HW1 Report

110700045 朱冠霖

Part I. Implementation

Part 1

1. Get the folder location of face and non-face.
2. List all the files in these folders by os.
3. Use endwith() to check that I only get the files I want.
4. Read them by GRAYSCALE style because we want the numpy array shape is (m, n)
5. Append the information of images and its classifications into dataset to create a list of tuples.
6. Return dataset.

```
16      # Begin your code (Part 1)
17      dataset = []
18
19      pathofFace = dataPath+'/face/'
20      pathofNonface = dataPath+'/non-face/'
21
22      for image in os.listdir(pathofFace):
23          if image.endswith('.pgm'):
24              Path = os.path.join(pathofFace, image)
25              img = cv2.imread(Path, cv2.IMREAD_GRAYSCALE)
26              dataset.append((img, 1))
27
28      for image in os.listdir(pathofNonface):
29          if image.endswith('.pgm'):
30              Path = os.path.join(pathofNonface, image)
31              img = cv2.imread(Path, cv2.IMREAD_GRAYSCALE)
32              dataset.append((img, 0))
33
34      # raise NotImplementedError("To be implemented")
35      # End your code (Part 1)
36      return dataset
```

Part 2

1. Get the size of features and dataset.
2. Initialize a list with a length equal to the number of features, where all elements in the list are 0
3. Compute the ϵ_i of all features and $\epsilon_i = \sum_j w_j |h_i(x_j) - y_j|$ and $h_j = 1 \; if \; f_i(x_i) < 0, \; h_j = 0, \; otherwise$.
4. Select the best classifier with lowest ϵ_i.
5. Return the best classifier and lowest ϵ_i.

```
158         # Begin your code (Part 2)
159         n_features = featureVals.shape[0]
160         n_dataset = featureVals.shape[1]
161         errorRate = np.zeros(n_features)
162 
163         for i in range(n_features):
164             for j in range(n_dataset):
165                 errorRate[i] += weights[j] * \
166                     abs((1 if featureVals[i][j] < 0 else 0)-labels[j])
167         bestError = float('inf')
168         bestClf = None
169         for i in range(n_features):
170             if errorRate[i] < bestError:
171                 bestClf = WeakClassifier(features[i])
172                 bestError = errorRate[i]
173         # raise ImportError("To be implemented")
174         # End your code (Part 2)
175         return bestClf, bestError
```

Part 3

1. Read the txt file and convert to a list.
2. Because the first row in the txt file has two data, one is the image name and the other is number of heads, I can use this to read the image and get every square's parameter convert to tuple and append to a list.
3. Read the image in two-way, grayscale and original .
4. To convert an image into a 19*19 grayscale format, the grayscale image and the parameter of squares should be utilized.
5. To determine if an image contains a face, utilize clf.classify. If a face is detected, the prediction value will be 1 and draw a green bounding box. If not, draw a red bounding box.
6. Show the image.

```
    # Begin your code (Part 4)
    Data = list(open(dataPath, "r"))
    line = 0
    while line < len(Data):
        Name, num = map(str, Data[line].split())
        line += 1
        people = []
        for k in range(int(num)):
            people.append(tuple(map(int, Data[line].split())))
            line += 1
        img = cv2.imread(os.path.join("data/detect/", Name))
        imgGray = cv2.imread(
            os.path.join("data/detect/", Name), cv2.IMREAD_GRAYSCALE)
        for person in people:
            x, y, w, h = person
            faceRegion = imgGray[y:y+h, x:x+w]
            resizedFace = cv2.resize(
                faceRegion, (19, 19), interpolation=cv2.INTER_LINEAR)
            prediction = clf.classify(resizedFace)

            if prediction == 1:
                cv2.rectangle(img, (x, y), (x+w, y+h),
                              color=(0, 255, 0), thickness=3)
            else:
                cv2.rectangle(img, (x, y), (x+w, y+h),
                              color=(0, 0, 255), thickness=3)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        fig, ax = plt.subplots()
        ax.axis('off')
        ax.imshow(img)
        plt.show()
    # raise NotImplementedError("To be implemented")
    # End your code (Part 4)
```

Part 6

1. I turn featureVals to 1/0 if the featureVals <0, it will be 1, otherwise, it will be 0.
2. Initialize the bestError to infinite and calculate the error for each classifier $(\epsilon_i = \sum_j w_j |h_i(x_j) - y_j|$ and $h_j = 1 \; if \; f_i(x_i) < 0, \; h_j = 0, \; otherwise, and \; convert \; \epsilon_i \; to \; {\epsilon_i}^{1/\sum_j |h_i(x_j) - y_j|}$
3. Select the best classifier with lowest ϵ_i.
4. Return the best classifier and lowest ϵ_i.

