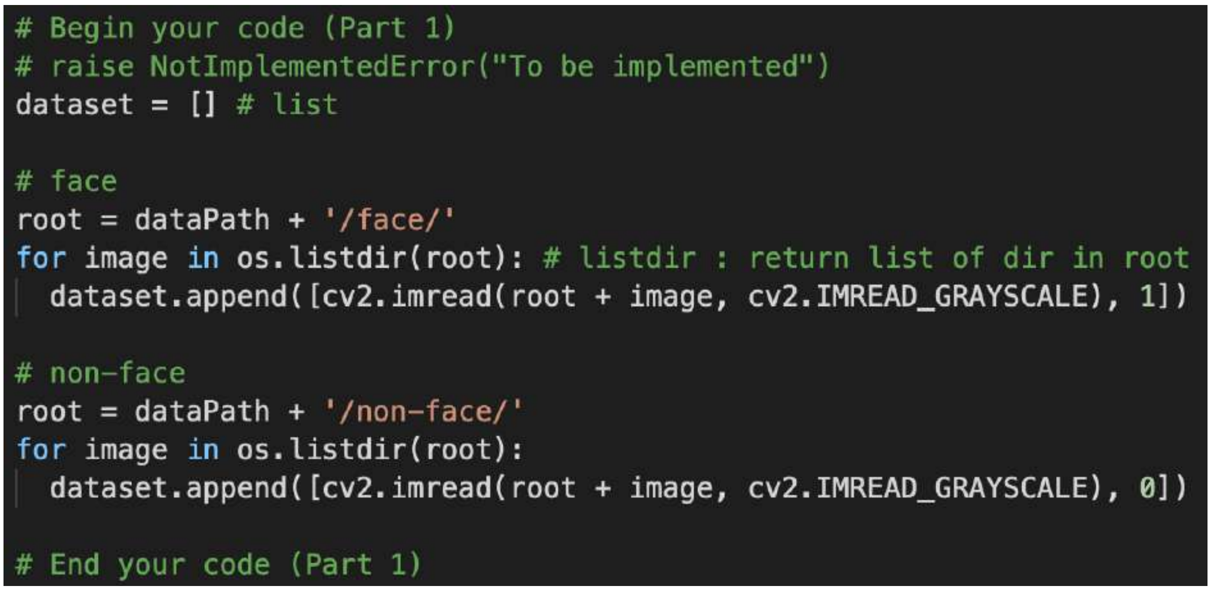
# Homework 1: Face Detection Report

110550107 傅莉妮

## Part I. Implementation (6%):

### Part 1

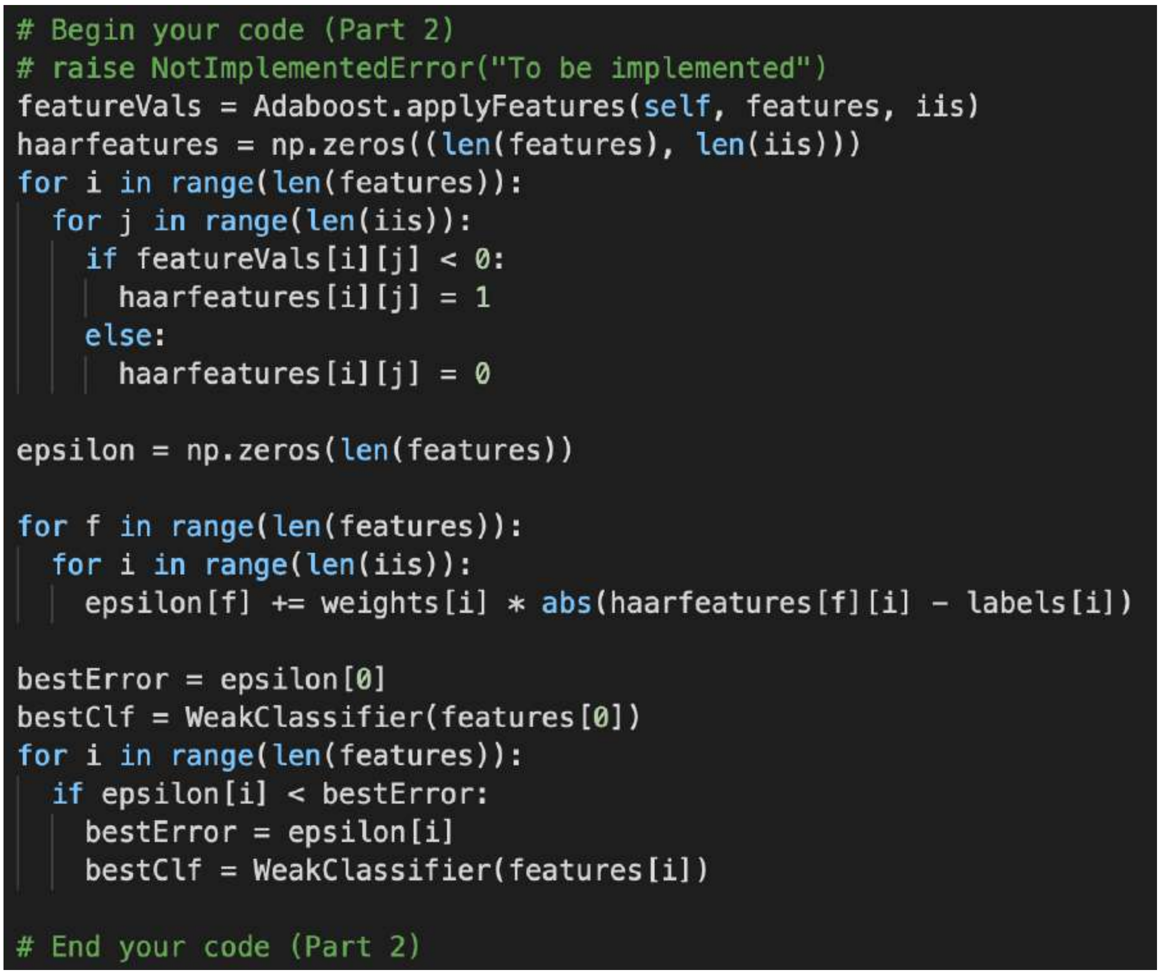


```python
# Begin your code (Part 1)
# raise NotImplementedError("To be implemented")
dataset = [] # list

# face
root = dataPath + '/face/'
for image in os.listdir(root): # listdir : return list of dir in root
  dataset.append([cv2.imread(root + image, cv2.IMREAD_GRAYSCALE), 1])

# non-face
root = dataPath + '/non-face/'
for image in os.listdir(root):
  dataset.append([cv2.imread(root + image, cv2.IMREAD_GRAYSCALE), 0])

# End your code (Part 1)
```

