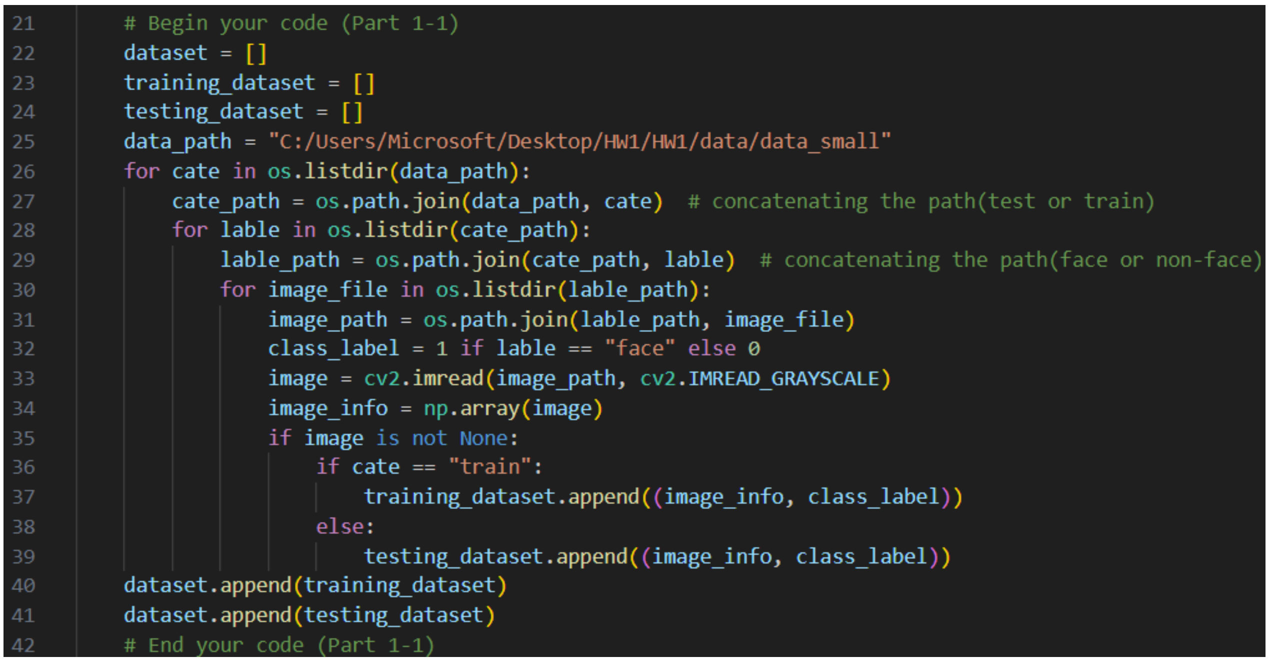
Homework 1: Face Detection Report Template

Part I. Implementation (5%):

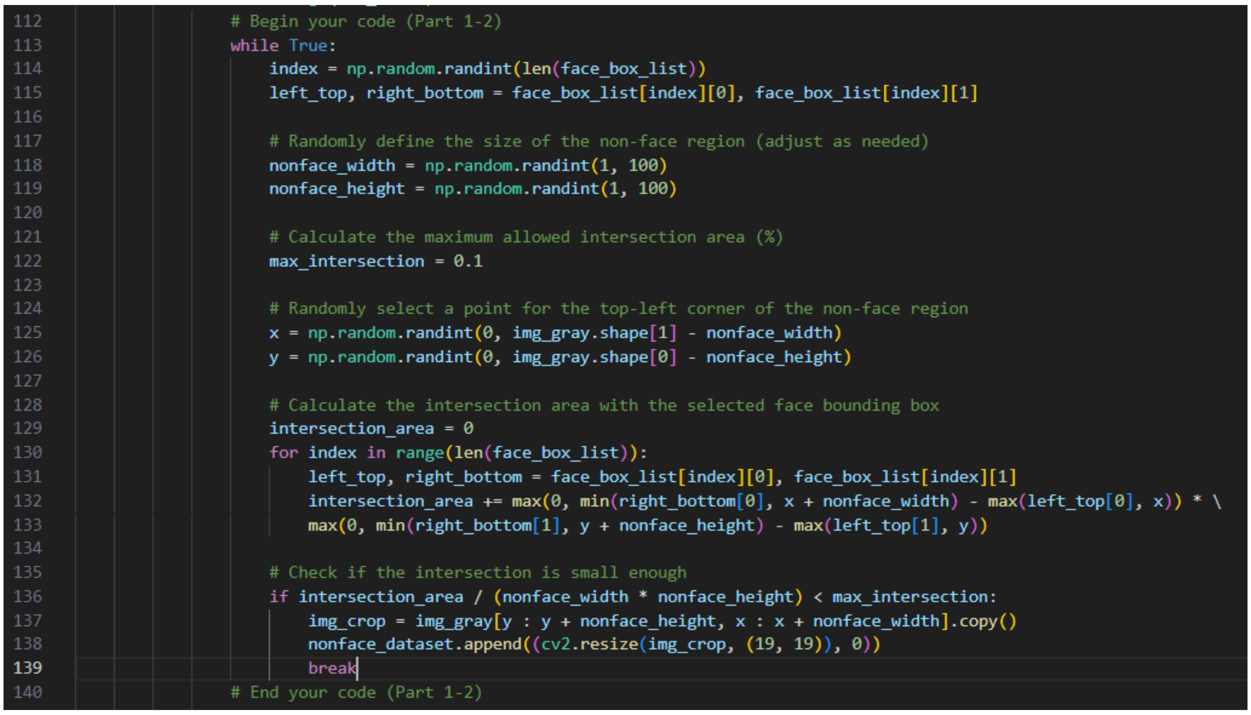
Part 1-1. Implement the “load_data_small” function in dataset.py that loads all images in the folder.



```python
    # Begin your code (Part 1-1)
    dataset = []
    training_dataset = []
    testing_dataset = []
    data_path = "C:/Users/Microsoft/Desktop/HW1/HW1/data/data_small"
    for cate in os.listdir(data_path):
        cate_path = os.path.join(data_path, cate)  # concatenating the path(test or train)
        for lable in os.listdir(cate_path):
            lable_path = os.path.join(cate_path, lable)  # concatenating the path(face or non-face)
            for image_file in os.listdir(lable_path):
                image_path = os.path.join(lable_path, image_file)
                class_label = 1 if lable == "face" else 0
                image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
                image_info = np.array(image)
                if image is not None:
                    if cate == "train":
                        training_dataset.append((image_info, class_label))
                    else:
                        testing_dataset.append((image_info, class_label))
    dataset.append(training_dataset)
    dataset.append(testing_dataset)
    # End your code (Part 1-1)
```

To begin with, I establish three sets: dataset, training_dataset, and testing_dataset, designed to store different kind of data. Data from the train fold will be placed into the training_dataset, while data from the test fold will be allocated to the testing_dataset.
Sequently, utilize for loops to iterate through various folders and load images. Convert images into numpy array format. Depending on the folder, assign labels and allocate them to either training_dataset or testing_dataset.
In the final step, consolidate the training_dataset and testing_dataset into the dataset for further processing.

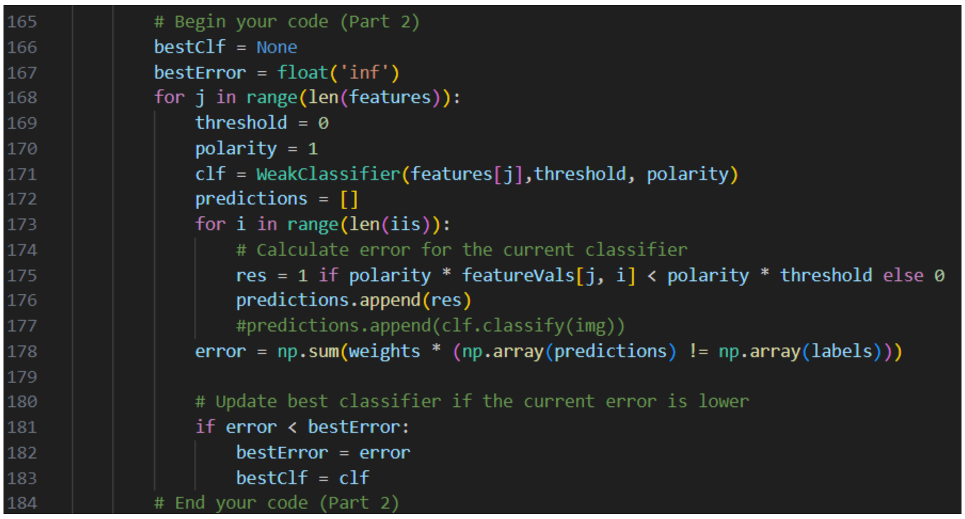
Part 1-2. Implement the "load_data_FDDB" function in dataset.py.



```python
                # Begin your code (Part 1-2)
                while True:
                    index = np.random.randint(len(face_box_list))
                    left_top, right_bottom = face_box_list[index][0], face_box_list[index][1]

                    # Randomly define the size of the non-face region (adjust as needed)
                    nonface_width = np.random.randint(1, 100)
                    nonface_height = np.random.randint(1, 100)

                    # Calculate the maximum allowed intersection area (%)
                    max_intersection = 0.1

                    # Randomly select a point for the top-left corner of the non-face region
                    x = np.random.randint(0, img_gray.shape[1] - nonface_width)
                    y = np.random.randint(0, img_gray.shape[0] - nonface_height)

                    # Calculate the intersection area with the selected face bounding box
                    intersection_area = 0
                    for index in range(len(face_box_list)):
                        left_top, right_bottom = face_box_list[index][0], face_box_list[index][1]
                        intersection_area += max(0, min(right_bottom[0], x + nonface_width) - max(left_top[0], x)) * \
                        max(0, min(right_bottom[1], y + nonface_height) - max(left_top[1], y))

                    # Check if the intersection is small enough
                    if intersection_area / (nonface_width * nonface_height) < max_intersection:
                        img_crop = img_gray[y : y + nonface_height, x : x + nonface_width].copy()
                        nonface_dataset.append((cv2.resize(img_crop, (19, 19)), 0))
                        break
                # End your code (Part 1-2)
```

First of all, I assign index with a random number representing which picture I want to choose.
Secondly, I determine the width and height of the image randomly and select the coordinates for the picture's top-left corner. These coordinates must not exceed the image boundary.
Lastly, I calculate the intersection area and intersection rate with the face picture. If the intersection rate exceeds our predefined threshold, I discard the selection and repeat the process until a satisfactory result is achieved.

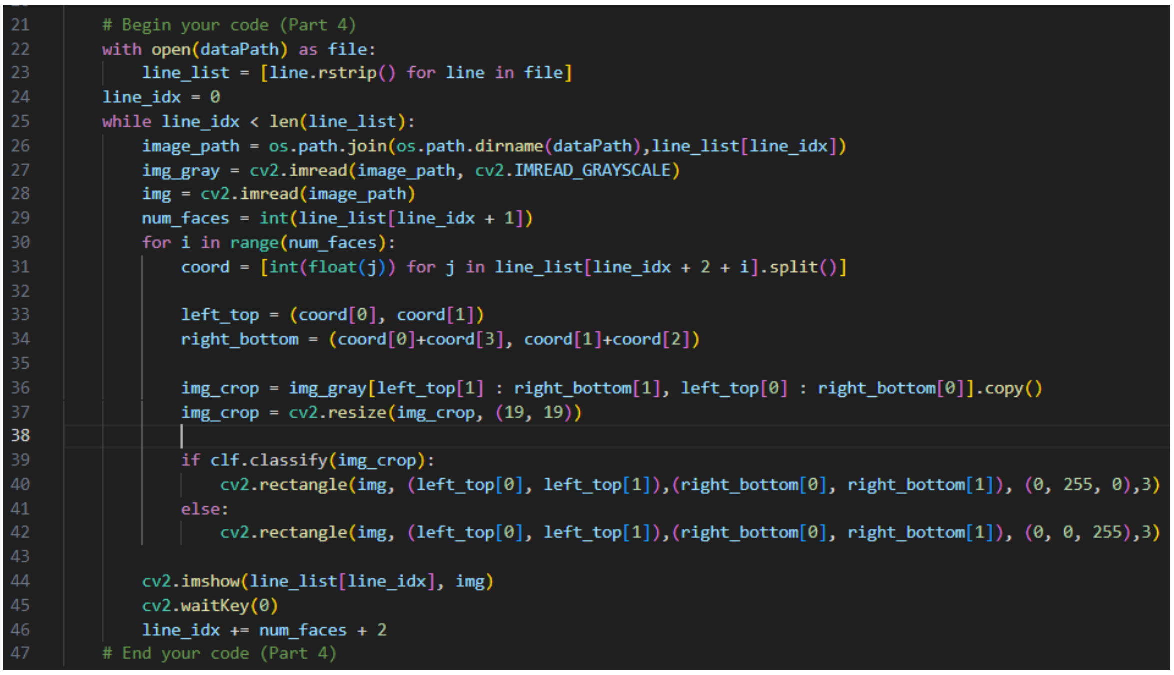
Part2. implement the “selectBest” function in adaboost.py to select the best weak classifier.



```python
        # Begin your code (Part 2)
        bestClf = None
        bestError = float('inf')
        for j in range(len(features)):
            threshold = 0
            polarity = 1
            clf = WeakClassifier(features[j],threshold, polarity)
            predictions = []
            for i in range(len(iis)):
                # Calculate error for the current classifier
                res = 1 if polarity * featureVals[j, i] < polarity * threshold else 0
                predictions.append(res)
                #predictions.append(clf.classify(img))
            error = np.sum(weights * (np.array(predictions) != np.array(labels)))

            # Update best classifier if the current error is lower
            if error < bestError:
                bestError = error
                bestClf = clf
        # End your code (Part 2)
```

I employ a for loop to iterate through each feature. Initialize the threshold to 0 and polarity to 1. Utilize the pre-calculated values stored in the featureVals list to predict whether the image is a face or non-face, and record the results in the prediction list. Subsequently, compare the predictions with the labels to calculate the error. If the error is minimal, update the bestError and bestClf.

