion(12, 1, 1, 5), RectangleRegion(13, 6, 1, 5)]) with accuracy: 145.000000 and alpha: 0.693685

Evaluate your classifier with training dataset
False Positive Rate: 30/100 (0.300000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 170/200 (0.850000)

Evaluate your classifier with test dataset
False Positive Rate: 54/100 (0.540000)
False Negative Rate: 12/100 (0.120000)
Accuracy: 134/200 (0.670000)

Detect faces at the assigned location using your classifier

Detect faces on your own images
```

    - Method 1(face = 0, non-face = 1), T = 10:



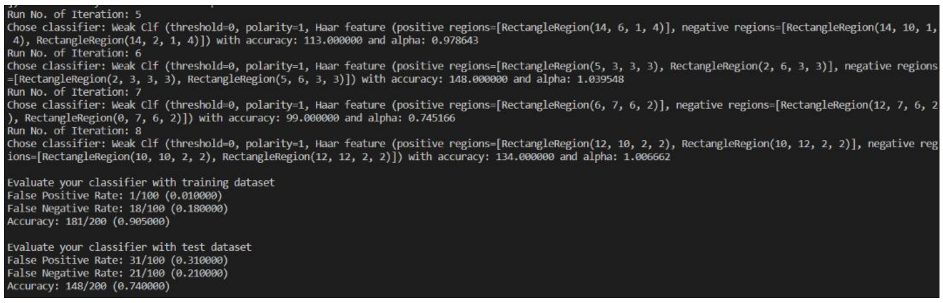
```
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 16, 4, 1), RectangleRegion(1, 17, 4, 1)], negative regi
ns=[RectangleRegion(1, 16, 4, 1), RectangleRegion(5, 17, 4, 1)]) with accuracy: 133.000000 and alpha: 0.710885

Evaluate your classifier with training dataset
False Positive Rate: 21/100 (0.210000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 179/200 (0.895000)

Evaluate your classifier with test dataset
False Positive Rate: 55/100 (0.550000)
False Negative Rate: 35/100 (0.350000)
Accuracy: 110/200 (0.550000)

Detect faces at the assigned location using your classifier

Detect faces on your own images
```

- Method 2(face = 1, non-face = 0), T = 4:



```
  h = 1 / (1 + np.exp(featureVals))
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(8, 2, 8, 3)], negative regions=[RectangleReg
]) with accuracy: 158.000000 and alpha: 1.285708
Run No. of Iteration: 2
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 8, 3, 3)], negative regions=[RectangleReg
]) with accuracy: 156.000000 and alpha: 0.940193
Run No. of Iteration: 3
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 3, 1, 9)], negative regions=[RectangleRe
)]) with accuracy: 150.000000 and alpha: 0.770697
Run No. of Iteration: 4
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(14, 11, 1, 2), RectangleRegion(13, 13, 1, 2)
ions=[RectangleRegion(13, 11, 1, 2), RectangleRegion(14, 13, 1, 2)]) with accuracy: 144.000000 and alpha: 0.741016

Evaluate your classifier with training dataset
False Positive Rate: 33/100 (0.330000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 167/200 (0.835000)