I create a list named dataset to save all the data of images. After that, I make a root to /face/ and /non-face/ folder under given dataPath, trace the list of directories in root, and use cv2.imread and IMREAD_GRAYSCALE to get all the infomation of each image.

After running this function, it will return a list with all the data and label of whether it is face(1) or not(0).

### Part 2

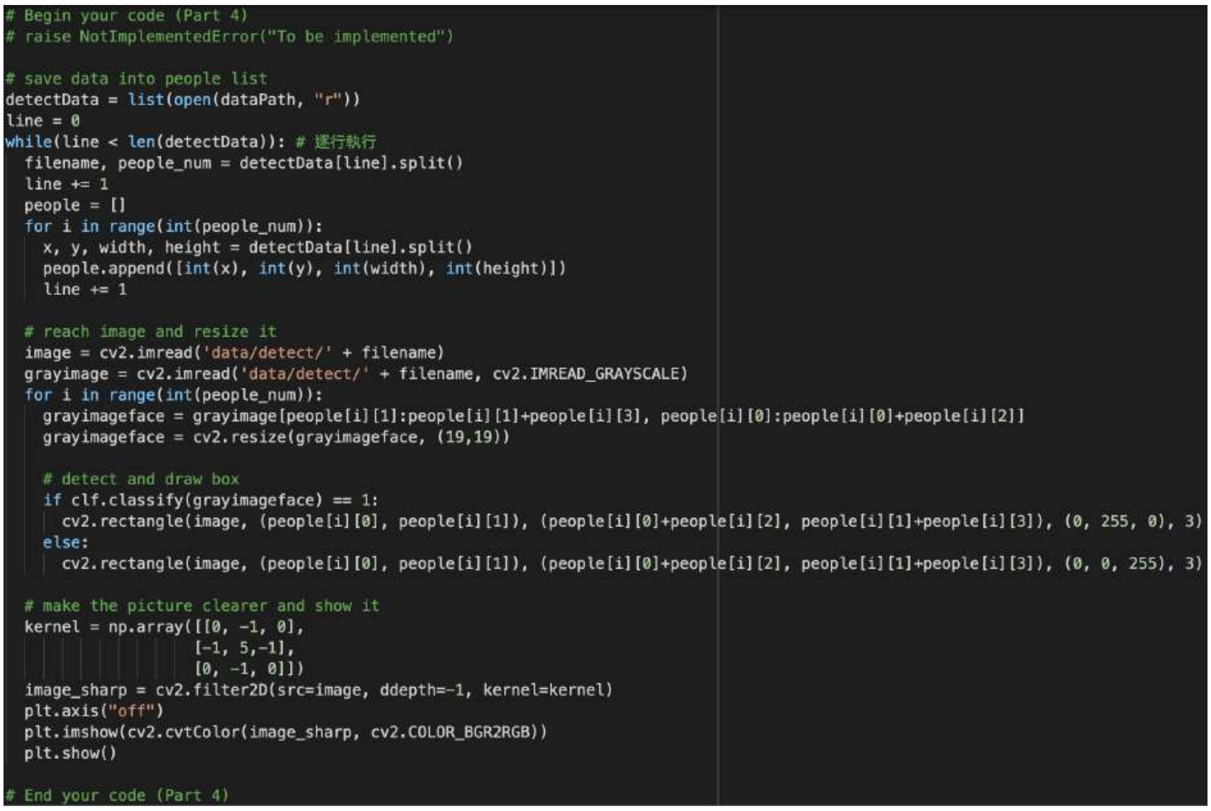


```python
# Begin your code (Part 2)
# raise NotImplementedError("To be implemented")
featureVals = Adaboost.applyFeatures(self, features, iis)
haarfeatures = np.zeros((len(features), len(iis)))
for i in range(len(features)):
    for j in range(len(iis)):
        if featureVals[i][j] < 0:
            haarfeatures[i][j] = 1
        else:
            haarfeatures[i][j] = 0

epsilon = np.zeros(len(features))

for f in range(len(features)):
    for i in range(len(iis)):
        epsilon[f] += weights[i] * abs(haarfeatures[f][i] - labels[i])

bestError = epsilon[0]
bestClf = WeakClassifier(features[0])
for i in range(len(features)):
    if epsilon[i] < bestError:
        bestError = epsilon[i]
        bestClf = WeakClassifier(features[i])

# End your code (Part 2)
```