Part4. implement the “detect” function in detection.py to load the images in the “DETECT” folder.



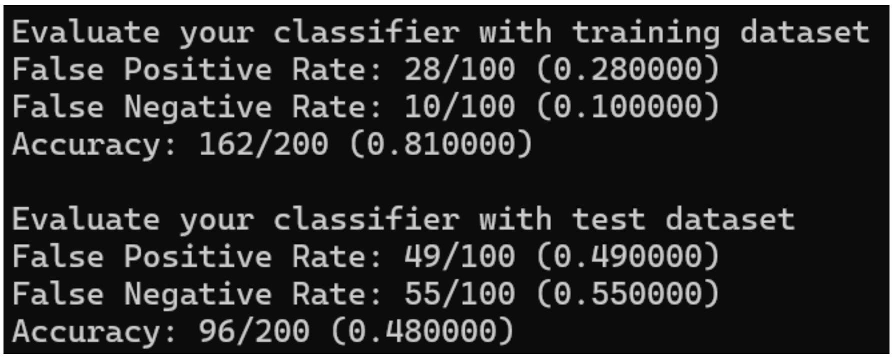
```python
    # Begin your code (Part 4)
    with open(dataPath) as file:
        line_list = [line.rstrip() for line in file]
    line_idx = 0
    while line_idx < len(line_list):
        image_path = os.path.join(os.path.dirname(dataPath),line_list[line_idx])
        img_gray = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        img = cv2.imread(image_path)
        num_faces = int(line_list[line_idx + 1])
        for i in range(num_faces):
            coord = [int(float(j)) for j in line_list[line_idx + 2 + i].split()]

            left_top = (coord[0], coord[1])
            right_bottom = (coord[0]+coord[3], coord[1]+coord[2])

            img_crop = img_gray[left_top[1] : right_bottom[1], left_top[0] : right_bottom[0]].copy()
            img_crop = cv2.resize(img_crop, (19, 19))

            if clf.classify(img_crop):
                cv2.rectangle(img, (left_top[0], left_top[1]),(right_bottom[0], right_bottom[1]), (0, 255, 0),3)
            else:
                cv2.rectangle(img, (left_top[0], left_top[1]),(right_bottom[0], right_bottom[1]), (0, 0, 255),3)

        cv2.imshow(line_list[line_idx], img)
        cv2.waitKey(0)
        line_idx += num_faces + 2
    # End your code (Part 4)
```

I utilize the dataPath to load a txt file, extracting its content line by line. Subsequently, I use the information in txt file to obtain the data path of an image and read it in grayscale format. Based on the coordinate of the picture's top-left corner, along with picture’s width and height, I crop the defined bounding box to predict whether it contains a face. Finally, I draw a rectangle on the original image to mark the prediction result. If it is a face, the rectangle is drawn in green (0, 255, 0); otherwise, it is drawn in red (0, 0, 255).

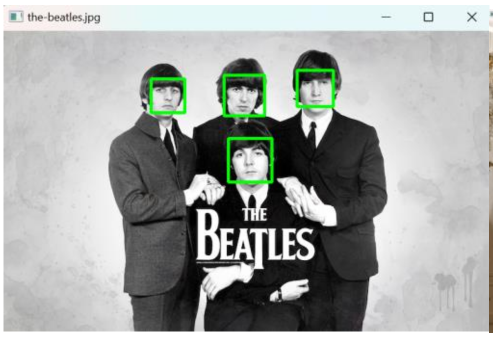
Part II. Results & Analysis (10%):

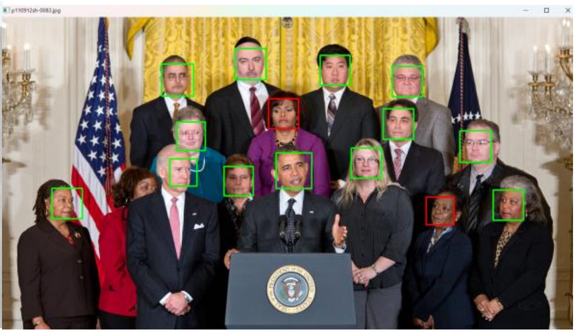
1. data small

data small (with T=1, threshold=0, polarity=1):

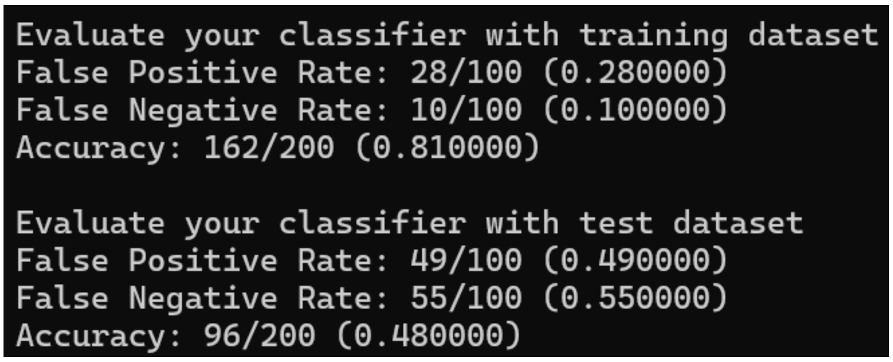


```
Evaluate your classifier with training dataset
False Positive Rate: 28/100 (0.280000)
False Negative Rate: 10/100 (0.100000)
Accuracy: 162/200 (0.810000)

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)
```

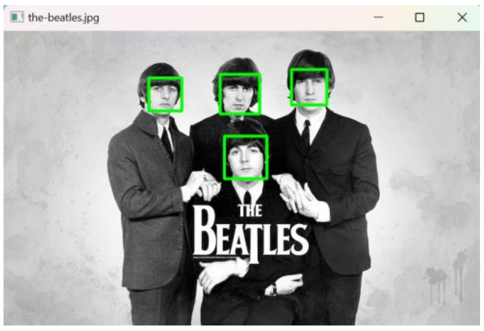


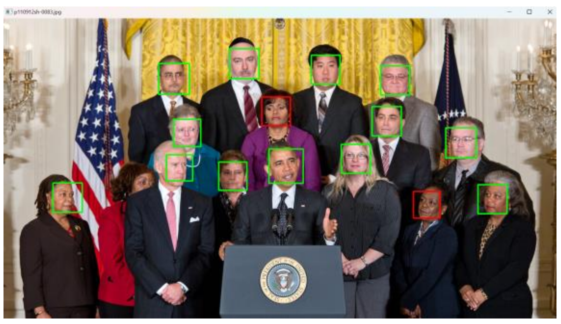


data small (with T=2, threshold=0, polarity=1):

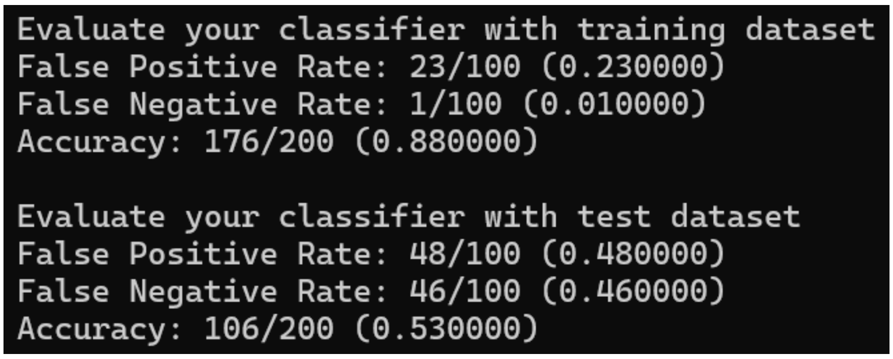


```
Evaluate your classifier with training dataset
False Positive Rate: 28/100 (0.280000)
False Negative Rate: 10/100 (0.100000)
Accuracy: 162/200 (0.810000)

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)
```

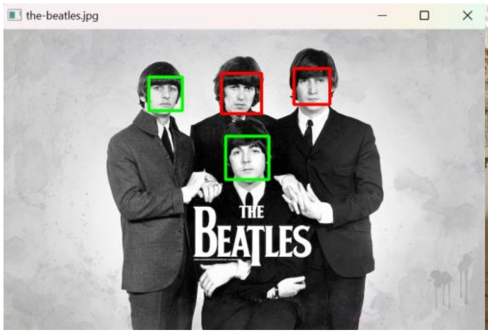


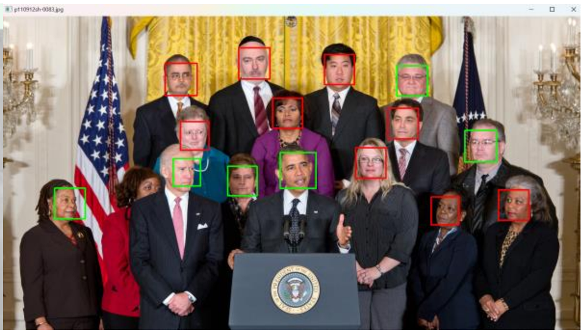


data small (with T=3, threshold=0, polarity=1):

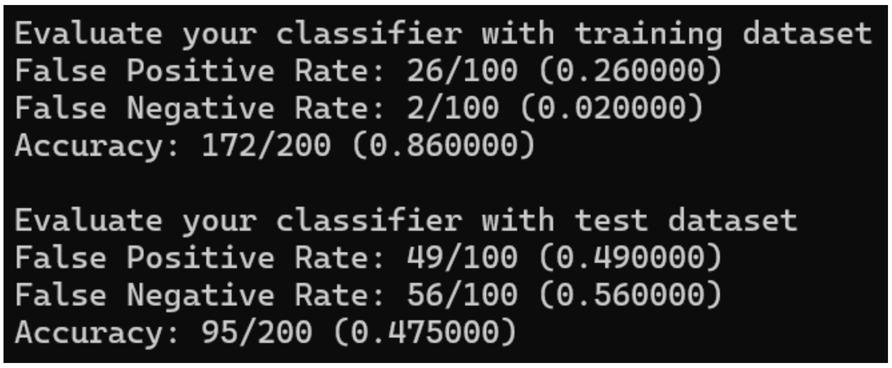


```
Evaluate your classifier with training dataset
False Positive Rate: 23/100 (0.230000)
False Negative Rate: 1/100 (0.010000)
Accuracy: 176/200 (0.880000)

Evaluate your classifier with test dataset
False Positive Rate: 48/100 (0.480000)
False Negative Rate: 46/100 (0.460000)
Accuracy: 106/200 (0.530000)
```

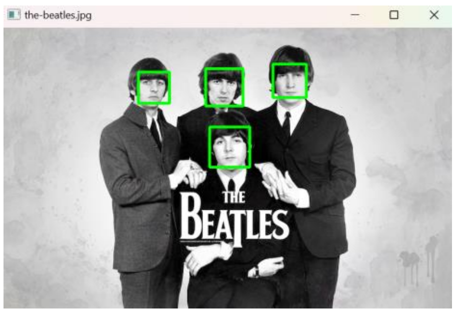


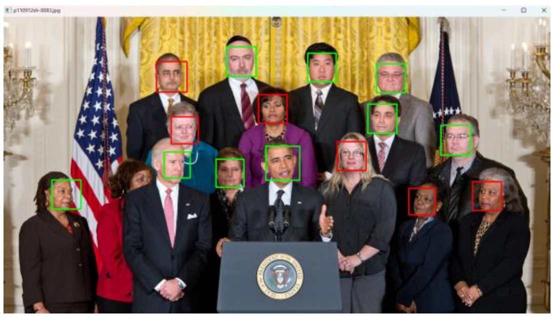


data small (with T=4, threshold=0, polarity=1):