Evaluate your classifier with test dataset
False Positive Rate: 56/100 (0.560000)
False Negative Rate: 7/100 (0.070000)
Accuracy: 137/200 (0.685000)
```

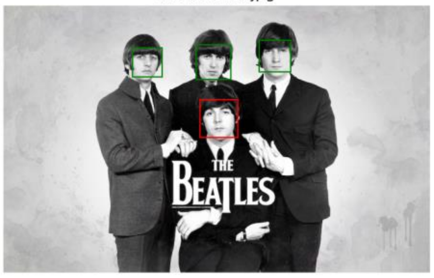
- Method 2(face = 0, non-face = 1), T = 8:



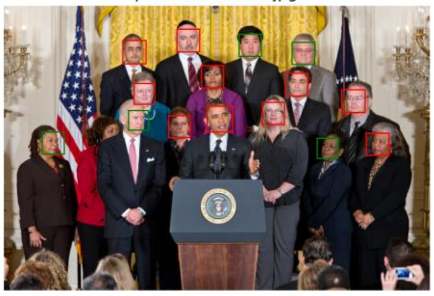
```
Run No. of Iteration: 5
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(14, 6, 1, 4)], negative regions=[RectangleRegion(14, 10, 1,
 4), RectangleRegion(14, 2, 1, 4)]) with accuracy: 113.000000 and alpha: 0.978643
Run No. of Iteration: 6
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 3, 3, 3), RectangleRegion(2, 6, 3, 3)], negative regions
=[RectangleRegion(2, 3, 3, 3), RectangleRegion(5, 6, 3, 3)]) with accuracy: 148.000000 and alpha: 1.039548
Run No. of Iteration: 7
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(6, 7, 6, 2)], negative regions=[RectangleRegion(12, 7, 6, 2
), RectangleRegion(0, 7, 6, 2)]) with accuracy: 99.000000 and alpha: 0.745166
Run No. of Iteration: 8
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(12, 10, 2, 2), RectangleRegion(10, 12, 2, 2)], negative reg
ions=[RectangleRegion(10, 10, 2, 2), RectangleRegion(12, 12, 2, 2)]) with accuracy: 134.000000 and alpha: 1.006662

Evaluate your classifier with training dataset
False Positive Rate: 1/100 (0.010000)
False Negative Rate: 18/100 (0.180000)
Accuracy: 181/200 (0.905000)