I did this part following the instructions of Adaboost algorithm. First, I use Adaboost.applyFeatures() to calculate all the values from feature function. I create a 2D array named haarfeatures to save the value of classifier. The classifier is that if its featureVals < 0 then haarfeatures = 1, otherwise haarfeatures = 0. I calculate epsilon using the same way as professor mentioned in the lecture.

After calculating all the values we want to know, it chooses bestError and bestClassifier by comparing their epsilons and returns them.

## Part 4



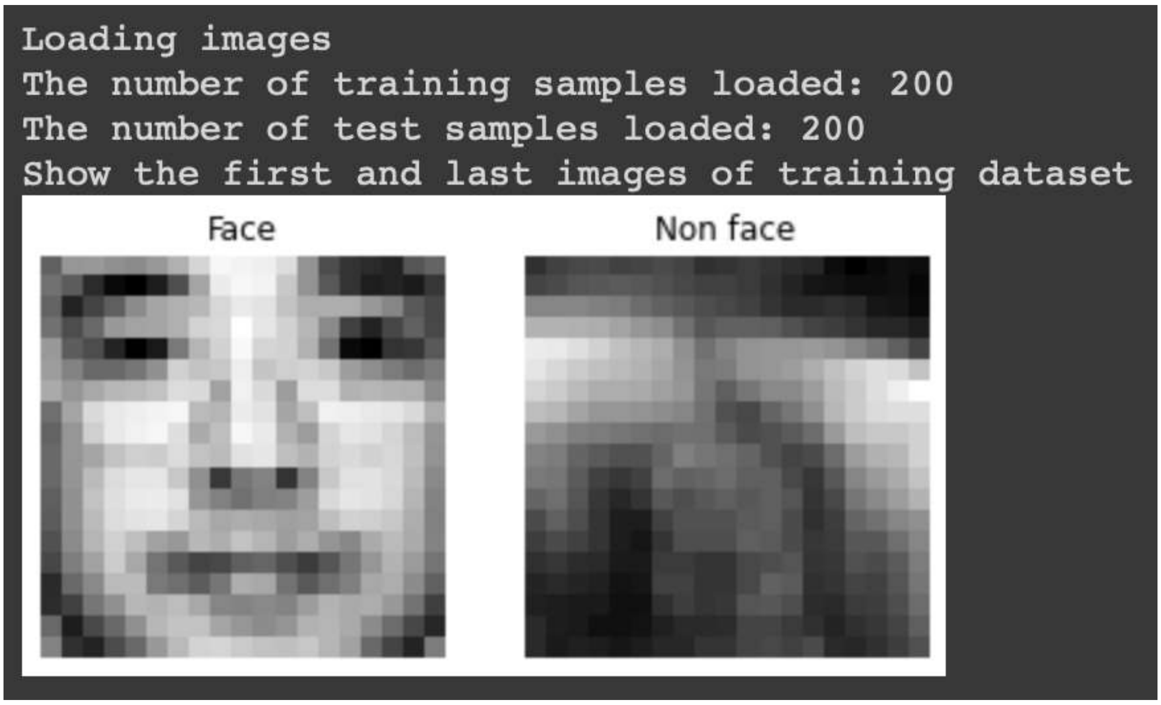
```python
# Begin your code (Part 4)
# raise NotImplementedError("To be implemented")

# save data into people list
detectData = list(open(dataPath, "r"))
line = 0
while(line < len(detectData)): # 逐行執行
  filename, people_num = detectData[line].split()
  line += 1
  people = []
  for i in range(int(people_num)):
    x, y, width, height = detectData[line].split()
    people.append([int(x), int(y), int(width), int(height)])
    line += 1

  # reach image and resize it
  image = cv2.imread('data/detect/' + filename)
  grayimage = cv2.imread('data/detect/' + filename, cv2.IMREAD_GRAYSCALE)
  for i in range(int(people_num)):
    grayimageface = grayimage[people[i][1]:people[i][1]+people[i][3], people[i][0]:people[i][0]+people[i][2]]
    grayimageface = cv2.resize(grayimageface, (19,19))

    # detect and draw box
    if clf.classify(grayimageface) == 1:
      cv2.rectangle(image, (people[i][0], people[i][1]), (people[i][0]+people[i][2], people[i][1]+people[i][3]), (0, 255, 0), 3)
    else:
      cv2.rectangle(image, (people[i][0], people[i][1]), (people[i][0]+people[i][2], people[i][1]+people[i][3]), (0, 0, 255), 3)

  # make the picture clearer and show it
  kernel = np.array([[0, -1, 0],
                     [-1, 5,-1],
                     [0, -1, 0]])
  image_sharp = cv2.filter2D(src=image, ddepth=-1, kernel=kernel)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(image_sharp, cv2.COLOR_BGR2RGB))
  plt.show()

# End your code (Part 4)
```

In part 4, I save all the data we want to detect into a list first. I create a list of list named people, which contains the data of all the faces in the pictures. after getting this list, I use the same way in part 1 to reach images. I load an original image and a grayscale image respectively. the original one is to mark whether they are faces and to show in the last. For grayscale images, I resize them into 19x19 and use clf.classify() to check them and draw the boxes on the original image.