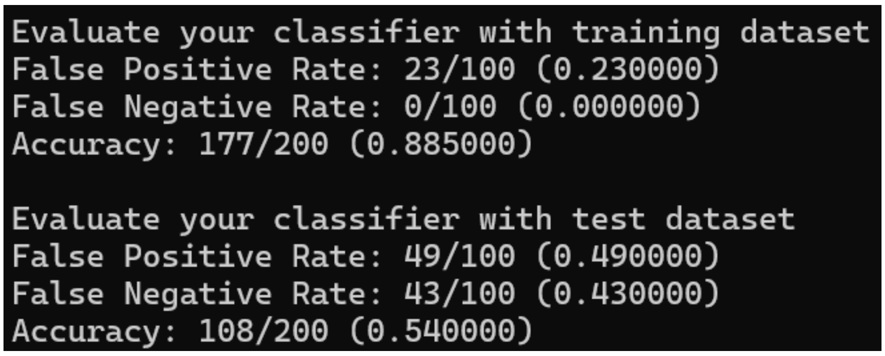


```
Evaluate your classifier with training dataset
False Positive Rate: 26/100 (0.260000)
False Negative Rate: 2/100 (0.020000)
Accuracy: 172/200 (0.860000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 56/100 (0.560000)
Accuracy: 95/200 (0.475000)
```

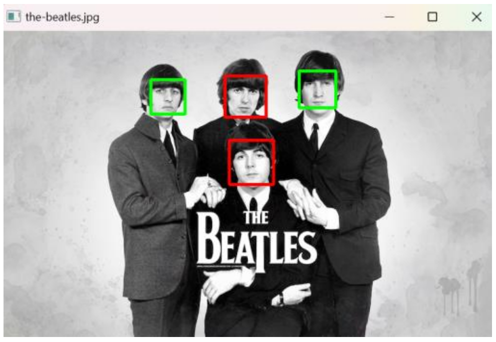


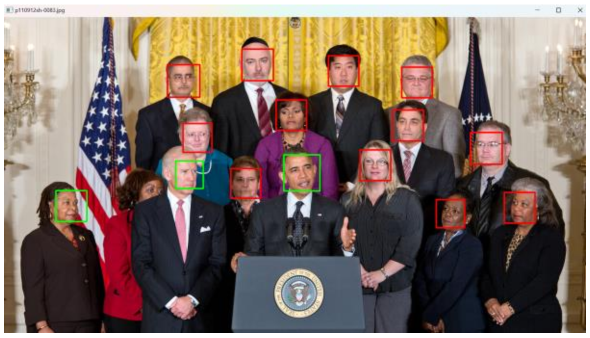


data small (with T=5, threshold=0, polarity=1):

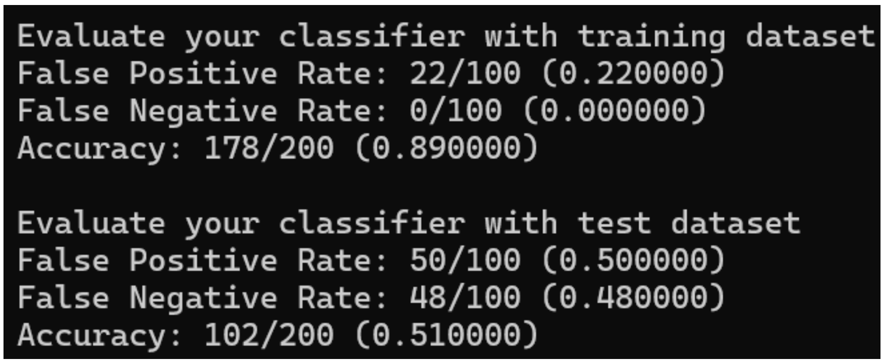


```
Evaluate your classifier with training dataset
False Positive Rate: 23/100 (0.230000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 177/200 (0.885000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 43/100 (0.430000)
Accuracy: 108/200 (0.540000)
```

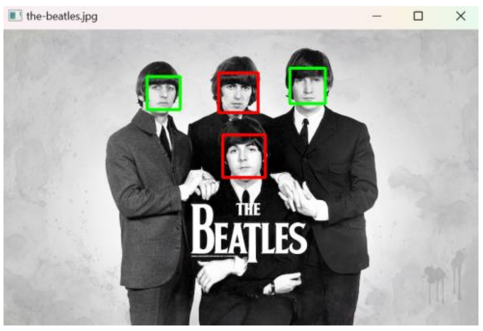


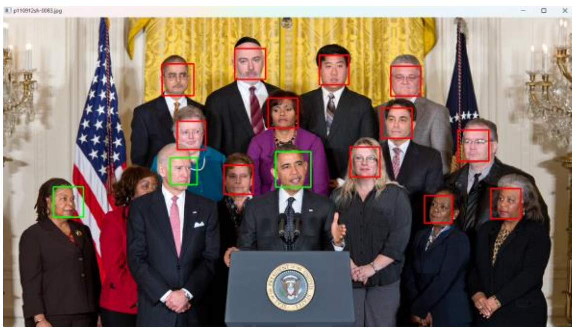


data small (with T=6, threshold=0, polarity=1):

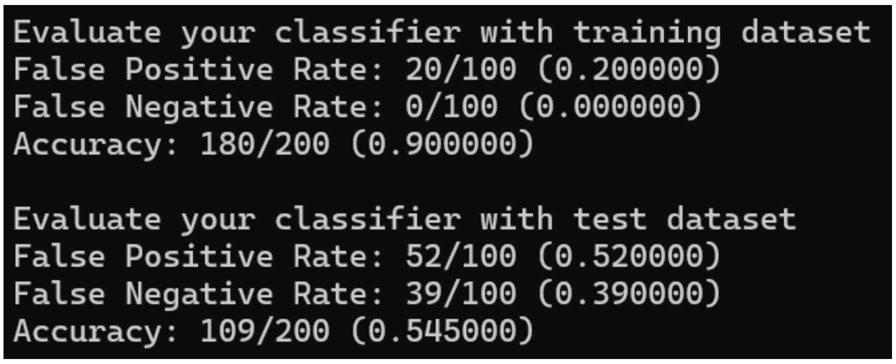


```
Evaluate your classifier with training dataset
False Positive Rate: 22/100 (0.220000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 178/200 (0.890000)

Evaluate your classifier with test dataset
False Positive Rate: 50/100 (0.500000)
False Negative Rate: 48/100 (0.480000)
Accuracy: 102/200 (0.510000)
```

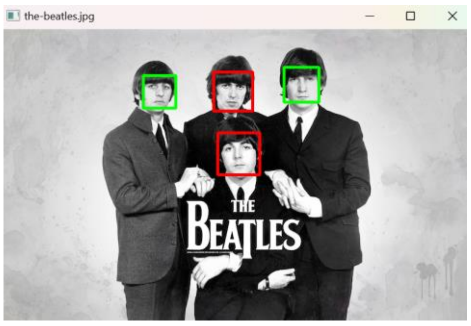


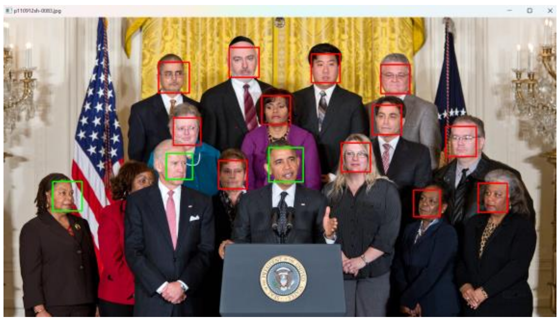


data small (with T=7, threshold=0, polarity=1):

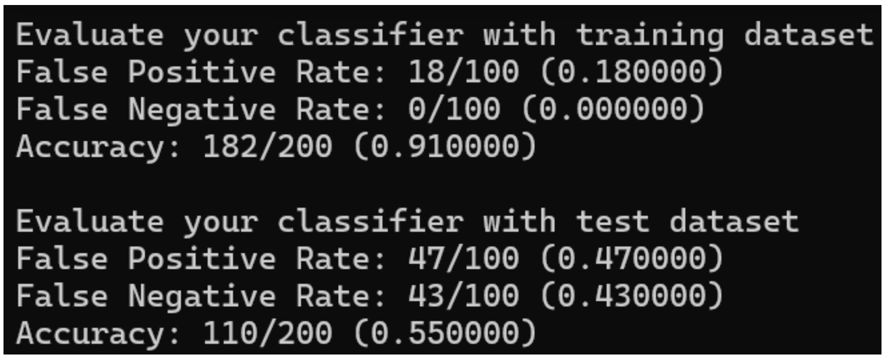


```
Evaluate your classifier with training dataset
False Positive Rate: 20/100 (0.200000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 180/200 (0.900000)

Evaluate your classifier with test dataset
False Positive Rate: 52/100 (0.520000)
False Negative Rate: 39/100 (0.390000)
Accuracy: 109/200 (0.545000)
```

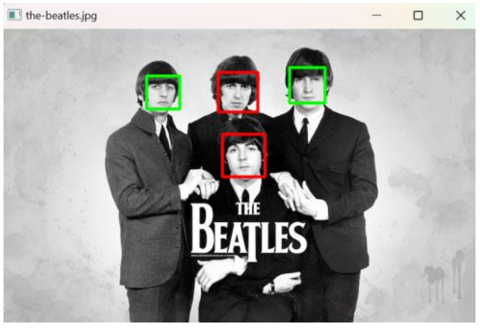


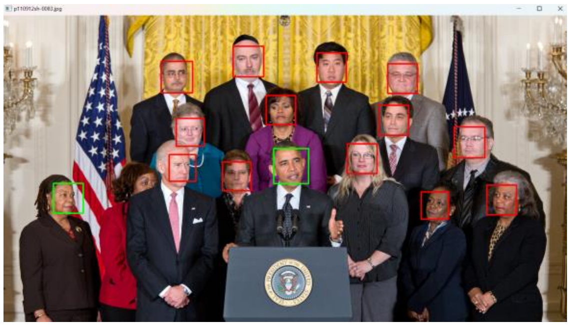


data small (with T=8, threshold=0, polarity=1):

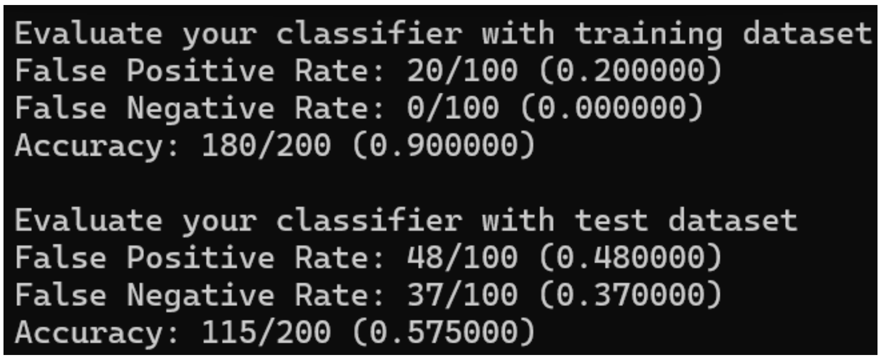


```
Evaluate your classifier with training dataset
False Positive Rate: 18/100 (0.180000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 182/200 (0.910000)

Evaluate your classifier with test dataset
False Positive Rate: 47/100 (0.470000)
False Negative Rate: 43/100 (0.430000)
Accuracy: 110/200 (0.550000)
```

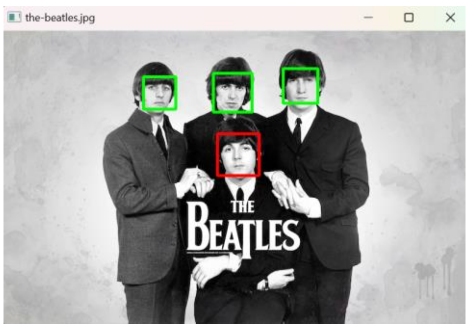


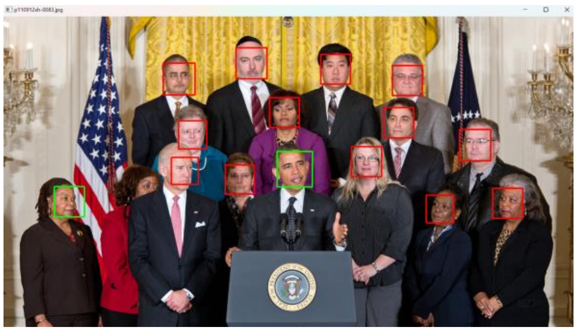


data small (with T=9, threshold=0, polarity=1):