Evaluate your classifier with test dataset
False Positive Rate: 31/100 (0.310000)
False Negative Rate: 21/100 (0.210000)
Accuracy: 148/200 (0.740000)
```

- Result for Part 4 (I only paste the best result on testing set in each cases):
  - Method 1(face = 1, nonface = 0), T = 6:

the-beatles.jpg



p110912sh-0083.jpg



XaHpf_z51huQS_JPHs-jkPhBp0dLlxFJwt-sPLpGJB0.jpg

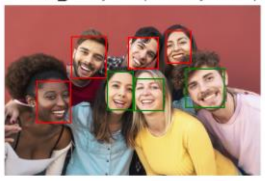
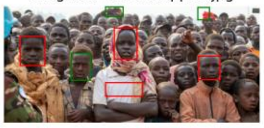
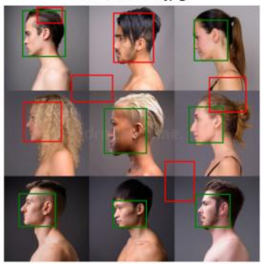


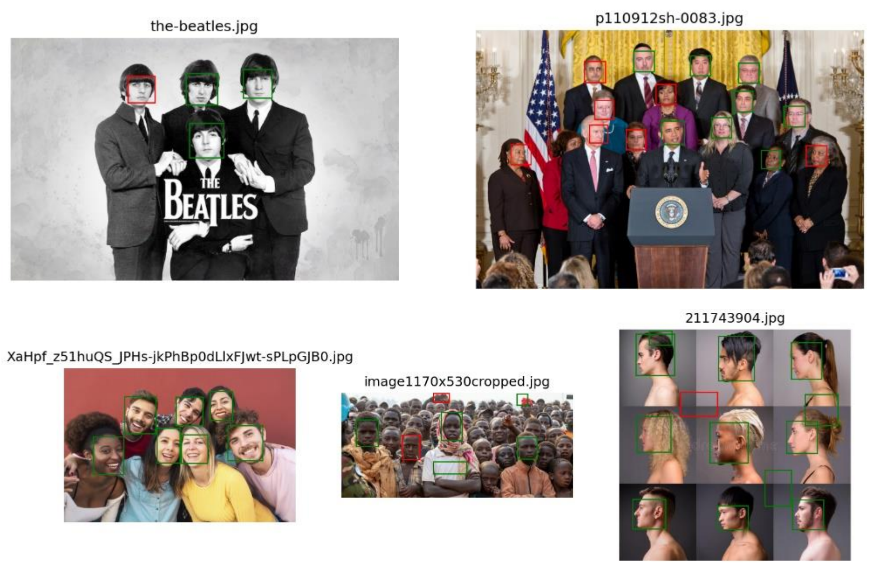
image1170x530cropped.jpg



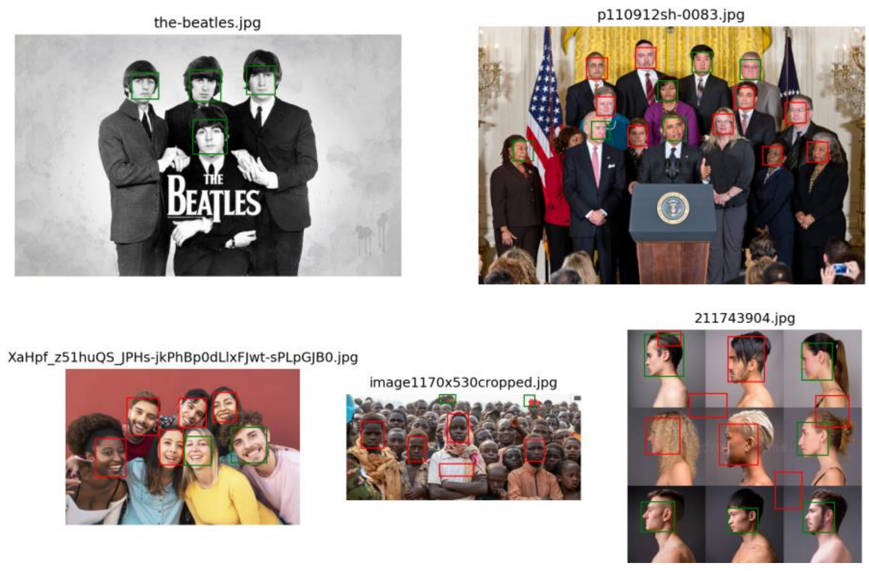
211743904.jpg



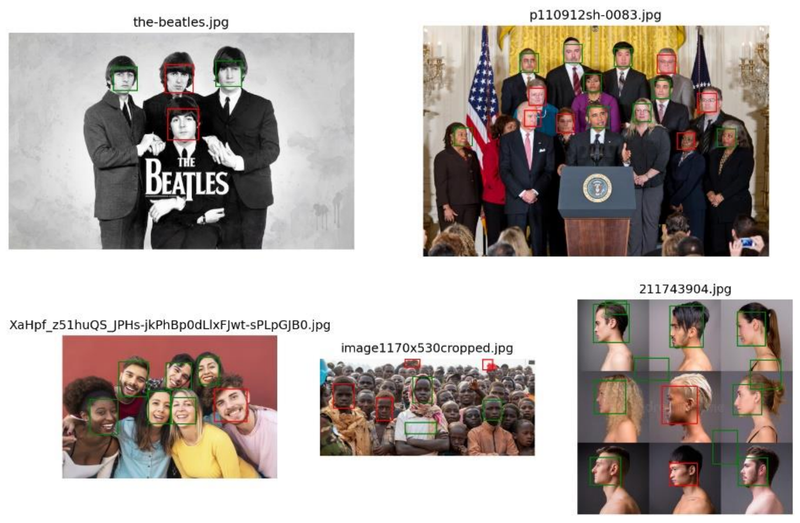
  - Method 1(face = 0, nonface = 1), T=10:



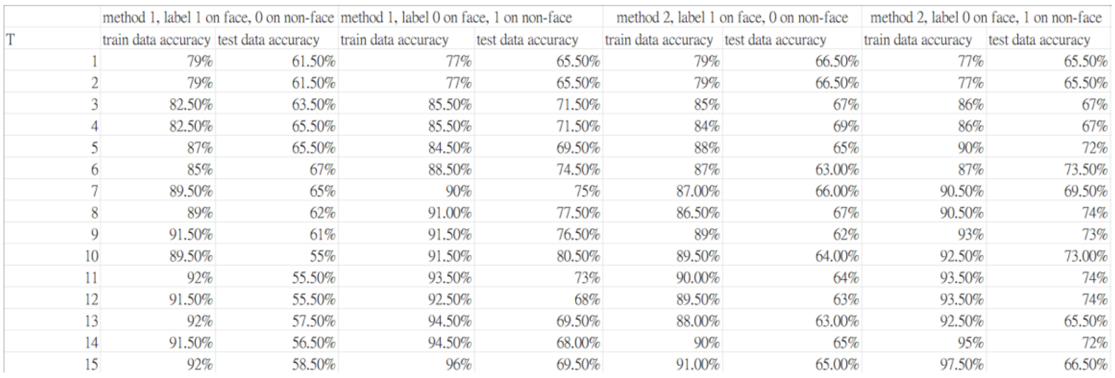
◆ Method 2(face = 1, nonface = 0), T = 4:



◆ Method 2(face = 0, nonface = 1), T = 8:

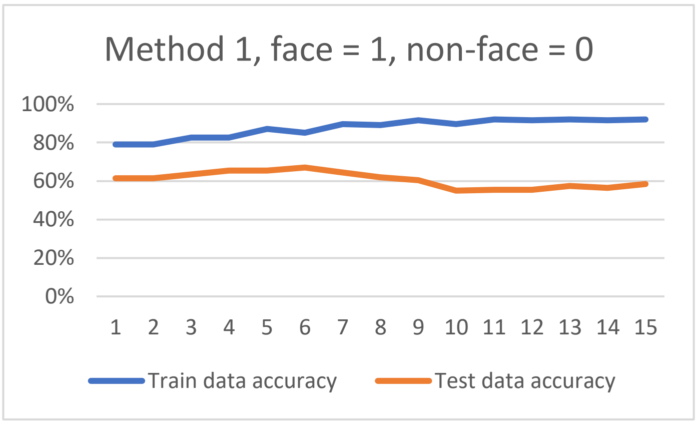


- My analysis or observation:
    - In order to fit the algorithm discussed in the lecture, the code I handed in is using method 1, T=6 (which is the best result in test data set), label face = 1, non-face = 0.
    - I used excel table to record each method, where T changes from 1 to 15:



	method 1, label 1 on face, 0 on non-face		method 1, label 0 on face, 1 on non-face		method 2, label 1 on face, 0 on non-face		method 2, label 0 on face, 1 on non-face	
T	train data accuracy	test data accuracy	train data accuracy	test data accuracy	train data accuracy	test data accuracy	train data accuracy	test data accuracy
1	79%	61.50%	77%	65.50%	79%	66.50%	77%	65.50%
2	79%	61.50%	77%	65.50%	79%	66.50%	77%	65.50%
3	82.50%	63.50%	85.50%	71.50%	85%	67%	86%	67%
4	82.50%	65.50%	85.50%	71.50%	84%	69%	86%	67%
5	87%	65.50%	84.50%	69.50%	88%	65%	90%	72%
6	85%	67%	88.50%	74.50%	87%	63.00%	87%	73.50%
7	89.50%	65%	90%	75%	87.00%	66.00%	90.50%	69.50%
8	89%	62%	91.00%	77.50%	86.50%	67%	90.50%	74%
