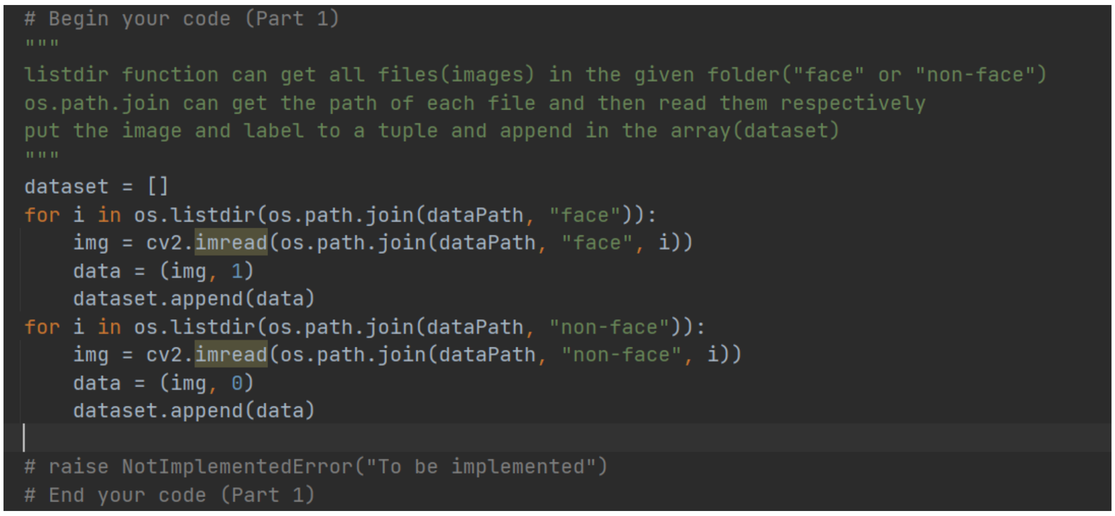
110550071 田松翰

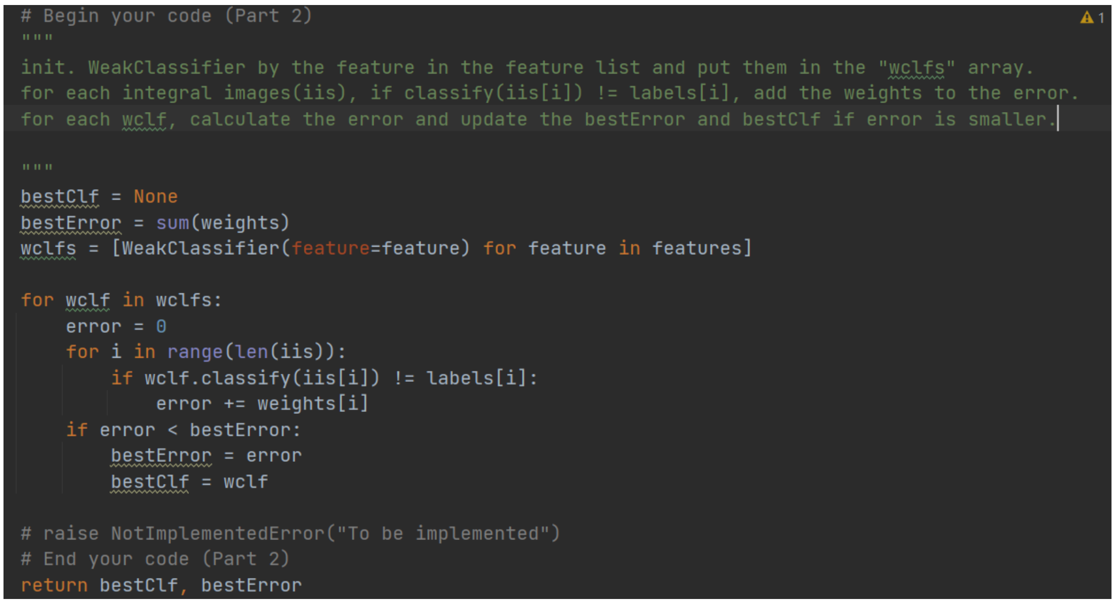
Part I: Implementation

Part 1:



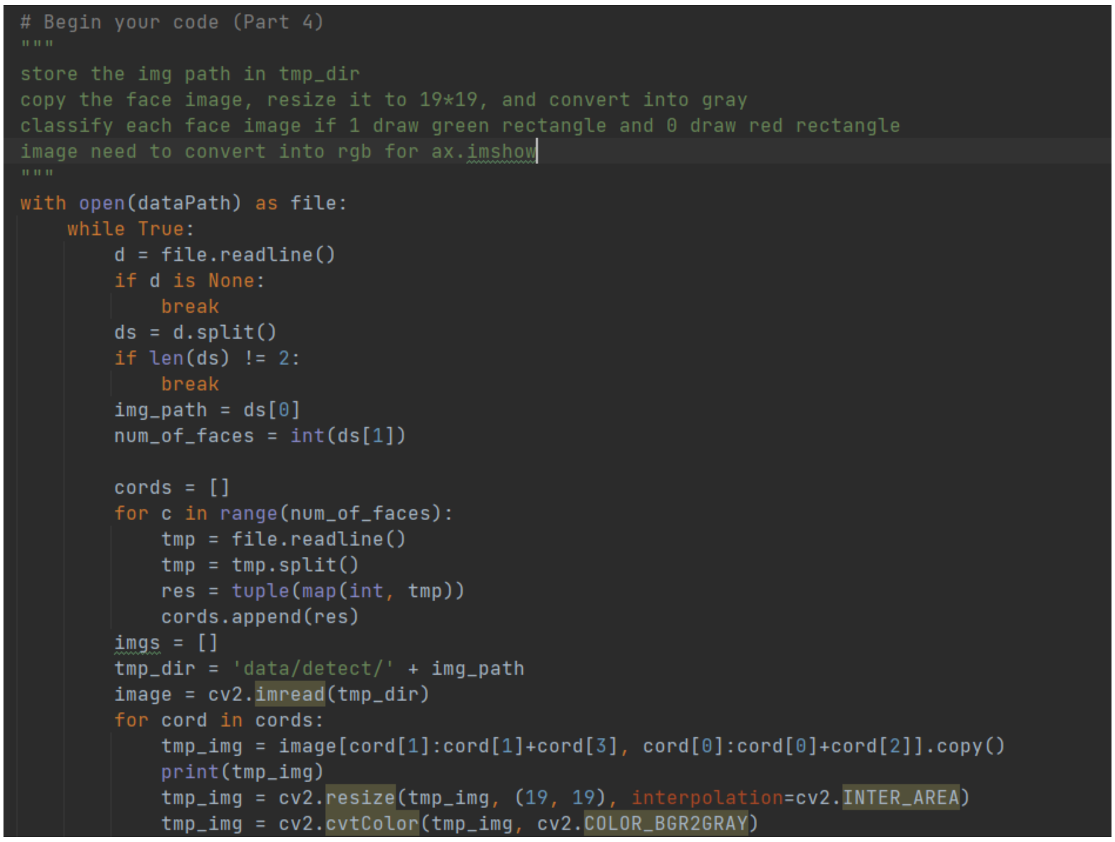
```python
# Begin your code (Part 1)
"""
listdir function can get all files(images) in the given folder("face" or "non-face")
os.path.join can get the path of each file and then read them respectively
put the image and label to a tuple and append in the array(dataset)
"""
dataset = []
for i in os.listdir(os.path.join(dataPath, "face")):
    img = cv2.imread(os.path.join(dataPath, "face", i))
    data = (img, 1)
    dataset.append(data)
for i in os.listdir(os.path.join(dataPath, "non-face")):
    img = cv2.imread(os.path.join(dataPath, "non-face", i))
    data = (img, 0)
    dataset.append(data)

# raise NotImplementedError("To be implemented")
# End your code (Part 1)
```

Part 2:

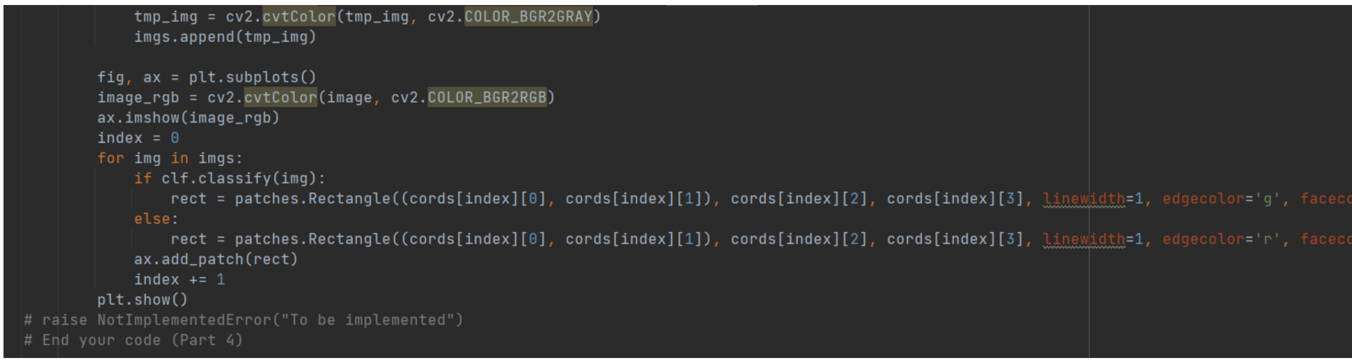


```python