## Part II. Results & Analysis (12%):

## Results

### Part 1:



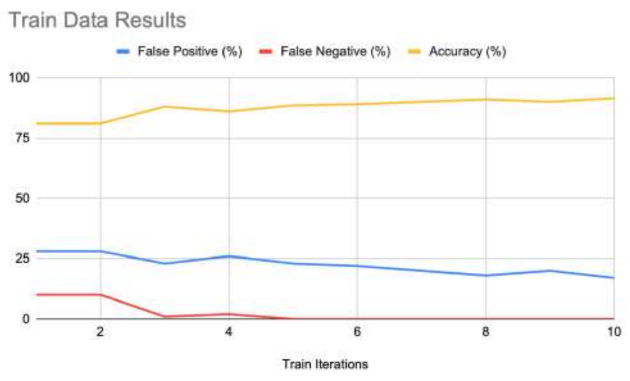
```
Loading images
The number of training samples loaded: 200
The number of test samples loaded: 200
Show the first and last images of training dataset
```

## Part 2:

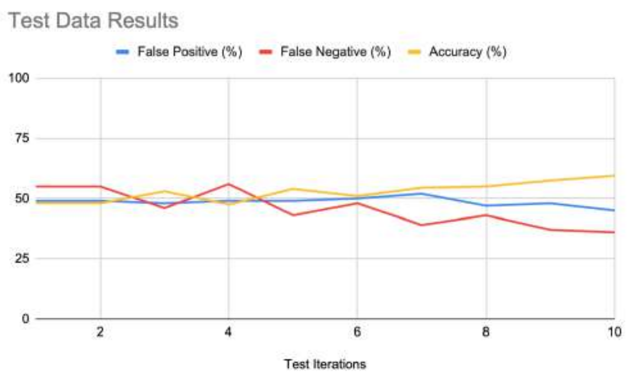
```
Start training your classifier
Computing integral images
Building features
Applying features to dataset
Selecting best features
Selected 5171 potential features
Initialize weights
Run No. of Iteration: 1
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[R
ectangleRegion(8, 0, 1, 3), RectangleRegion(7, 3, 1, 3)], negative regions=[RectangleR
egion(7, 0, 1, 3), RectangleRegion(8, 3, 1, 3)]) with accuracy: 162.000000 and alpha:
 1.450010

Evaluate your classifier with training dataset
False Positive Rate: 28/100 (0.280000)
False Negative Rate: 10/100 (0.100000)
Accuracy: 162/200 (0.810000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 55/100 (0.550000)
Accuracy: 96/200 (0.480000)
```

## Part 3:

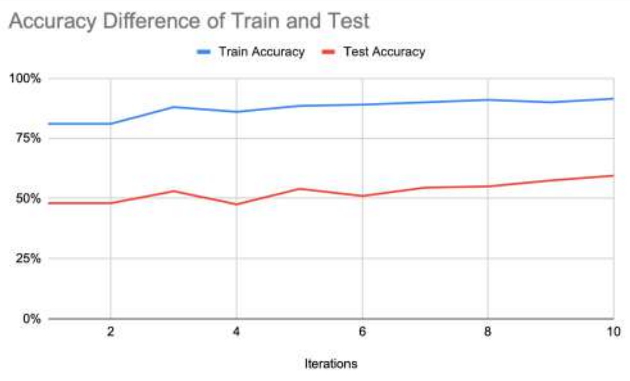
| Train: | False Positive | False Negative | Accuracy |
|---|---|---|---|
| 1 | 28 | 10 | 81% |
| 2 | 28 | 10 | 81% |
| 3 | 23 | 1 | 88% |
| 4 | 26 | 2 | 86% |
| 5 | 23 | 0 | 88.5% |
| 6 | 22 | 0 | 89% |
| 7 | 20 | 0 | 90% |
| 8 | 18 | 0 | 91% |
| 9 | 20 | 0 | 90% |
| 10 | 17 | 0 | 91.5% |



| Test: | False Positive | False Negative | Accuracy |
|---|---|---|---|
| 1 | 49 | 55 | 48% |
| 2 | 49 | 55 | 48% |
| 3 | 48 | 46 | 53% |
| 4 | 49 | 56 | 47.5% |
| 5 | 49 | 43 | 54% |
| 6 | 50 | 48 | 51% |
| 7 | 52 | 39 | 54.5% |
| 8 | 47 | 43 | 55% |
| 9 | 48 | 37 | 57.5% |
| 10 | 45 | 36 | 59.5% |