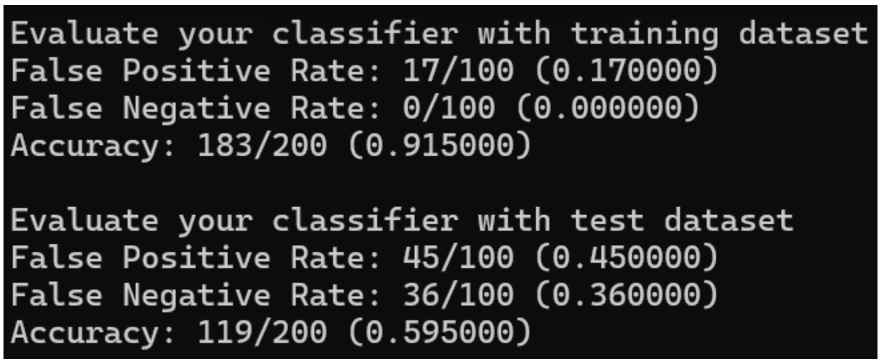


```
Evaluate your classifier with training dataset
False Positive Rate: 20/100 (0.200000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 180/200 (0.900000)

Evaluate your classifier with test dataset
False Positive Rate: 48/100 (0.480000)
False Negative Rate: 37/100 (0.370000)
Accuracy: 115/200 (0.575000)
```

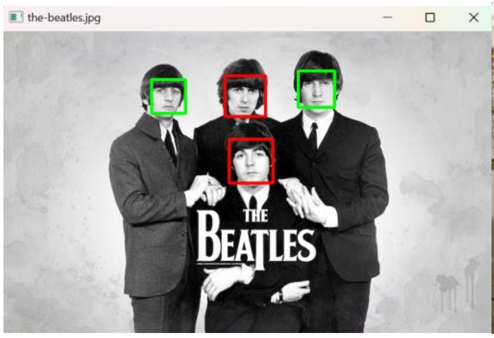


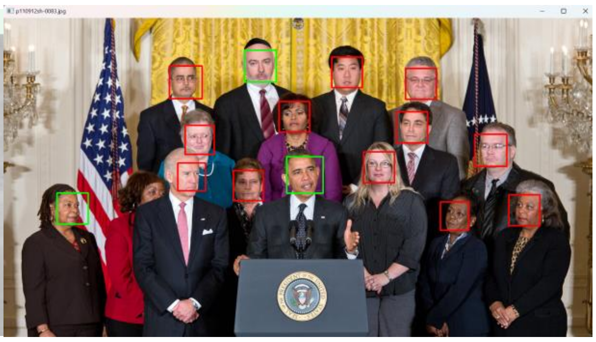


data small (with T=10, threshold=0, polarity=1):

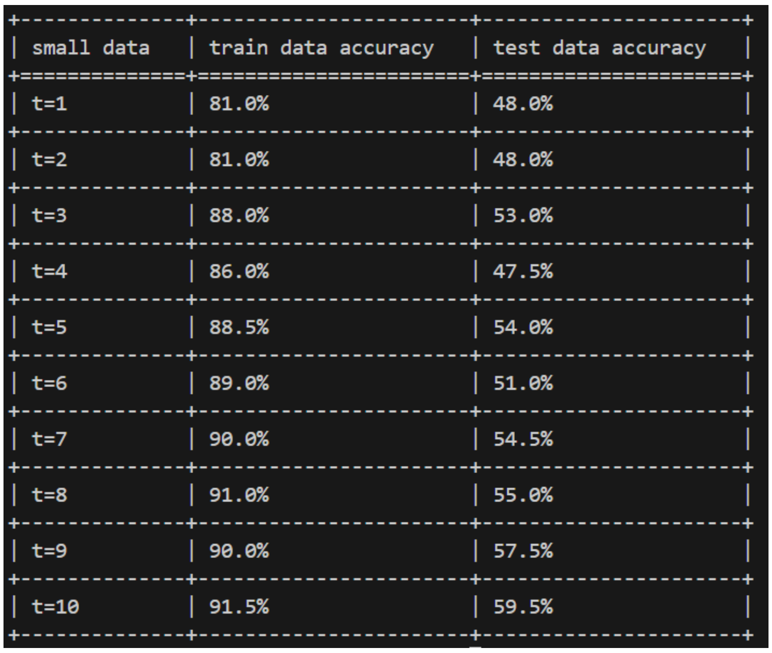


```
Evaluate your classifier with training dataset
False Positive Rate: 17/100 (0.170000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 183/200 (0.915000)

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)
```

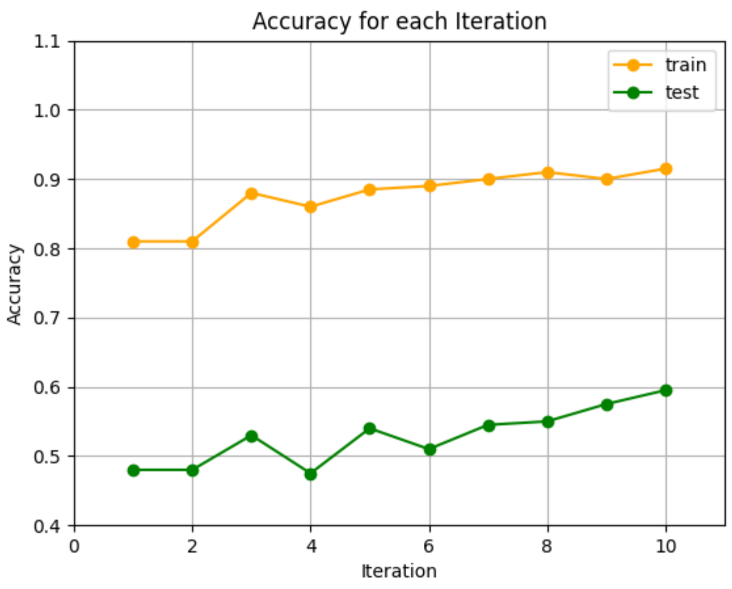




Conclusion:



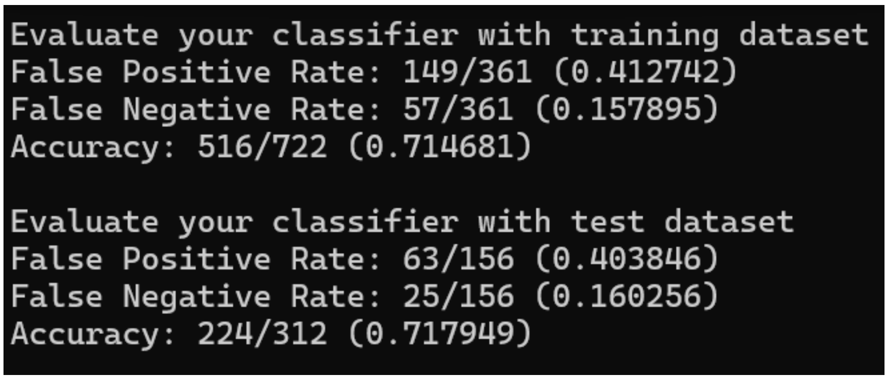
```
+--------------+----------------------+----------------------+
| small data   | train data accuracy  | test data accuracy   |
+==============+======================+======================+
| t=1          | 81.0%                | 48.0%                |
+--------------+----------------------+----------------------+
| t=2          | 81.0%                | 48.0%                |
+--------------+----------------------+----------------------+
| t=3          | 88.0%                | 53.0%                |
+--------------+----------------------+----------------------+
| t=4          | 86.0%                | 47.5%                |
+--------------+----------------------+----------------------+
| t=5          | 88.5%                | 54.0%                |
+--------------+----------------------+----------------------+
| t=6          | 89.0%                | 51.0%                |
+--------------+----------------------+----------------------+
| t=7          | 90.0%                | 54.5%                |
+--------------+----------------------+----------------------+
| t=8          | 91.0%                | 55.0%                |
+--------------+----------------------+----------------------+
| t=9          | 90.0%                | 57.5%                |
+--------------+----------------------+----------------------+
| t=10         | 91.5%                | 59.5%                |
+--------------+----------------------+----------------------+
```



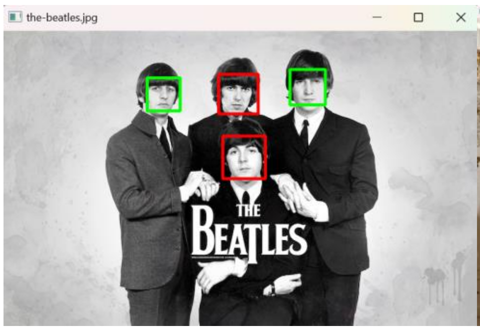
⇨ According to the line chart, we can observe that the accuracy on the test data increases steadily as the number of iterations increases. However, it shows that after 10 iterations, the test accuracy remains relatively low, around 60%. In contrast, the accuracy on the training data is significantly higher, reaching 91.5% after 10 iterations. This gap between training and test accuracy suggests that the model may be overfitting to the training data, failing to predict well to unseen data.

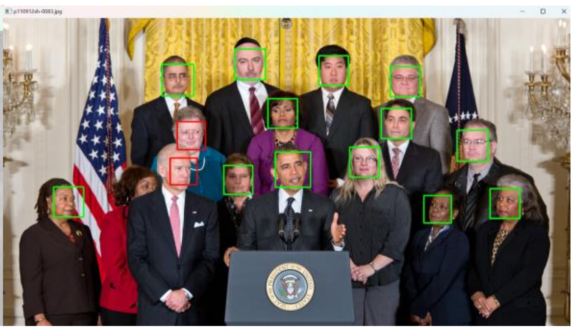
2. data FDDB(using FDDB-fold-01-ellipseList.txt and FDDB-fold-04-ellipseList.txt)

   data FDDB (with T=1, threshold=0, polarity=1):

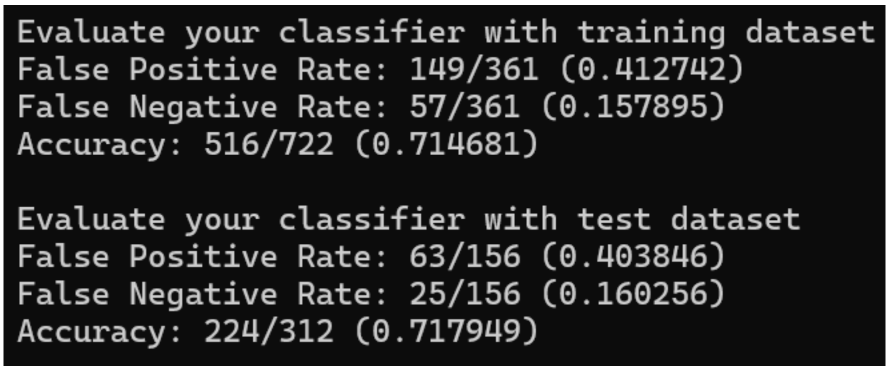


```
Evaluate your classifier with training dataset
False Positive Rate: 149/361 (0.412742)
False Negative Rate: 57/361 (0.157895)
Accuracy: 516/722 (0.714681)

Evaluate your classifier with test dataset
False Positive Rate: 63/156 (0.403846)
False Negative Rate: 25/156 (0.160256)
Accuracy: 224/312 (0.717949)
```

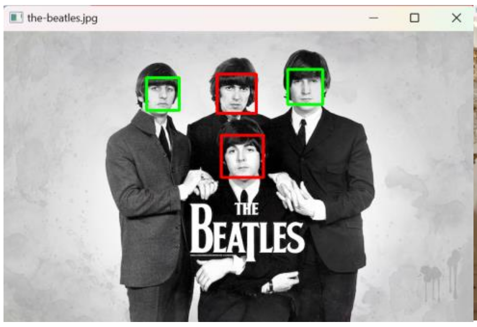


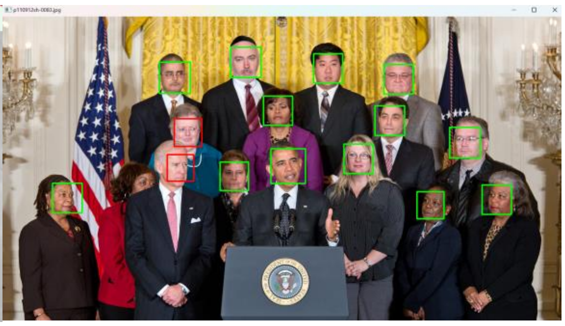


data FDDB (with T=2, threshold=0, polarity=1):