# Begin your code (Part 2)
"""
init. WeakClassifier by the feature in the feature list and put them in the "wclfs" array.
for each integral images(iis), if classify(iis[i]) != labels[i], add the weights to the error.
for each wclf, calculate the error and update the bestError and bestClf if error is smaller.

"""
bestClf = None
bestError = sum(weights)
wclfs = [WeakClassifier(feature=feature) for feature in features]

for wclf in wclfs:
    error = 0
    for i in range(len(iis)):
        if wclf.classify(iis[i]) != labels[i]:
            error += weights[i]
    if error < bestError:
        bestError = error
        bestClf = wclf

# raise NotImplementedError("To be implemented")
# End your code (Part 2)
return bestClf, bestError
```

Part 4:

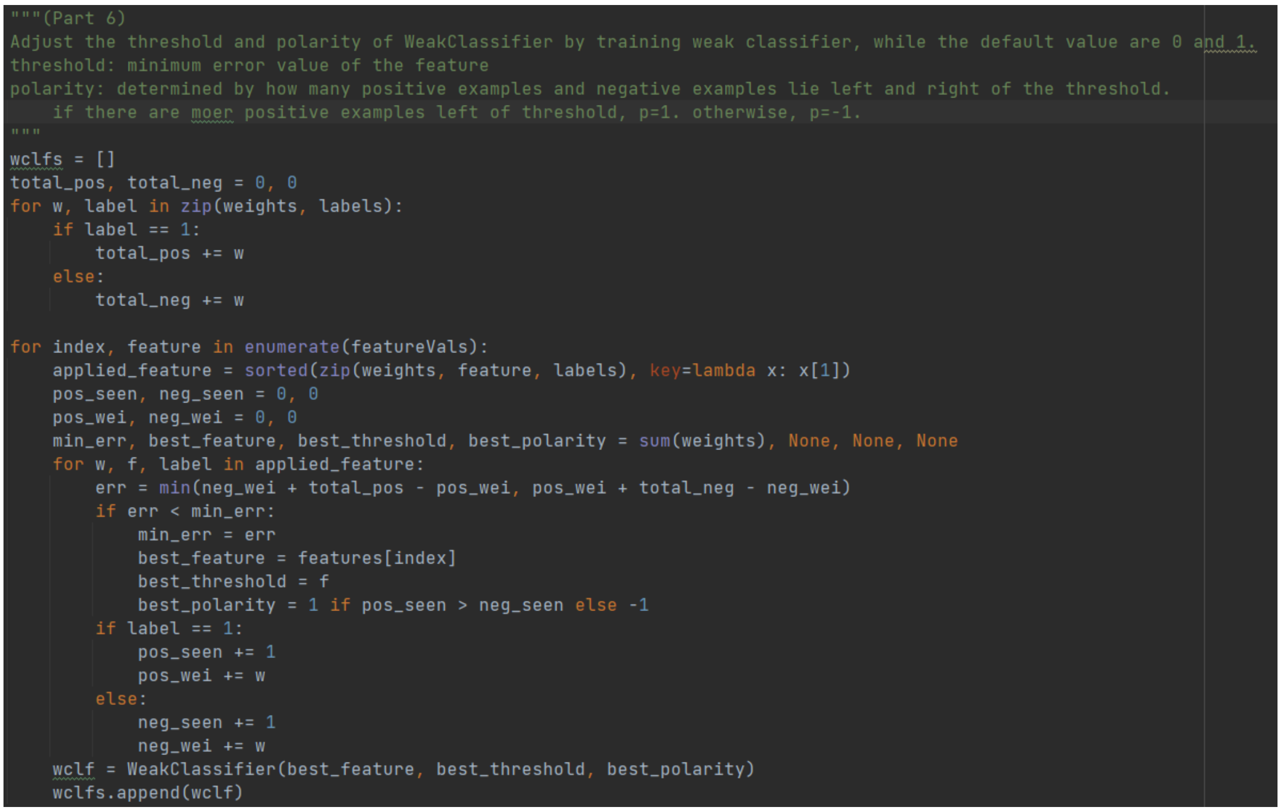


```python
# Begin your code (Part 4)
"""
store the img path in tmp_dir
copy the face image, resize it to 19*19, and convert into gray