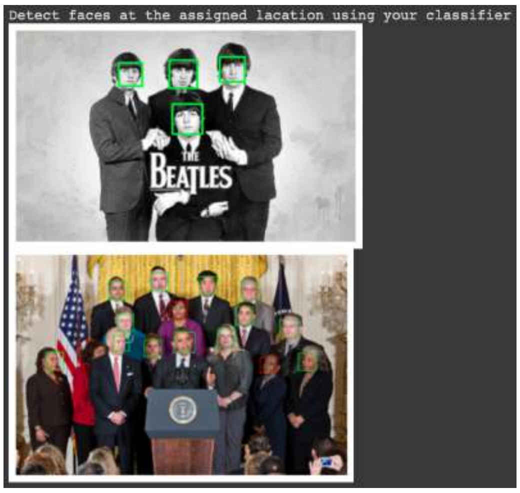
| Iterations | Train Accuracy | Test Accuracy |
|---|---|---|
| 1 | 81% | 48% |
| 2 | 81% | 48% |
| 3 | 88% | 53% |
| 4 | 86% | 47.5% |
| 5 | 88.5% | 54% |
| 6 | 89% | 51% |
| 7 | 90% | 54.5% |
| 8 | 91% | 55% |
| 9 | 90% | 57.5% |
| 10 | 91.5% | 59.5% |



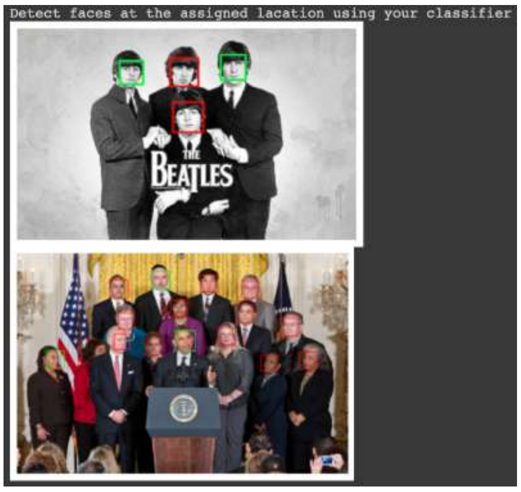
In the line chart above, we can observe that for both Train and Test data, the accuracy is rising slowly while the iterations increase. Train data results is far more accurate than test data results, the difference between two of their accuracy is 32% when T=10. Their difference didn't change much for each iteration.

## Part 4:

- T=1:



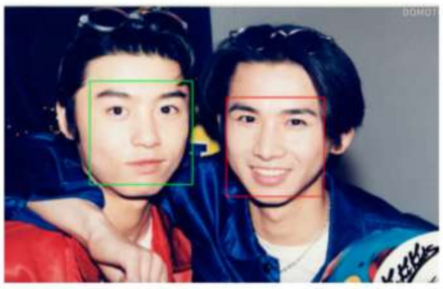
- T=10:



## Part 5:

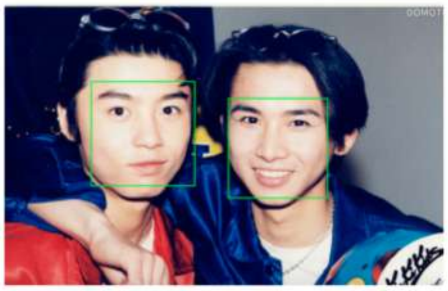
(I added nana.png with 3 additional arbitrary non-face areas to check if it will work well on non-face detection)