9	91.50%	61%	91.50%	76.50%	89%	62%	93%	73%
10	89.50%	55%	91.50%	80.50%	89.50%	64.00%	92.50%	73.00%
11	92%	55.50%	93.50%	73%	90.00%	64%	93.50%	74%
12	91.50%	55.50%	92.50%	68%	89.50%	63%	93.50%	74%
13	92%	57.50%	94.50%	69.50%	88.00%	63.00%	92.50%	65.50%
14	91.50%	56.50%	94.50%	68.00%	90%	65%	95%	72%
15	92%	58.50%	96%	69.50%	91.00%	65.00%	97.50%	66.50%

    - Analysis about T:



I found the train data accuracy would gradually grow up, while test data accuracy would gradually grow up and fall down after T = 6.

In my opinion, the reason why the accuracy changes like this is due to overfitting problem we discussed in the lecture 3. That is, when I use a more complex method to train my model, the model will fit into the train data set, so when I applied it into the test data, it cannot work appropriately.

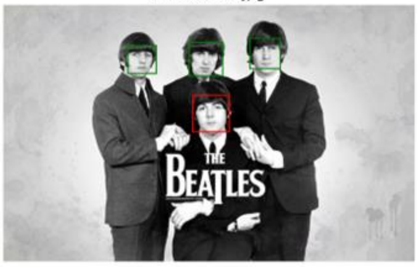
This result verified what the professor said. We should always train our model from the simple one and strike a balance between the complexity of the model to achieve the best result.

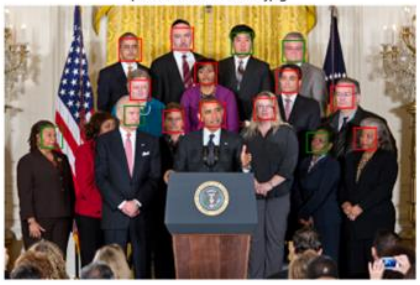
    - Results may change from the way we resize pictures:

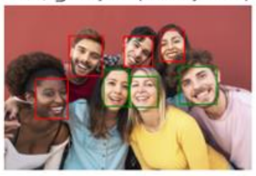
I found that if we change the interpretation in the detection.py, we may get the different result.

Noted that the following images are all conducted under the method 1, face=1, non-face=0, T = 10.

        - INTER_CUBIC:


the-beatles.jpg


p110912sh-0083.jpg


XaHpf_z51huQS_JPHs-jkPhBp0dLlxFJwt-sPLpGJB0.jpg

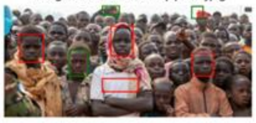
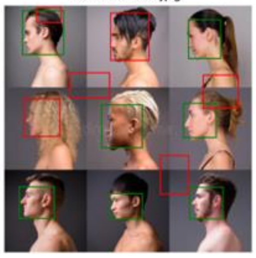
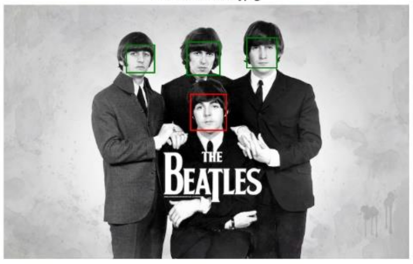
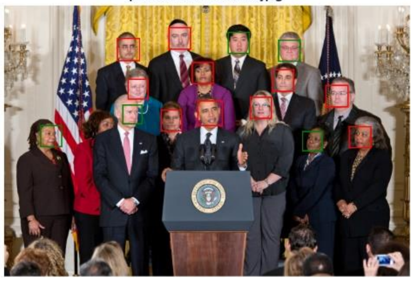


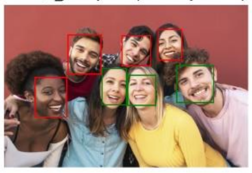
image1170x530cropped.jpg


211743904.jpg

◆ INTER_LINEAR:


the-beatles.jpg


p110912sh-0083.jpg


XaHpf_z51huQS_JPHs-jkPhBp0dLlxFJwt-sPLpGJB0.jpg

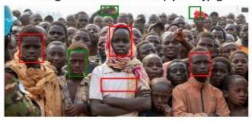