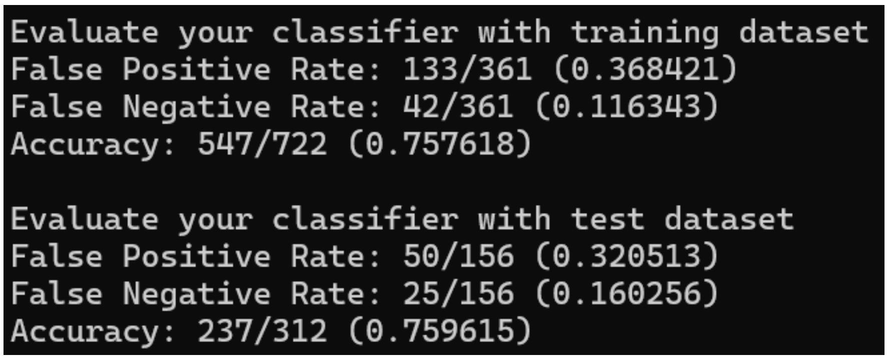


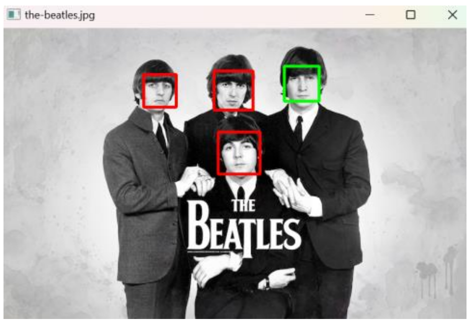
```
Evaluate your classifier with training dataset
False Positive Rate: 149/361 (0.412742)
False Negative Rate: 57/361 (0.157895)
Accuracy: 516/722 (0.714681)

Evaluate your classifier with test dataset
False Positive Rate: 63/156 (0.403846)
False Negative Rate: 25/156 (0.160256)
Accuracy: 224/312 (0.717949)
```

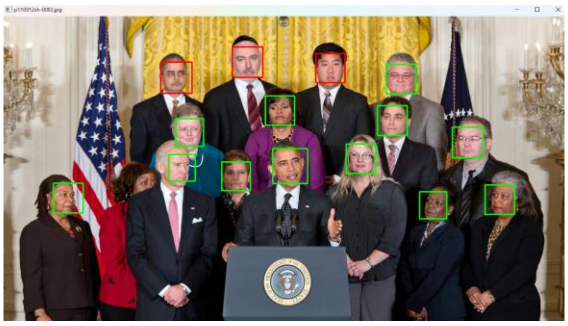




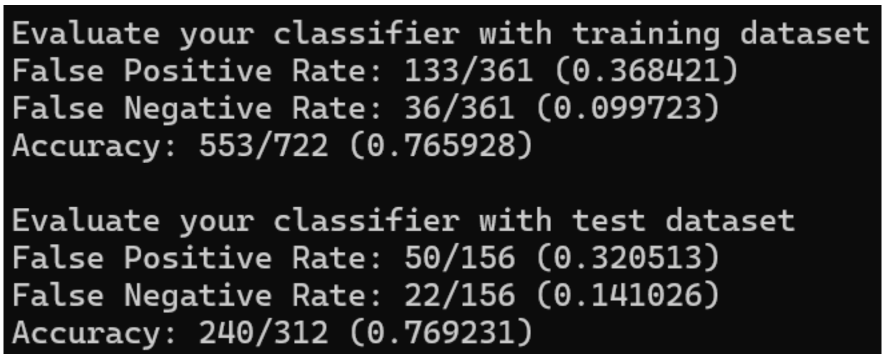
data FDDB (with T=3, threshold=0, polarity=1):

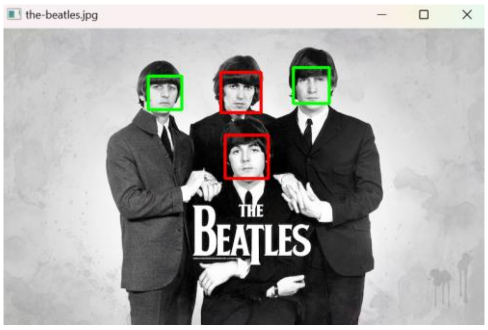

```
Evaluate your classifier with training dataset
False Positive Rate: 133/361 (0.368421)
False Negative Rate: 42/361 (0.116343)
Accuracy: 547/722 (0.757618)

Evaluate your classifier with test dataset
False Positive Rate: 50/156 (0.320513)
False Negative Rate: 25/156 (0.160256)
Accuracy: 237/312 (0.759615)
```

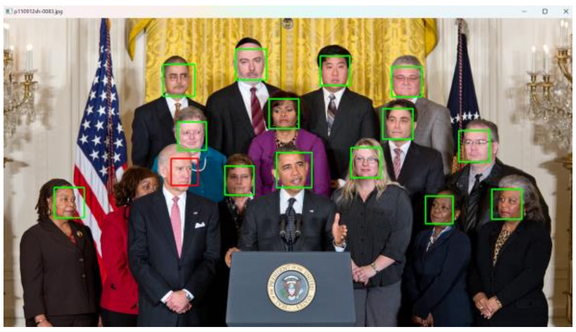




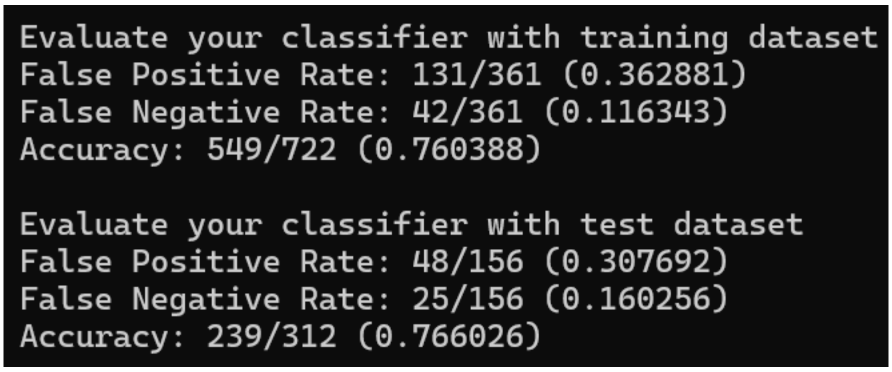
data FDDB (with T=4, threshold=0, polarity=1):


```
Evaluate your classifier with training dataset
False Positive Rate: 133/361 (0.368421)
False Negative Rate: 36/361 (0.099723)
Accuracy: 553/722 (0.765928)

Evaluate your classifier with test dataset
False Positive Rate: 50/156 (0.320513)
False Negative Rate: 22/156 (0.141026)
Accuracy: 240/312 (0.769231)
```

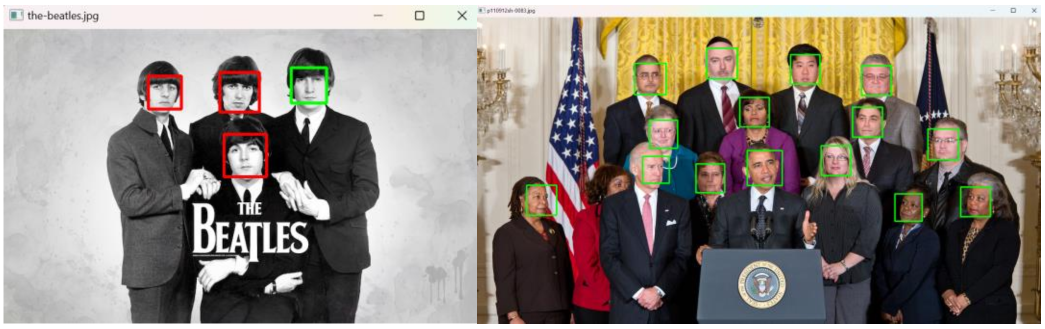




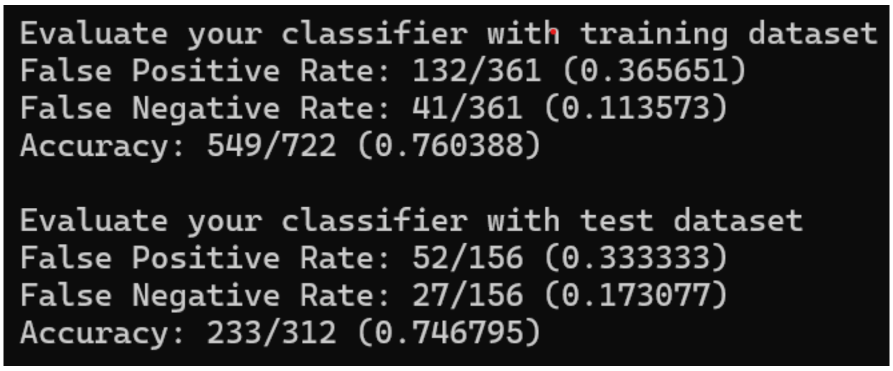
data FDDB (with T=5, threshold=0, polarity=1):



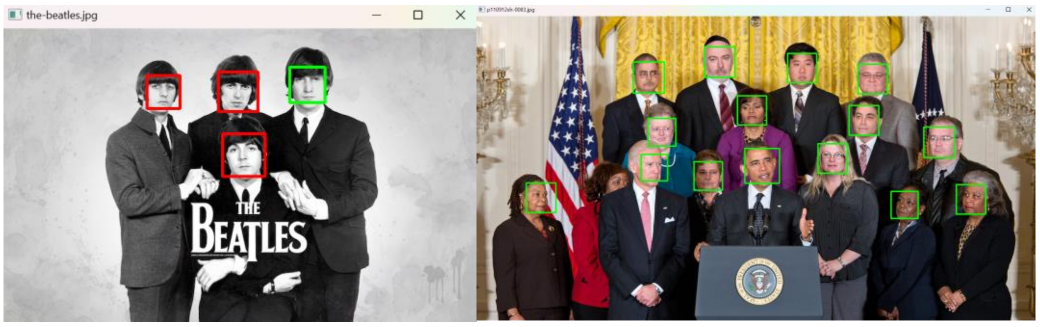
```
Evaluate your classifier with training dataset
False Positive Rate: 131/361 (0.362881)
False Negative Rate: 42/361 (0.116343)
Accuracy: 549/722 (0.760388)

Evaluate your classifier with test dataset
False Positive Rate: 48/156 (0.307692)
False Negative Rate: 25/156 (0.160256)
Accuracy: 239/312 (0.766026)
```



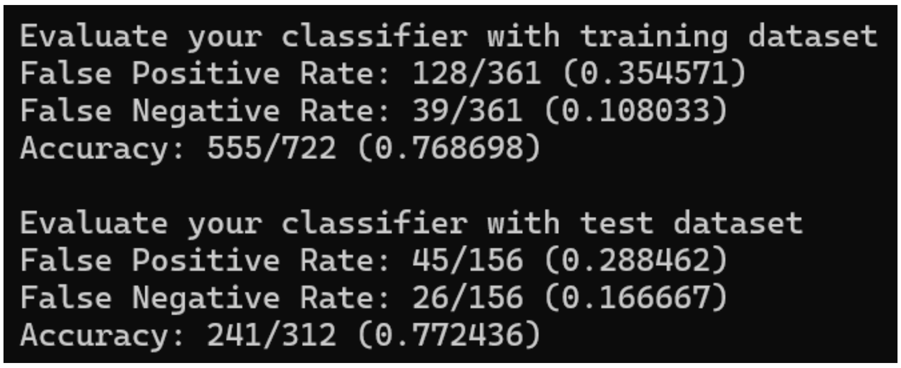
data FDDB (with T=6, threshold=0, polarity=1):



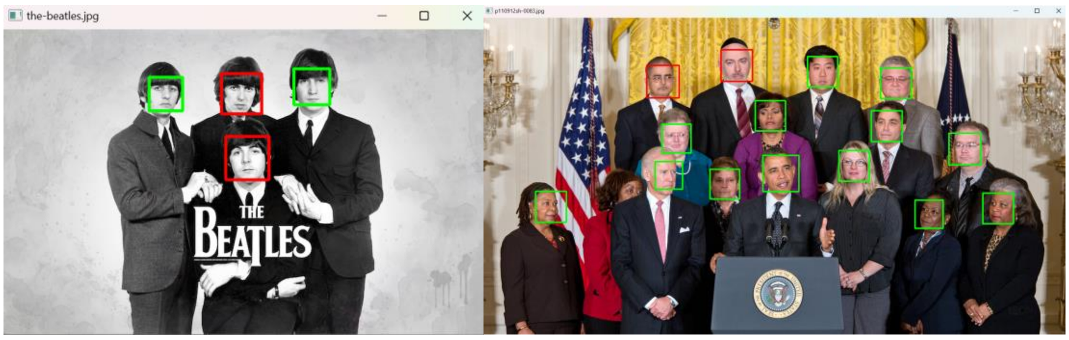
```
Evaluate your classifier with training dataset
False Positive Rate: 132/361 (0.365651)
False Negative Rate: 41/361 (0.113573)
Accuracy: 549/722 (0.760388)

Evaluate your classifier with test dataset
False Positive Rate: 52/156 (0.333333)
False Negative Rate: 27/156 (0.173077)
Accuracy: 233/312 (0.746795)
```