- T=1:
  - kinkikids.jpg

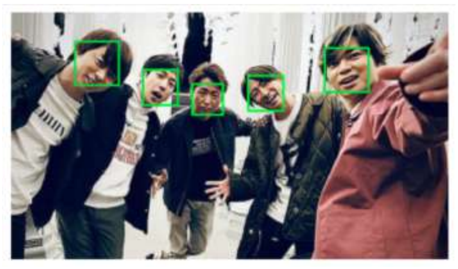


  - arashi.jpg

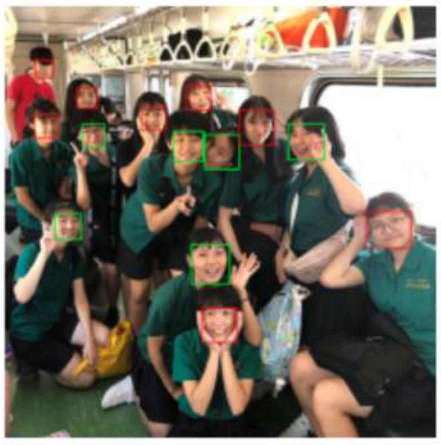
- T=10:
  - kinkikids.jpg



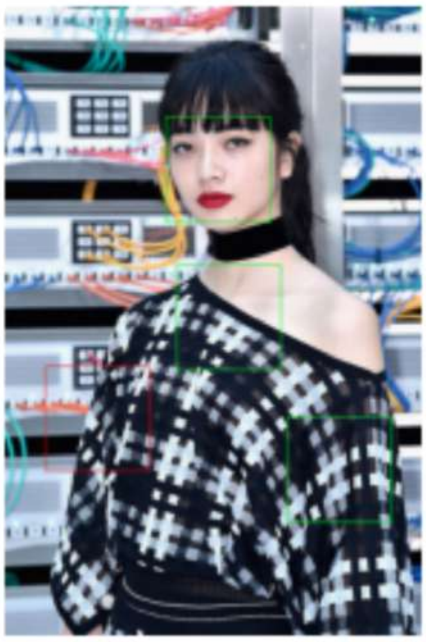
  - arashi.jpg

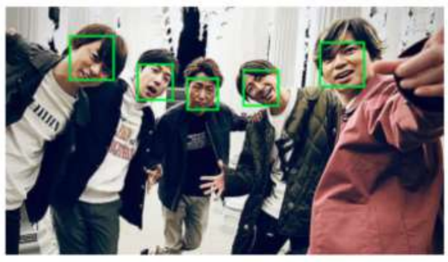


- highschool.JPG

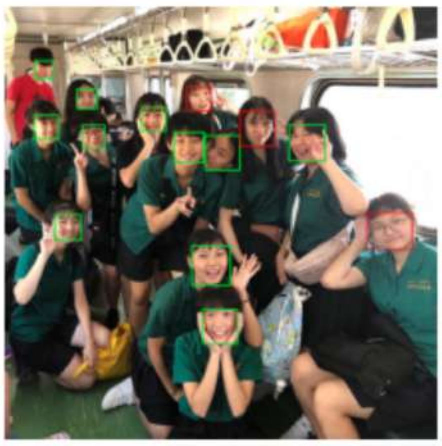


- nana.png

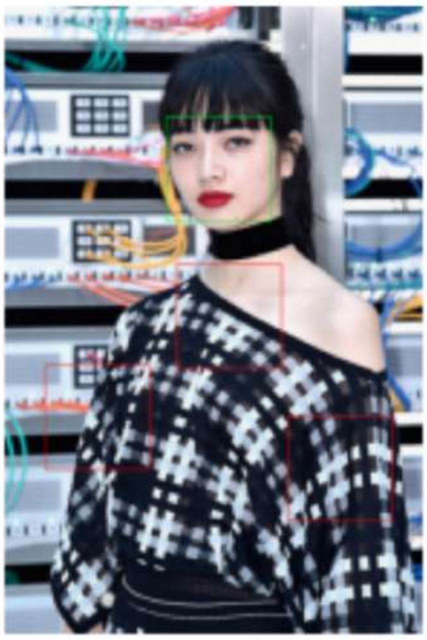




- highschool.JPG



- nana.png

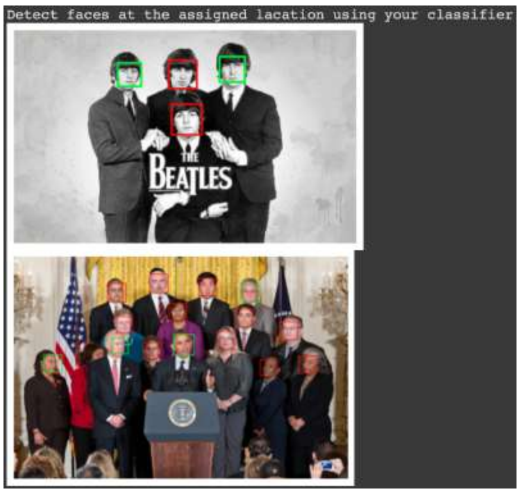


## Analysis

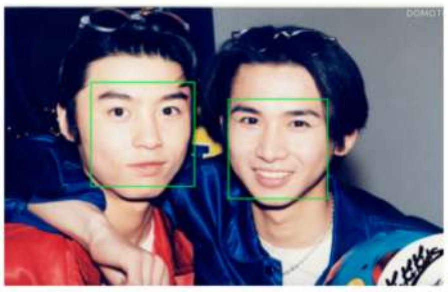
In part 4, it is much more accurate when T=1, for 17 correctly detected and 2 wrongly detected as non-face, while 5 correct and 14 wrong for T=10. However, when I was testing my own images, I found that T=10 is more accurate than T=1 for 22 correct and 3 wrong in T=10 and 14 correct and 11 wrong in T=1.

After getting the results for T=1 and T=10, I expected that it will be more accurate for bigger T, therefore I tested for T=20 and T=30. However, it wasn't performing as my expection.

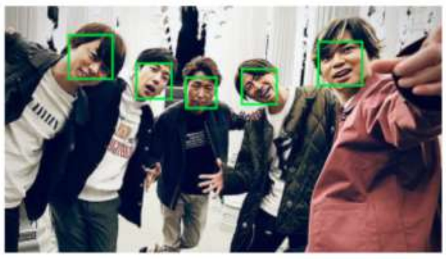
- T=20:



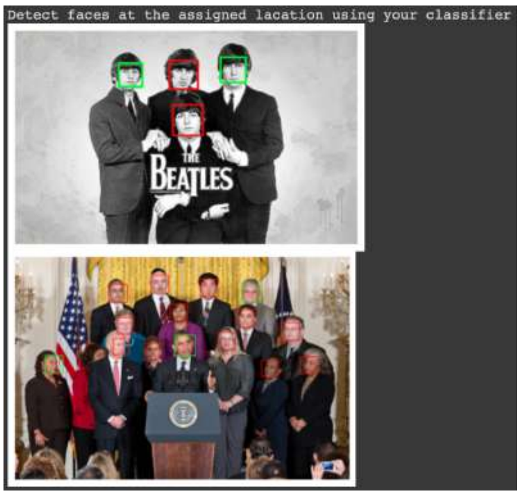
  - kinkikids.jpg



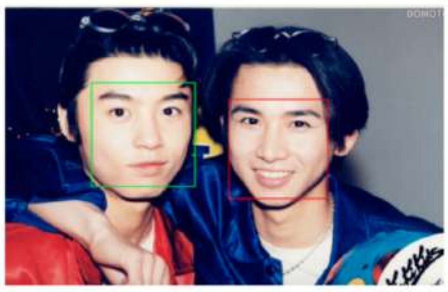
  - arashi.jpg



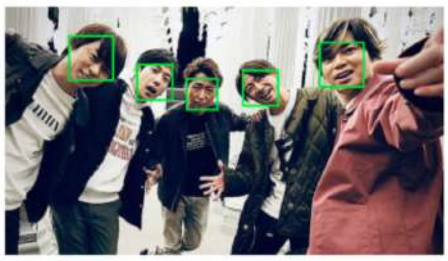
- T=30:



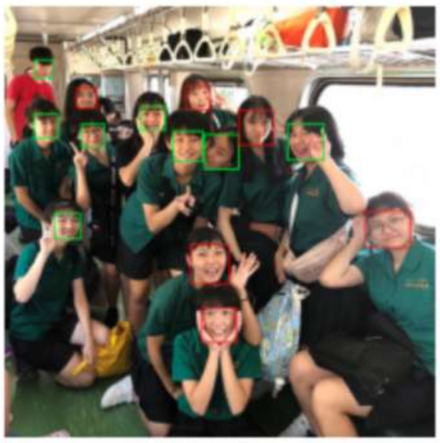
  - kinkikids.jpg



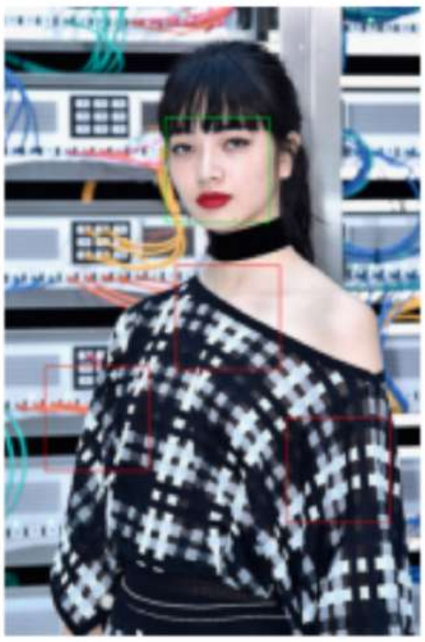
  - arashi.jpg



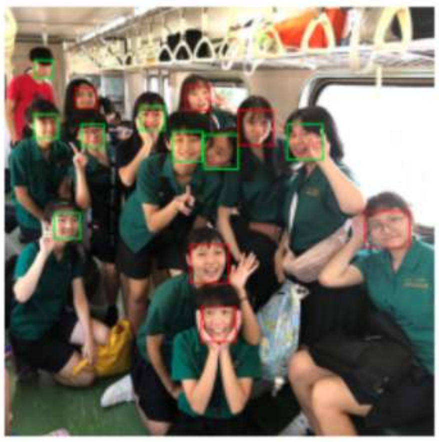
- highschool.JPG



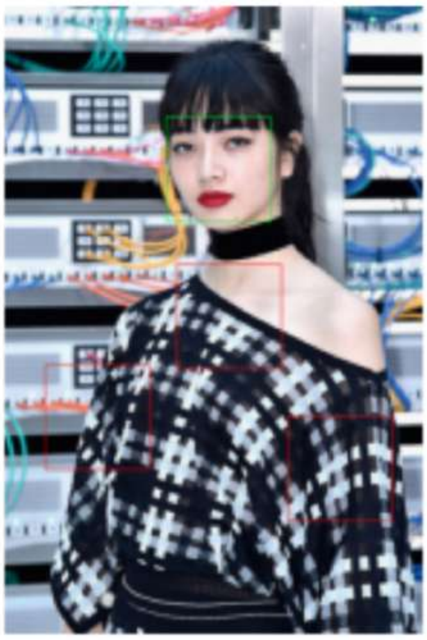
- nana.png



- highschool.JPG



- nana.png



| Iterations | Train Accuracy | Test Accuracy | Correctness (part 4) | Correctness (part 5) |
|---|---|---|---|---|
| 1 | 81% | 48% | 17/19 | 14/25 |
| 10 | 91.5% | 59.5% | 5/19 | 22/25 |
| 20 | 97% | 57.5% | 6/19 | 19/25 |
| 30 | 98.5% | 54% | 5/19 | 18/25 |

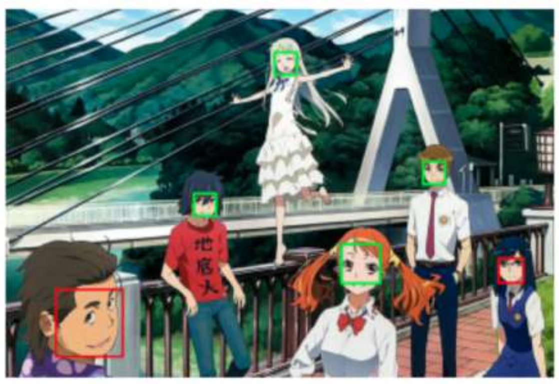
The train accuracy is getting better when T increases, but the test accuracy is not. The test accuracy is decreasing from T=10 to T=30.