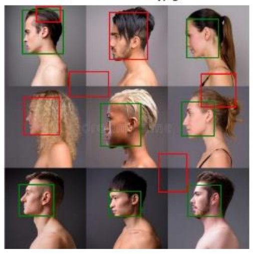
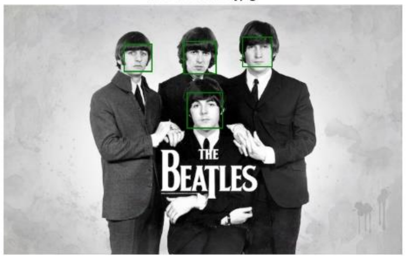
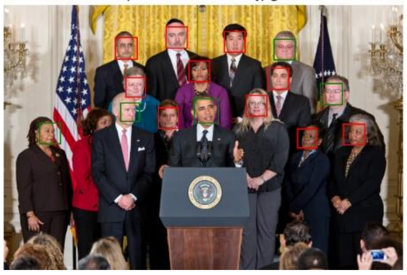


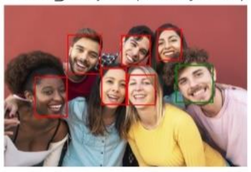
image1170x530cropped.jpg

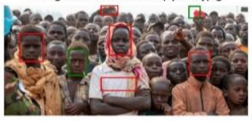

211743904.jpg

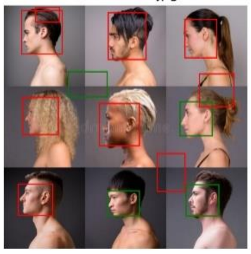
◆ INTER_NEAREST:


the-beatles.jpg

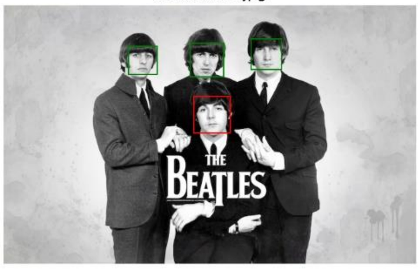

p110912sh-0083.jpg

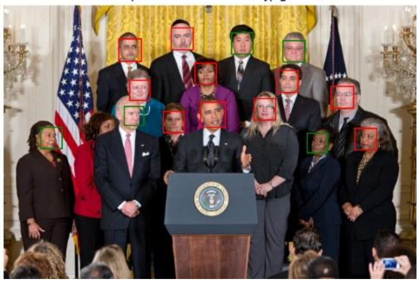


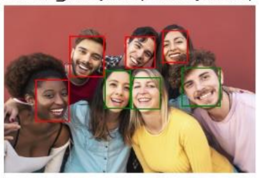


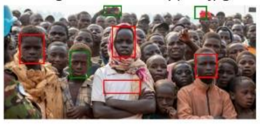


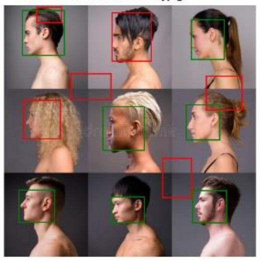
- FONT_HERSHEY_PLAIN:









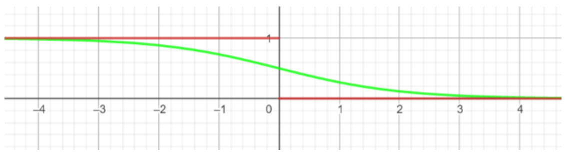


- My opinion: The reason is that the different interpretation will resize the selected region differently, so the classifier may get different results.

■ Compare method 1 and method 2:

The difference between method 1 and method 2 is only the way how we compute the h's value.

Method 1 is $h_j(x_i) = \begin{cases} 1 & f_j(x_i) < 0 \\ 0 & else \end{cases}$.



While method 2 is $g(x) = \dfrac{1}{1+e^x}$.

By looking at the right graph, where red line is method 1 and green line is method 2, we can found that method 2 changes not that intense when the feature values is at 0.

After analyze the table, when I label face = 1 and non-face = 0, the method 2 is generally better than method 1. While when I label face = 0 and non-face = 1, the method 2 is generally worse than method 1.

Part III. Answer the questions (12%):

1. Please describe a problem you encountered and how you solved it.
   - I found that in detection.py, if we only use *image = cv2.imread(f"data/detect/{filename}", flags=cv2.IMREAD_UNCHANGED)*, the image display on the screen will be in blue. After googling the problem, I found that the cv2.imread is default using BGR mode instead of the RGB mode. Furthermore, I found an discussion rgb-image-display-in-matplotlib-plt-imshow-returns-a-blue-image in stack overflow, and I used his method, which is put *image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)*. The method makes image displayed on the screen look perfectly.