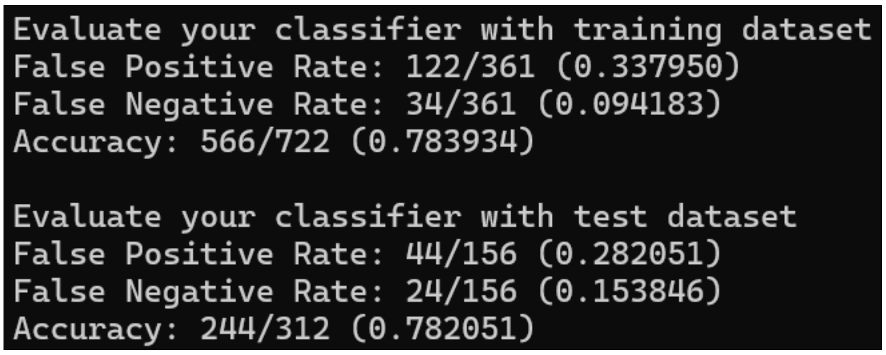
data FDDB (with T=7, threshold=0, polarity=1):



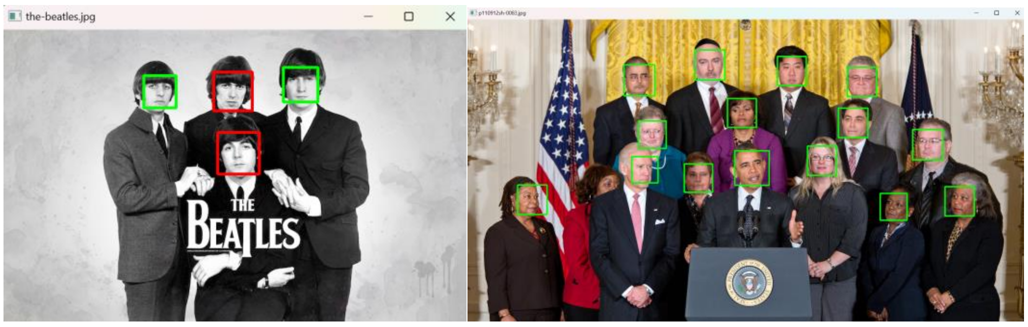
```
Evaluate your classifier with training dataset
False Positive Rate: 128/361 (0.354571)
False Negative Rate: 39/361 (0.108033)
Accuracy: 555/722 (0.768698)

Evaluate your classifier with test dataset
False Positive Rate: 45/156 (0.288462)
False Negative Rate: 26/156 (0.166667)
Accuracy: 241/312 (0.772436)
```



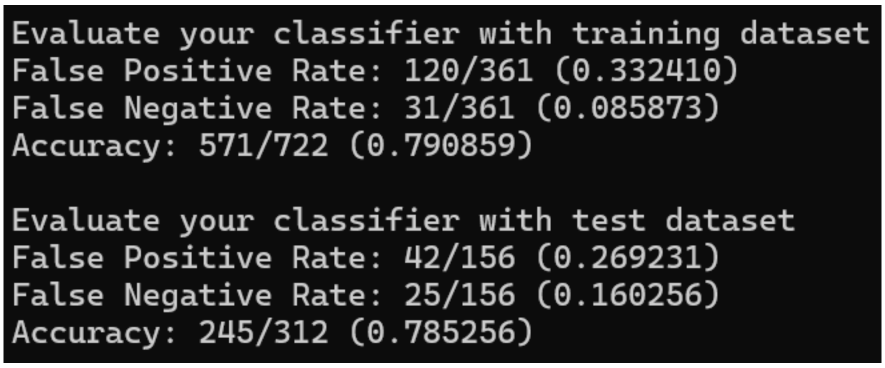
data FDDB (with T=8, threshold=0, polarity=1):



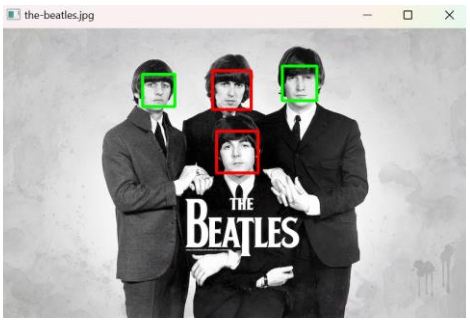
```
Evaluate your classifier with training dataset
False Positive Rate: 122/361 (0.337950)
False Negative Rate: 34/361 (0.094183)
Accuracy: 566/722 (0.783934)

Evaluate your classifier with test dataset
False Positive Rate: 44/156 (0.282051)
False Negative Rate: 24/156 (0.153846)
Accuracy: 244/312 (0.782051)
```

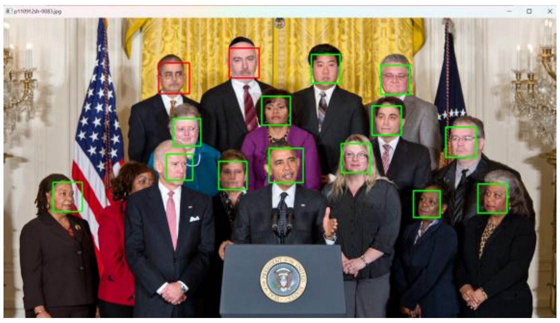


data FDDB (with T=9, threshold=0, polarity=1):

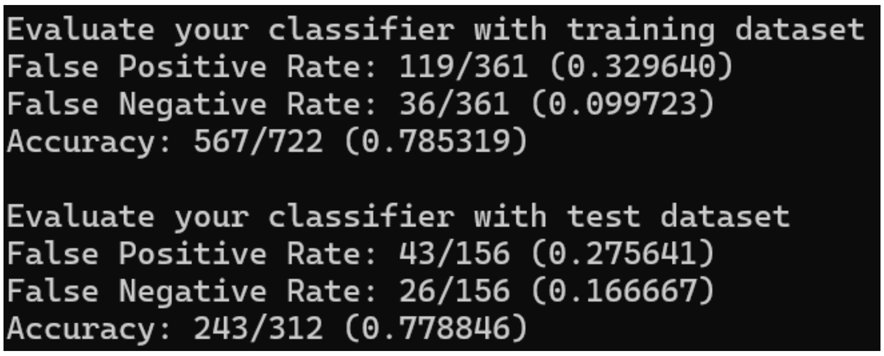


```
Evaluate your classifier with training dataset
False Positive Rate: 120/361 (0.332410)
False Negative Rate: 31/361 (0.085873)
Accuracy: 571/722 (0.790859)

Evaluate your classifier with test dataset
False Positive Rate: 42/156 (0.269231)
False Negative Rate: 25/156 (0.160256)
Accuracy: 245/312 (0.785256)
```

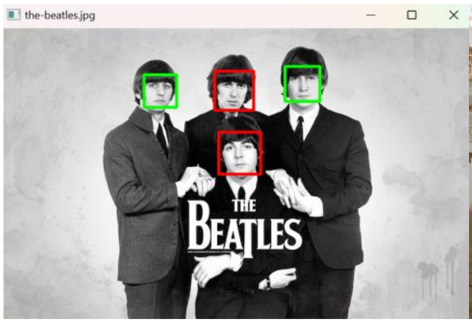


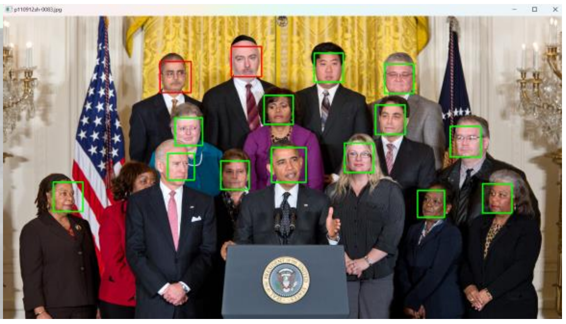


data FDDB (with T=10, threshold=0, polarity=1):

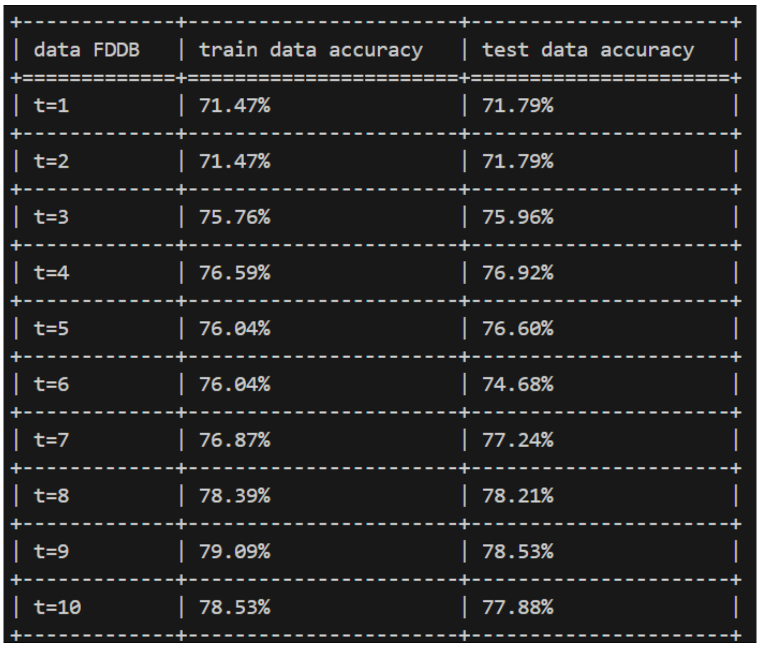


```
Evaluate your classifier with training dataset
False Positive Rate: 119/361 (0.329640)
False Negative Rate: 36/361 (0.099723)
Accuracy: 567/722 (0.785319)

Evaluate your classifier with test dataset
False Positive Rate: 43/156 (0.275641)
False Negative Rate: 26/156 (0.166667)
Accuracy: 243/312 (0.778846)
```

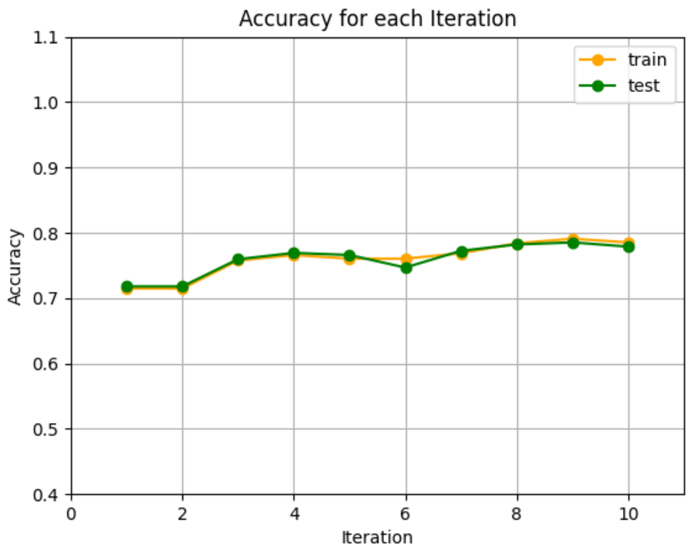




Conclusion:



```
+-------------+-----------------------+----------------------+
| data FDDB   | train data accuracy   | test data accuracy   |
+=============+=======================+======================+
| t=1         | 71.47%                | 71.79%               |
+-------------+-----------------------+----------------------+
| t=2         | 71.47%                | 71.79%               |
+-------------+-----------------------+----------------------+
| t=3         | 75.76%                | 75.96%               |
+-------------+-----------------------+----------------------+
| t=4         | 76.59%                | 76.92%               |
+-------------+-----------------------+----------------------+
| t=5         | 76.04%                | 76.60%               |
+-------------+-----------------------+----------------------+
| t=6         | 76.04%                | 74.68%               |
+-------------+-----------------------+----------------------+
| t=7         | 76.87%                | 77.24%               |
+-------------+-----------------------+----------------------+
| t=8         | 78.39%                | 78.21%               |
+-------------+-----------------------+----------------------+
| t=9         | 79.09%                | 78.53%               |
+-------------+-----------------------+----------------------+
| t=10        | 78.53%                | 77.88%               |
+-------------+-----------------------+----------------------+
```



⇨ According to the line chart, we can observe that both the training and testing accuracies increase gradually with more iterations, though the rate of improvement is relatively modest. Interestingly, the accuracies on the training and testing data are quite similar throughout the iterations, with no significant gap between the two. This suggests that the model is not exhibiting substantial overfitting to the training data.