classify each face image if 1 draw green rectangle and 0 draw red rectangle
image need to convert into rgb for ax.imshow
"""
with open(dataPath) as file:
    while True:
        d = file.readline()
        if d is None:
            break
        ds = d.split()
        if len(ds) != 2:
            break
        img_path = ds[0]
        num_of_faces = int(ds[1])

        cords = []
        for c in range(num_of_faces):
            tmp = file.readline()
            tmp = tmp.split()
            res = tuple(map(int, tmp))
            cords.append(res)
        imgs = []
        tmp_dir = 'data/detect/' + img_path
        image = cv2.imread(tmp_dir)
        for cord in cords:
            tmp_img = image[cord[1]:cord[1]+cord[3], cord[0]:cord[0]+cord[2]].copy()
            print(tmp_img)
            tmp_img = cv2.resize(tmp_img, (19, 19), interpolation=cv2.INTER_AREA)
            tmp_img = cv2.cvtColor(tmp_img, cv2.COLOR_BGR2GRAY)
```



```python
            tmp_img = cv2.cvtColor(tmp_img, cv2.COLOR_BGR2GRAY)
            imgs.append(tmp_img)

        fig, ax = plt.subplots()
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        ax.imshow(image_rgb)
        index = 0
        for img in imgs:
            if clf.classify(img):
                rect = patches.Rectangle((cords[index][0], cords[index][1]), cords[index][2], cords[index][3], linewidth=1, edgecolor='g', facec
            else:
                rect = patches.Rectangle((cords[index][0], cords[index][1]), cords[index][2], cords[index][3], linewidth=1, edgecolor='r', facec
            ax.add_patch(rect)
            index += 1
        plt.show()
# raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

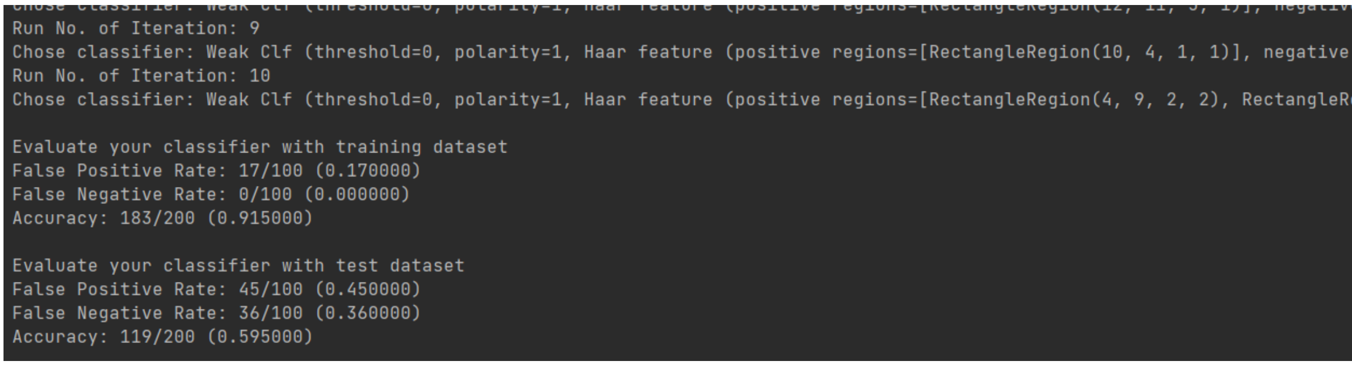
Part 6:



```python
"""(Part 6)
Adjust the threshold and polarity of WeakClassifier by training weak classifier, while the default value are 0 and 1.