2. What are the limitations of the Viola-Jones' algorithm?
   1. Sensitivity to lighting conditions: The Viola-Jones algorithm relies on analyzing the contrast between different regions of an image to detect objects. This means that it can be sensitive to changes in lighting conditions, which can affect the contrast and make it more difficult to detect objects accurately.

   2. Limited accuracy for non-frontal faces: As the professor discussed in the lecture, the algorithm was designed to detect frontal faces, and it may not be as accurate for non-frontal faces. This can be a limitation in applications where accurate detection of non-frontal faces is important.
3. Based on Viola-Jones' algorithm, how to improve the accuracy except changing the training dataset and parameter T?
   1. Use more complex features: The Viola-Jones algorithm uses Haar-like features to detect objects, but we can use more complex features to improve the accuracy of face detection.
   2. Use multiple detectors: Instead of relying on a single detector detect the whole face, multiple detectors can be trained to detect different parts of the object being detected. For example, in face detection, one detector can be trained to detect the eyes, another to detect the nose, and another to detect the mouth. Combining the output of these detectors can improve the

accuracy of the overall detection.

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.

- We may try Convolutional Neural Networks discussed in the lecture.
  - Convolutional Neural Networks
    - Pros
      - High accuracy: CNNs have shown to achieve state-of-the-art accuracy in object detection tasks, including face detection, due to their ability to learn and represent complex features of images.