Comparison:

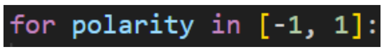
Based on the provided information, it is evident that the model trained on the FDDB dataset achieves higher classification accuracy compared to the one trained on the small dataset. This gap in performance can likely be attributed to the disparity in data volume between the two sets, with the FDDB dataset comprising 517 images and the smaller dataset containing only 200 images. The limited size of the smaller dataset likely led to overfitting during model training, resulting in poor performance on testing data. This poor performance can also be observed by the numerous red rectangles, indicating misclassifications, in the image of Obama.

Part 6: Implement another classifier (Bonus)

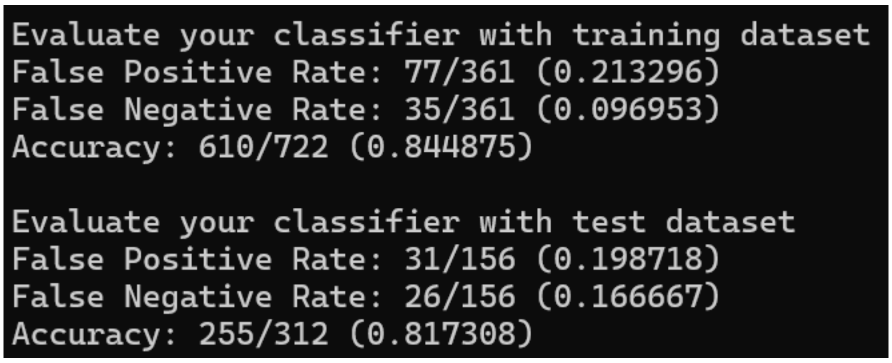
To achieve higher accuracy, we opted for the FDDB dataset and selected T=8 as the parameter, which exhibited the highest accuracy in previous training iterations.

1. change polarity

   The weak classifier can select its polarity as either 1 or -1. (with T=8, threshold=0)



```
for polarity in [-1, 1]:
```



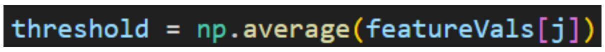
```
Evaluate your classifier with training dataset
False Positive Rate: 77/361 (0.213296)
False Negative Rate: 35/361 (0.096953)
Accuracy: 610/722 (0.844875)

Evaluate your classifier with test dataset
False Positive Rate: 31/156 (0.198718)
False Negative Rate: 26/156 (0.166667)
Accuracy: 255/312 (0.817308)
```

⇨ We can observe that the accuracy has successfully increased to 81.7%. Therefore, this parameter adjustment has a positive impact on the model's predictions. Consequently, in subsequent models, the polarity will be allowed to choose either 1 or -1.

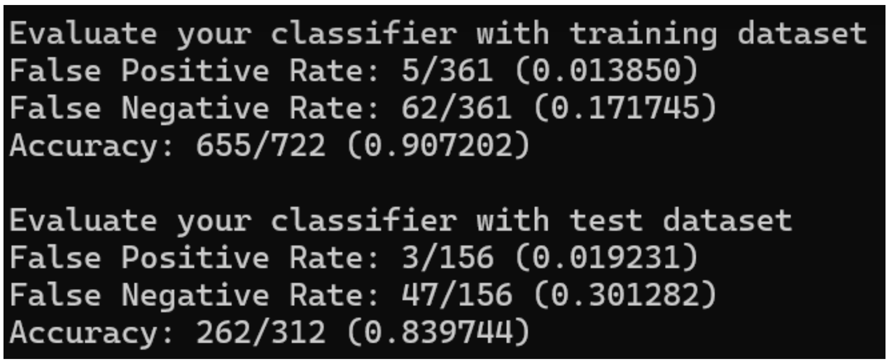
2. change threshold

   Method1

   threshold equals to the average of the feature values for different features.



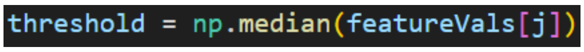
```
threshold = np.average(featureVals[j])
```



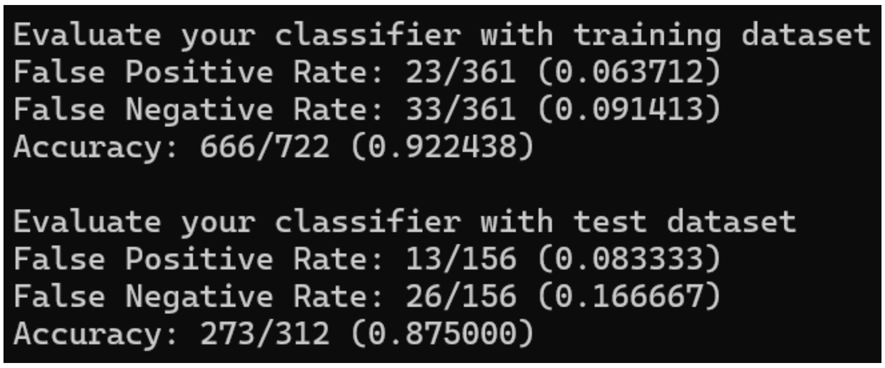
```
Evaluate your classifier with training dataset
False Positive Rate: 5/361 (0.013850)
False Negative Rate: 62/361 (0.171745)
Accuracy: 655/722 (0.907202)

Evaluate your classifier with test dataset
False Positive Rate: 3/156 (0.019231)
False Negative Rate: 47/156 (0.301282)
Accuracy: 262/312 (0.839744)
```

   Method2

   threshold equals to the median of the feature values for different features.



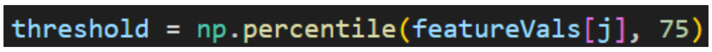
```
threshold = np.median(featureVals[j])
```



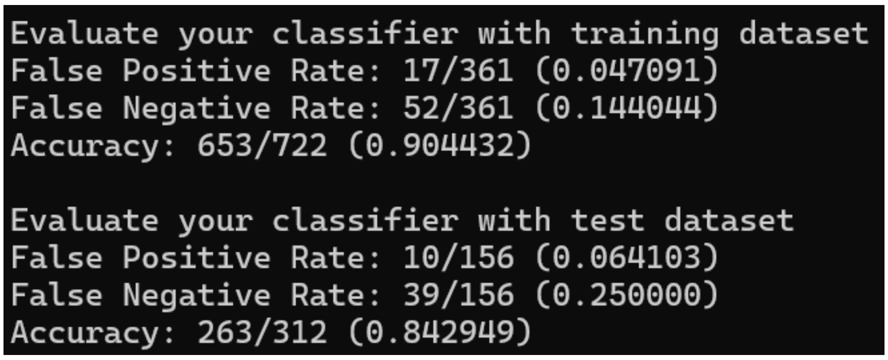
```
Evaluate your classifier with training dataset
False Positive Rate: 23/361 (0.063712)
False Negative Rate: 33/361 (0.091413)
Accuracy: 666/722 (0.922438)

Evaluate your classifier with test dataset
False Positive Rate: 13/156 (0.083333)
False Negative Rate: 26/156 (0.166667)
Accuracy: 273/312 (0.875000)
```

   Method3

   threshold equals to the 75th percentile of the feature values for different features.



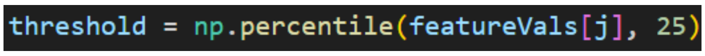
```
threshold = np.percentile(featureVals[j], 75)
```



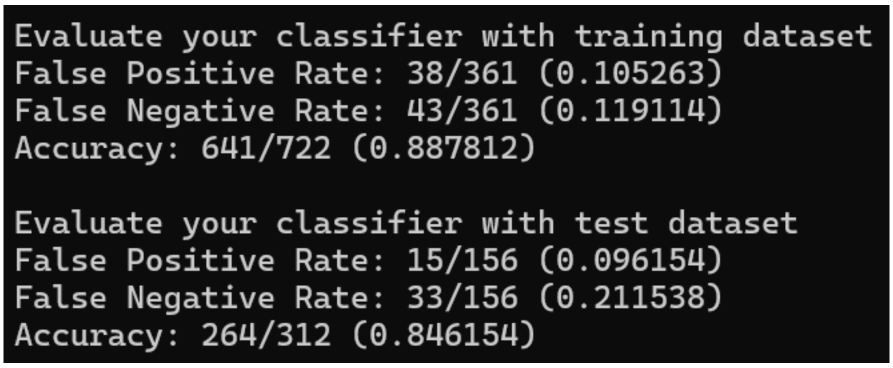
```
Evaluate your classifier with training dataset
False Positive Rate: 17/361 (0.047091)
False Negative Rate: 52/361 (0.144044)
Accuracy: 653/722 (0.904432)

Evaluate your classifier with test dataset
False Positive Rate: 10/156 (0.064103)
False Negative Rate: 39/156 (0.250000)
Accuracy: 263/312 (0.842949)
```

Method4

threshold equals to the 25th percentile of the feature values for different features.

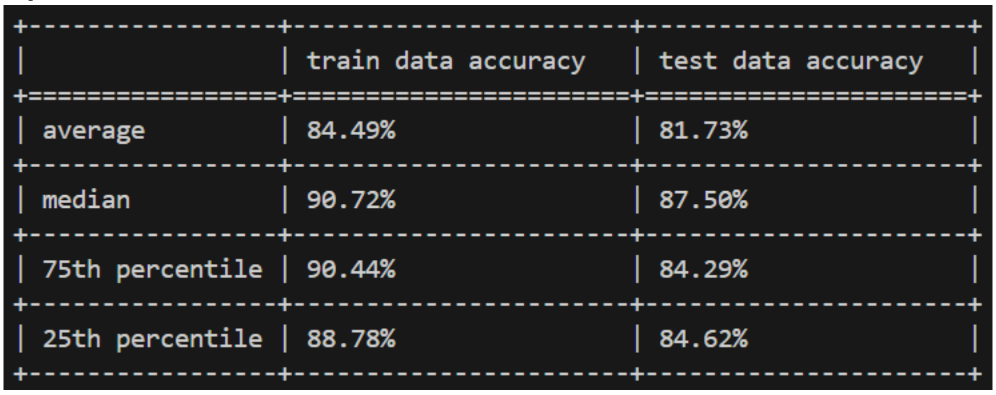


```
threshold = np.percentile(featureVals[j], 25)
```



```
Evaluate your classifier with training dataset
False Positive Rate: 38/361 (0.105263)
False Negative Rate: 43/361 (0.119114)
Accuracy: 641/722 (0.887812)

Evaluate your classifier with test dataset
False Positive Rate: 15/156 (0.096154)
False Negative Rate: 33/156 (0.211538)
Accuracy: 264/312 (0.846154)
```

Comparison:

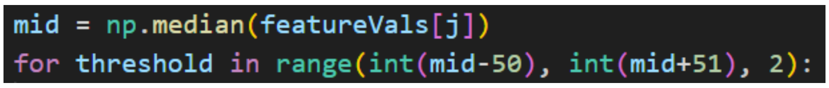


```
+-----------------+----------------------+----------------------+
|                 | train data accuracy  | test data accuracy   |
+=================+======================+======================+
| average         | 84.49%               | 81.73%               |
+-----------------+----------------------+----------------------+
| median          | 90.72%               | 87.50%               |
+-----------------+----------------------+----------------------+
| 75th percentile | 90.44%               | 84.29%               |
+-----------------+----------------------+----------------------+
| 25th percentile | 88.78%               | 84.62%               |
+-----------------+----------------------+----------------------+
```

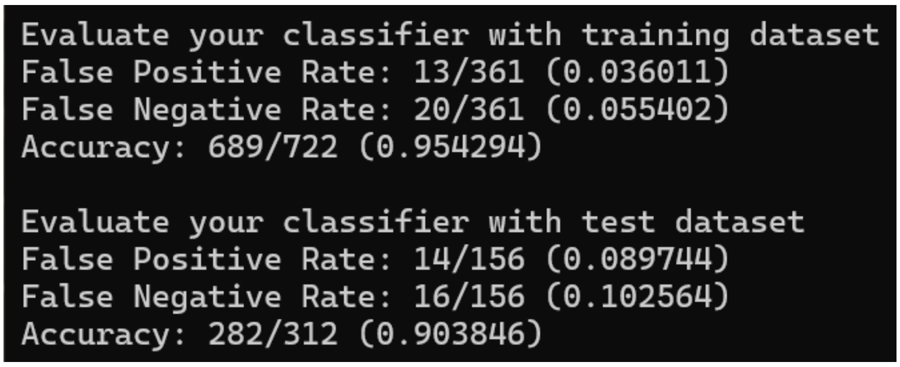
⇨ From the table above, we can observe that the highest accuracy in both the training and testing data occurs when the threshold equals the median. Hence, it suggests that the optimal threshold value might lie around the median of each feature's values.