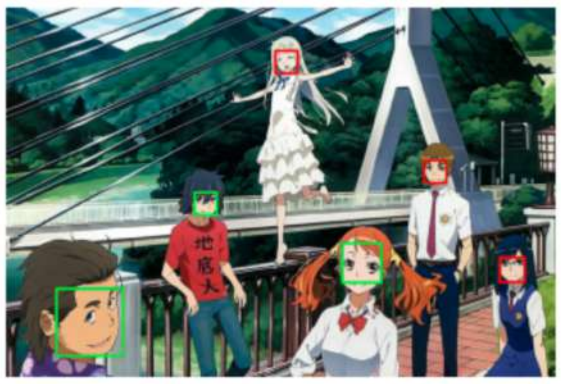
For the last image “nana.png”, it didn’t perfom well in T=1, for 2/4 correctness. However, it is 100% correct for T=10, 20, and 30.

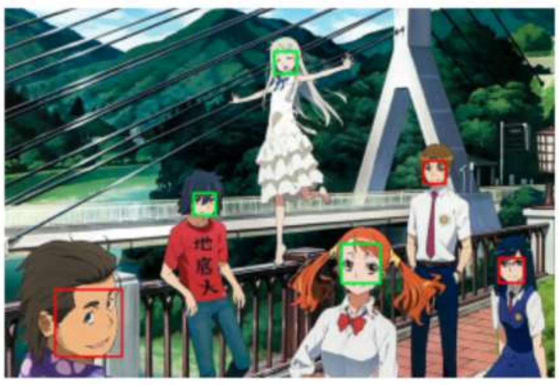
The correctness of T=20 and T=30 for part 4 is quite similar with T=10. However, both T=20 and 30 is worse than T=10 for the correctness for part 5. Although the train accuracy of T=30 is the best, the performance of T=30 for testing other images is worse than T=20 and T=10 overall.

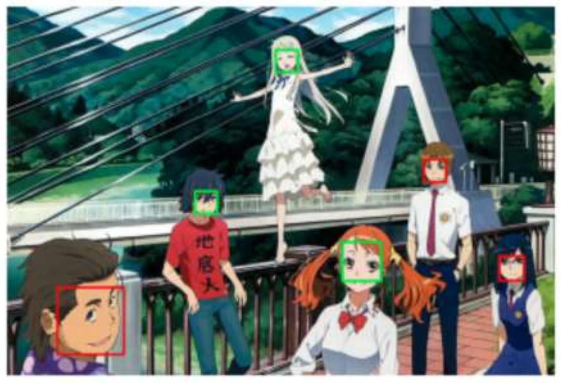
## Additional test for manga image:

I am curious about whether this detector can classify images without real face but only drawing, so I did this experiment additionally, just to fulfill my own curiouness. I think that the reason why it can classify some of the ‘faces’ is that haar-like features captures the black or white areas of each block, and sketch like this also captures only the main outlines of faces.


T=1


T=10


T=20


T=30

## Part III. Answer the questions (12%):

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

   I got pictures with totally opposite colors at first. Therefore I use cv2.COLOR_BGR2RGB to get the correct image. Besides, I got some pictures with terrible resolution at first, so I use Kernel to sharpen the images.

   One of the biggest problem I encountered is that I haven't learned python before, and I even don't know how to read image at first. I searched and learned every syntax when I need it. After working on this project, I understand python better and I am more confident on coding python now.

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

   a. light sensitive: Viola-Jones' algorithm is mainly to check the brightness of a grayscale image. Therefore it may be sensitive to high exposure.

   b. slow time: The training time is slow. It needs to run 5 minutes for T=30 on my device.

   c. orientation sensitive: In the 2 images given in the project, I found if the faces is turned, it may not be accurate much. This is even more obvious when iteration time increases.

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

   a. use more training dataset.

   b. choose a better classifier.

   c. use multiple classifiers. e.g.: one for frontal faces and the other one for tilted or turned faces.

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 one of the most important part of detection is shape. If we can come up with a method that can catch the contours, then we can detect almost everything. For example, we draw the outline of things when we want to do a sketch. I think the reason that we can draw outlines is that it has different colors with the others. However, how many outlines should we draw and how to ask a computer to draw them are the problems. One of the existing method is

Histogram of oriented gradient(HOG) algorithm. It normalizes the image and calculates its gradients to obtain their feature vectors. I think it is also an implementation of “catching outlines”.

The best thought I can come up with is to combine Adaboost algorithm and HOG algorithm. I want to create a detector with 2 or more classifiers. Since the property of Adaboost(Haar-like feature) is that it is sensitive to light, I think it may be able to catch the brightness of every part of a face. Besides that, I want to use HOG to catch the shapes of face. Because that Adaboost is only for fixed scale of a face, we can prepare more classifiers for tilted or turned faces.

pros:

1. More accurate than only use one algorithm
2. Adaptive to different scales of faces
3. Intuitive. Similar as humans’ actions to capture a face

cons:

1. Slower than both algorithm
2. Needs more datasets (diverse angle, tilted and turned faces) to train it