threshold: minimum error value of the feature
polarity: determined by how many positive examples and negative examples lie left and right of the threshold.
    if there are moer positive examples left of threshold, p=1. otherwise, p=-1.
"""
wclfs = []
total_pos, total_neg = 0, 0
for w, label in zip(weights, labels):
    if label == 1:
        total_pos += w
    else:
        total_neg += w

for index, feature in enumerate(featureVals):
    applied_feature = sorted(zip(weights, feature, labels), key=lambda x: x[1])
    pos_seen, neg_seen = 0, 0
    pos_wei, neg_wei = 0, 0
    min_err, best_feature, best_threshold, best_polarity = sum(weights), None, None, None
    for w, f, label in applied_feature:
        err = min(neg_wei + total_pos - pos_wei, pos_wei + total_neg - neg_wei)
        if err < min_err:
            min_err = err
            best_feature = features[index]
            best_threshold = f
            best_polarity = 1 if pos_seen > neg_seen else -1
        if label == 1:
            pos_seen += 1
            pos_wei += w
        else:
            neg_seen += 1
            neg_wei += w
    wclf = WeakClassifier(best_feature, best_threshold, best_polarity)
    wclfs.append(wclf)
```

Part II: Result & Analysis

Result:



```
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(12, 11, 3, 1)], negativ
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 4, 1, 1)], negative
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 9, 2, 2), RectangleRe

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)
```

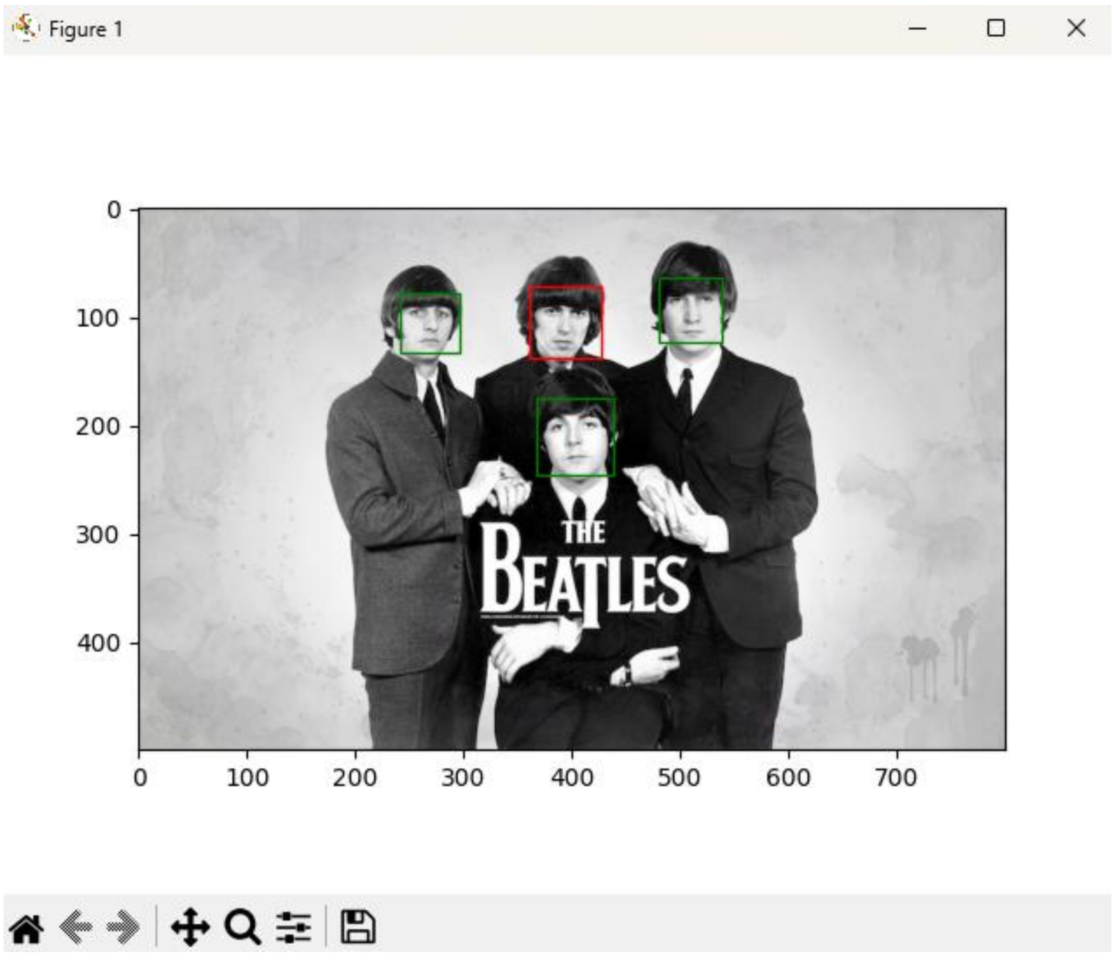


Figure 1
0
100
200
300
400
0
100
200
300
400
500
600
700
THE
BEATLES

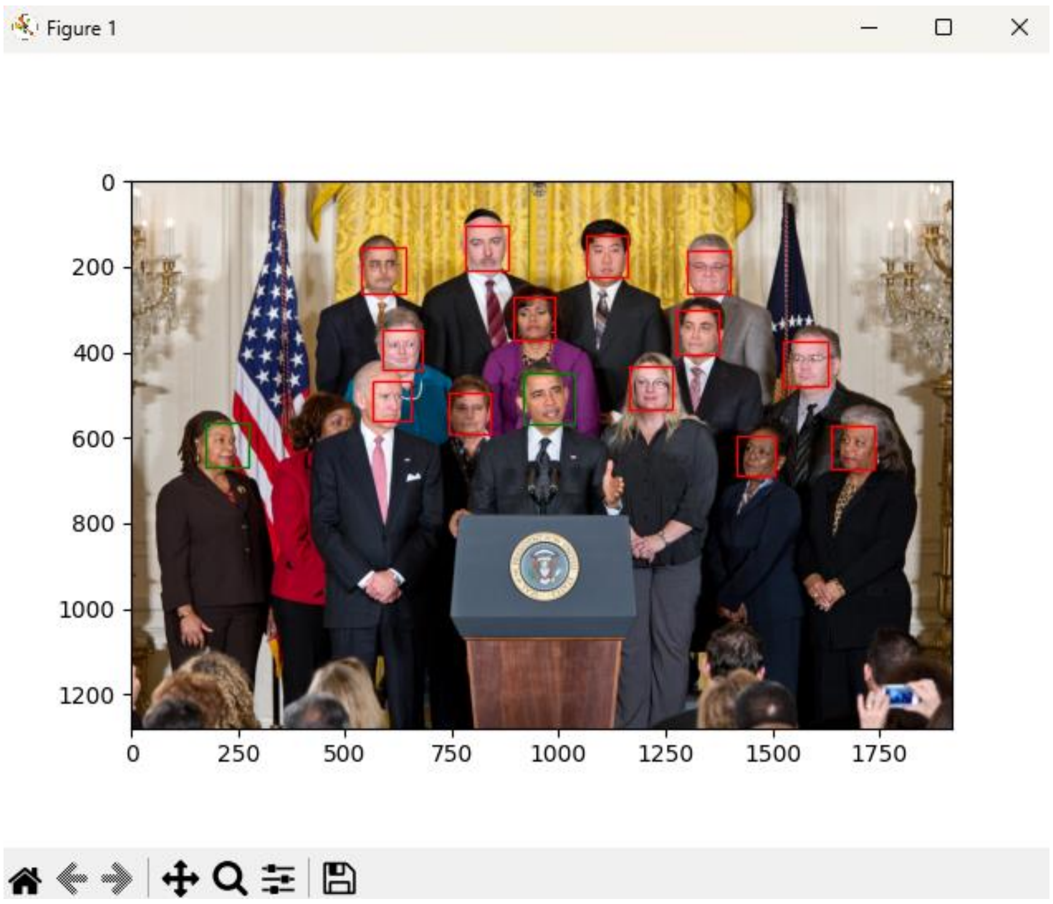


Figure 1
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750

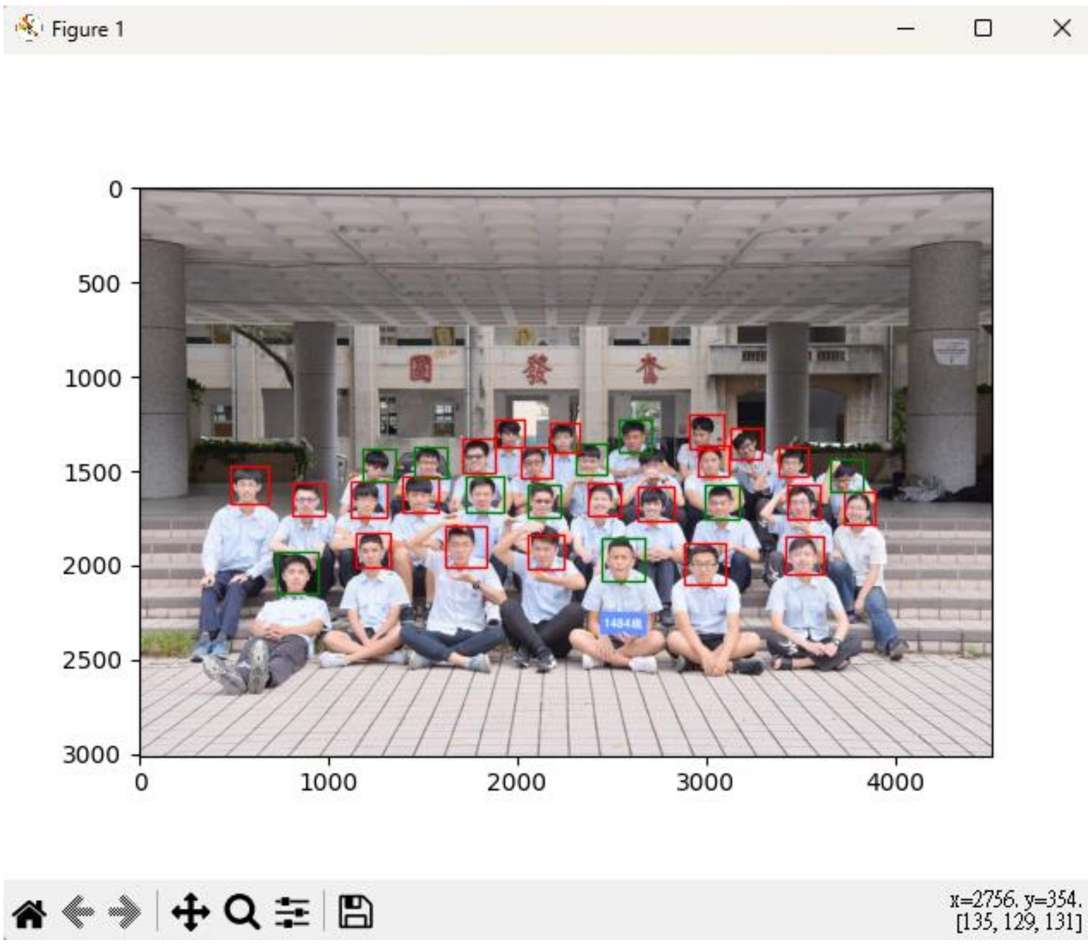


Figure 1
0
500
1000
1500
2000
2500
3000
0
1000
2000
3000
4000
x=2756, y=354,
[135, 129, 131]

Analysis:

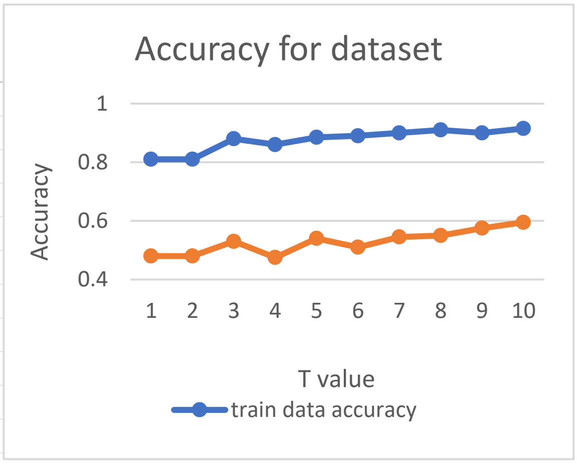