      - Can handle variability in image conditions: CNNs can handle variability in image conditions, such as changes in lighting, pose, and occlusion, due to their ability to learn and represent different features at different scales.
      - End-to-end training: CNNs can be trained end-to-end, meaning that all the parameters of the network can be optimized together. This leads to better optimization and less need for manual tuning
    - Cons
      - Computationally expensive: CNNs require large amounts of computational resources, making them less suitable for real-time applications on low-power devices.
      - Require large amounts of training data: CNNs require large amounts of training data to achieve high accuracy, which can be a limitation in certain applications.
      - Black box model: CNNs can be difficult to interpret, making it hard to understand how the model arrives at its decision.
  - Adaboost
    - Pros
      - Efficient: Adaboost is computationally efficient, requiring fewer computational resources compared to CNNs.
      - Require small set of training data: Adaboost can achieve high accuracy with relatively small amounts of training data.
      - Easy to interpret: Adaboost is a relatively simple algorithm, making it easy to interpret and understand how the model arrives at its decision.
    - Cons
      - Limited to simple features: Adaboost is limited to simple features, such as Haar-like features, making it less effective in handling

complex variations in image conditions.

- Sensitive to noise: Adaboost can be sensitive to noise in the training data, which can affect its accuracy.
- May require manual tuning: Adaboost requires manual tuning of the T parameter, which can be time-consuming and require expertise to set correctly.