3. change threshold

   The threshold could potentially fall within a range of numbers approximately spanning from the median minus 50 to the median plus 50 of the feature values across different features.



```
mid = np.median(featureVals[j])
for threshold in range(int(mid-50), int(mid+51), 2):
```

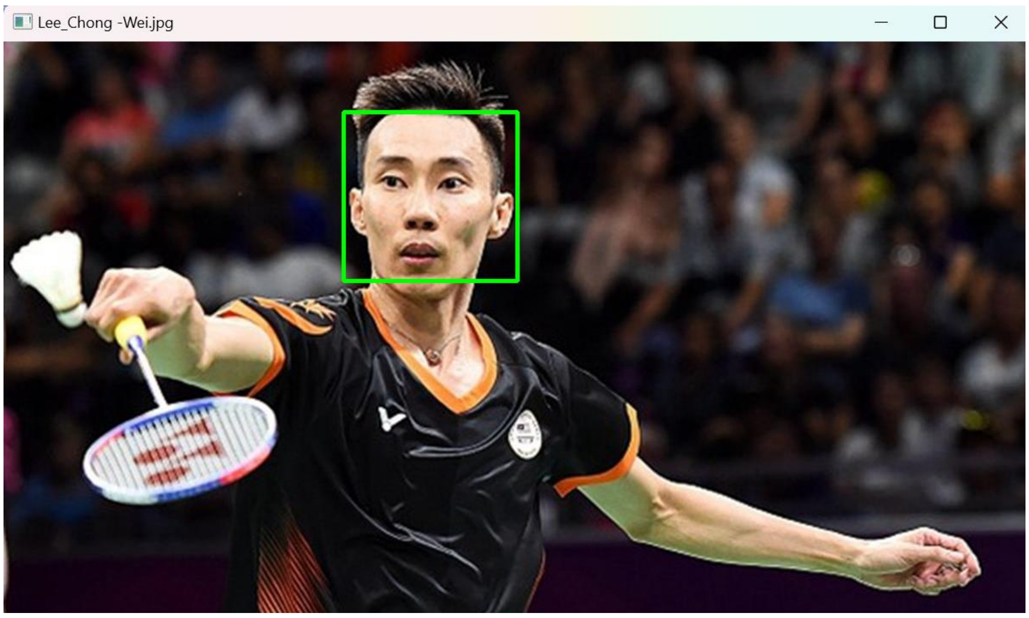


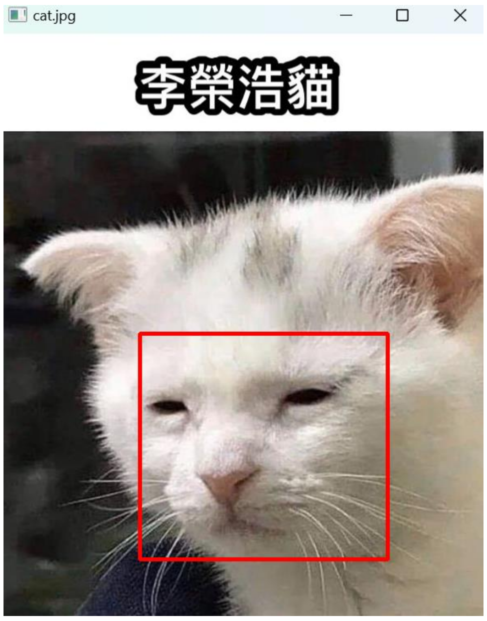
```
Evaluate your classifier with training dataset
False Positive Rate: 13/361 (0.036011)
False Negative Rate: 20/361 (0.055402)
Accuracy: 689/722 (0.954294)

Evaluate your classifier with test dataset
False Positive Rate: 14/156 (0.089744)
False Negative Rate: 16/156 (0.102564)
Accuracy: 282/312 (0.903846)
```

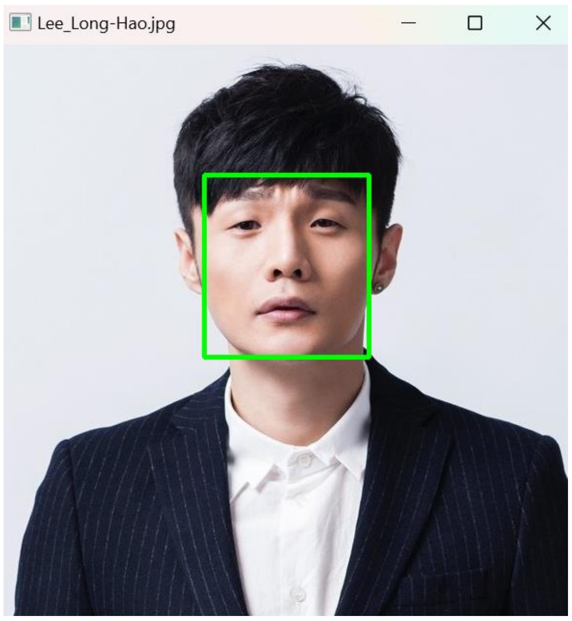
⇨ We achieved an accuracy exceeding 90%, marking significant progress in correctly classifying images compared to the original accuracy of 78.53%.

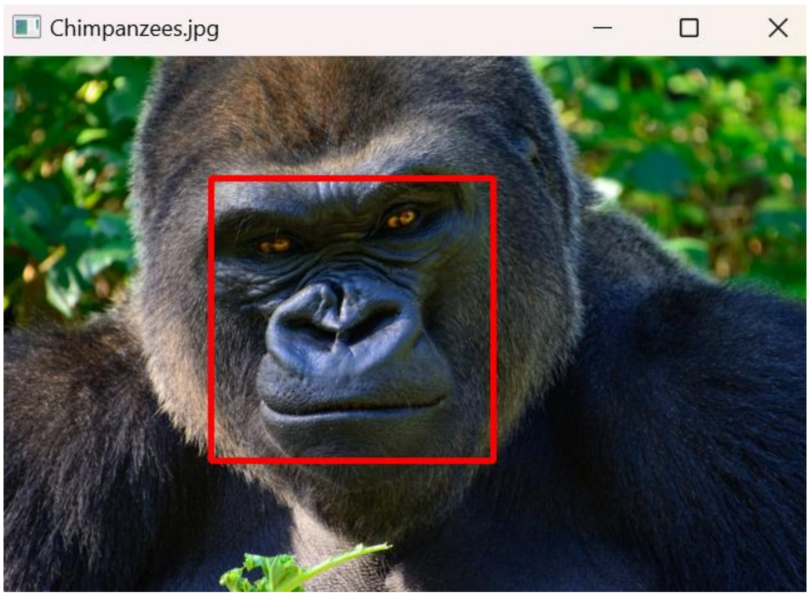
Part 5: Test classifier on your own images

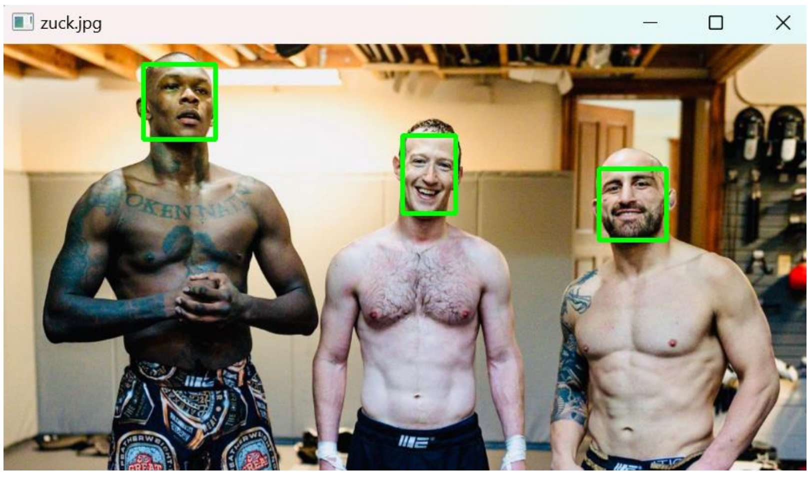
Use the model which we get the highest accuracy before to test my own image.




cat.jpg
李榮浩貓


Lee_Long-Hao.jpg

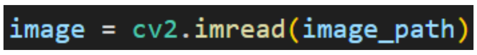

Chimpanzees.jpg


zuck.jpg

Part III. Answer the questions (15%):

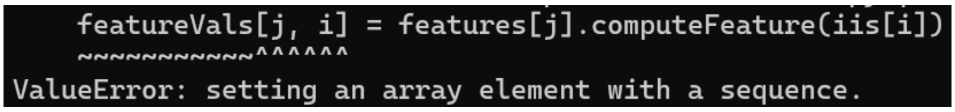
1. Please describe a problem you encountered and how you solved it.

A: When I want to read the image, I just use this code:



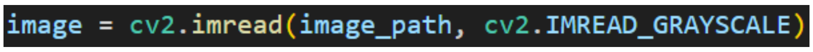
```
image = cv2.imread(image_path)
```

At the beginning, it can load the dataset successfully. Therefore, I do not suspect that I have anything wrong until following mistake appears:



```
    featureVals[j, i] = features[j].computeFeature(iis[i])
    ~~~~~~~~~~~~^^^^^^
ValueError: setting an array element with a sequence.
```

The mistake is attributed to loading the image as a RGB image, causing the numpy array to increase by one dimension (to record the value of RGB). To solve this issue, we simply need to replace the code with:



```
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

2. How do you generate “nonface” data by cropping images?

A: I randomly select the width and height of images and then randomly choose the coordinates of the cropping image's left corner. Next, I calculate the intersection area between the face boundary box and the cropping images. If there are multiple faces, I iterate the calculation process for each face boundary box. Finally, I sum up all intersection areas and calculate the intersection rate. If it exceeds the predefined threshold, I discard the cropping image and repeat the steps until I successfully obtain a cropping image.

3. What are the limitations of the Viola-Jones’ algorithm?

A: There are many limitations of the Viola-Jones’ algorithm. I will list some as following:

(1) Dependency on training data: The performance of the Viola-Jones algorithm heavily relies on the quality and diversity of the training data. Insufficient or biased training data can lead to reduced accuracy and generalization capabilities. (I've encountered some disparities in the accuracy of model predictions when training the algorithm with various datasets from FDDB. This variation serves as evidence to highlight the limitation of the Viola-Jones algorithm.)

(2) Frontal face bias: The algorithm was primarily designed to detect frontal or near-frontal faces. Its performance degrades significantly when dealing with faces at different angles or orientations.

(3) Sensitive to lighting conditions: Variations in lighting can impact the algorithm's performance.

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

A:

(1) Adjust Threshold and Polarity: Each classifier for different features may have optimal threshold and polarity settings. Setting all of them to the same value may not yield the best performance. Therefore, just like the process I do in the bonus part, we can individually choose the best threshold and polarity values for each classifier. This approach can lead to higher prediction accuracy by allowing each classifier to adapt its parameters according to the characteristics of the features it's examining.

(2) Feature Selection: The Viola-Jones algorithm uses Haar-like features to encode the image information. If we explore different types of features, such as Local Binary Patterns (LBP), or Histograms of Oriented Gradients (HOG), which may capture more discriminative information than the traditional Haar-like features.

5. 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

A:

There are some face detection method I found in Internet. Among them, Face Detection using Convolutional Neural Networks (CNNs) stands out as one of the most prevalent techniques utilized

Pros:

(1) Automatic Feature Learning: CNNs can automatically learn rich and discriminative features from the input data during training, eliminating the need for manual feature engineering or selection. This is a significant advantage over Adaboost, which relies on hand-crafted Haar-like features. (However, every coin has two sides. This characteristic also leads to negative impact which I mention in the 'Interpretability' below.)

(2) End-to-End Training: CNNs allows for end-to-end training, where the feature extraction and classification stages are jointly optimized, potentially leading to better performance. In contrast, Adaboost has separate stages for feature selection and boosting.

Cons:

(1) Computational Complexity: Training and inference with CNNs can be computationally expensive. One of the reasons is that CNNs need lots of data to be trained. In contrast, the Adaboost algorithm is relatively lightweight.

(2) Interpretability: Unlike Adaboost ensembling weak classifier, CNNs is often criticized for being "black boxes," making it difficult to interpret and understand the learned features and decision-making process. This disadvantage makes us difficult to figure out the way to optimize if our result is not ideal.