Training data has much higher accuracy than testing data.

And from t=3, training data accuracy has gradually stabilized.

It has not much influence from t bigger-than-3.

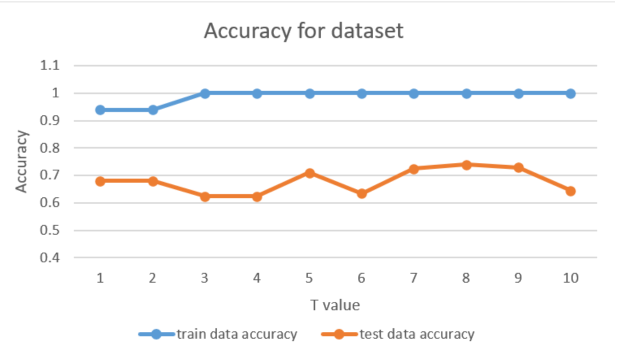
Original:

T	train data accuracy	test data accuracy
1	81.00%	48.00%
2	81.00%	48.00%
3	88.00%	53.00%
4	86.00%	47.50%
5	88.50%	54.00%
6	89.00%	51.00%
7	90.00%	54.50%
8	91.00%	55.00%
9	90.00%	57.50%
10	91.50%	59.50%



Change the threshold and polarity:

T	train data accuracy	test data accuracy
1	94.00%	68.00%
2	94.00%	68.00%
3	100.00%	62.50%
4	100.00%	62.50%
5	100.00%	71.00%
6	100.00%	63.50%
7	100.00%	72.50%
8	100.00%	74.00%
9	100.00%	73.00%
10	100.00%	64.50%



Part III:

1. Problem encountered and solution:

I don’t know how to use vscode to run python so I go to Pycharm. However, Pycharm will report error in some weird place, and I need to fix it. For example, in classify of classifier.py, it has some trouble identifying type of self.feature.computeFeature(x) causing it can’t return the correct value, so I need to deal with the different type and return the value respectively.

2. Limitations of Viola-Jones' algorithm:
Size of image in training dataset need to be the same of testing dataset.
Face in front view has higher accuracy. It's hard to detect if the face rotates.

3. How to improve the algorithm w/o changing dataset and T?
Reduce the not-face part in the rectangle we input to test.
Use composite features instead of the single feature we use originally.

4. Another possible face detection method:
Train a classifier that can simultaneously classify eyes, nose, ears…, and every part may give a weight for how it might be positively, and combining all weights from different parts and we can get the accuracy for whole image, while VJ algorithm need to divide into many weak classifiers.
The new classifier may identify many kinds of faces, like monkeys', dogs' or humans', if we train different dataset. But VJ only can detect face or non-face.