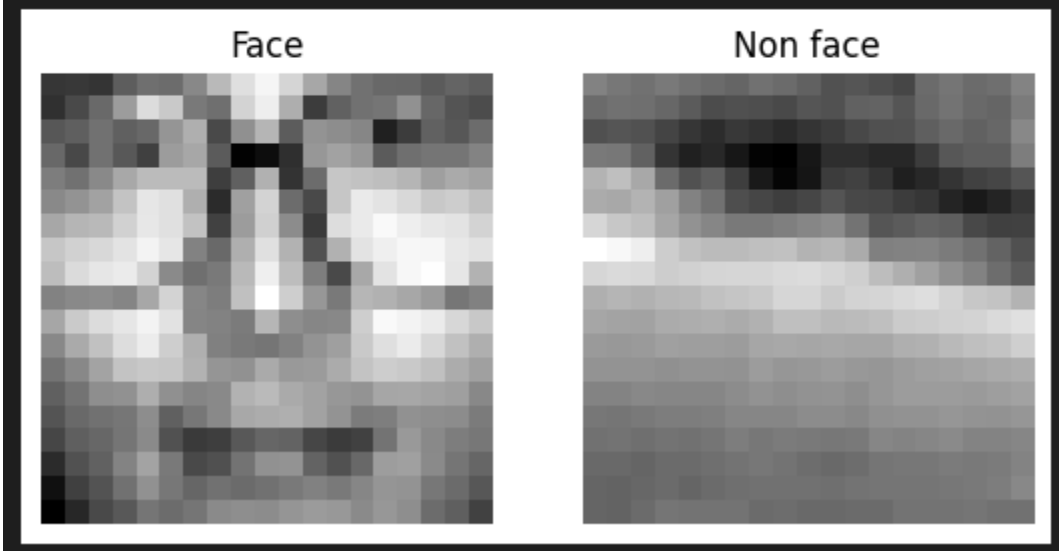
```
180         n_features = featureVals.shape[0]
181         n_dataset = featureVals.shape[1]
182         featureVals=np.where(featureVals<0 , 1, 0)
183         bestError=float('inf')
184         cnt=0
185         for i in range(n_features):
186             error=0.0
187             for j in range(n_dataset):
188                 error=error*weights[j]*abs(featureVals[i][j]-labels[j])
189                 cnt=cnt+abs(featureVals[i][j]-labels[j])
190             error=error**(1/cnt)
191             if error<bestError:
192                 bestError=error
193                 bestClf=WeakClassifier(features[i])
194         return bestClf, bestError
195
196         # End your code (Part 6)
```

Part II. Results & Analysis

Part 1

After doing part 1, I will get the following results.

```
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

After doing part 2, I will get the following results.

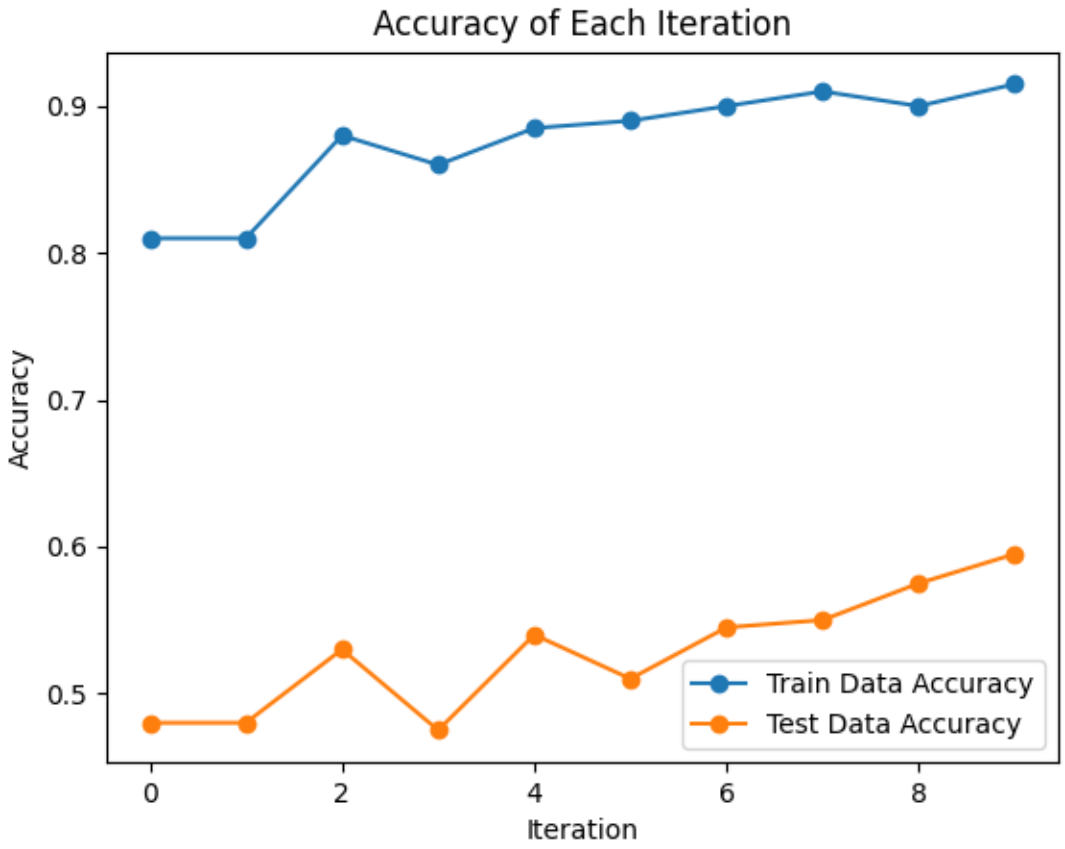
```
Start training your classifier
Computing integral images
Building features
Applying features to dataset
Selecting best features
Selected 5171 potential features
Initialize weights
Run No. of Iteration: 1
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(8,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

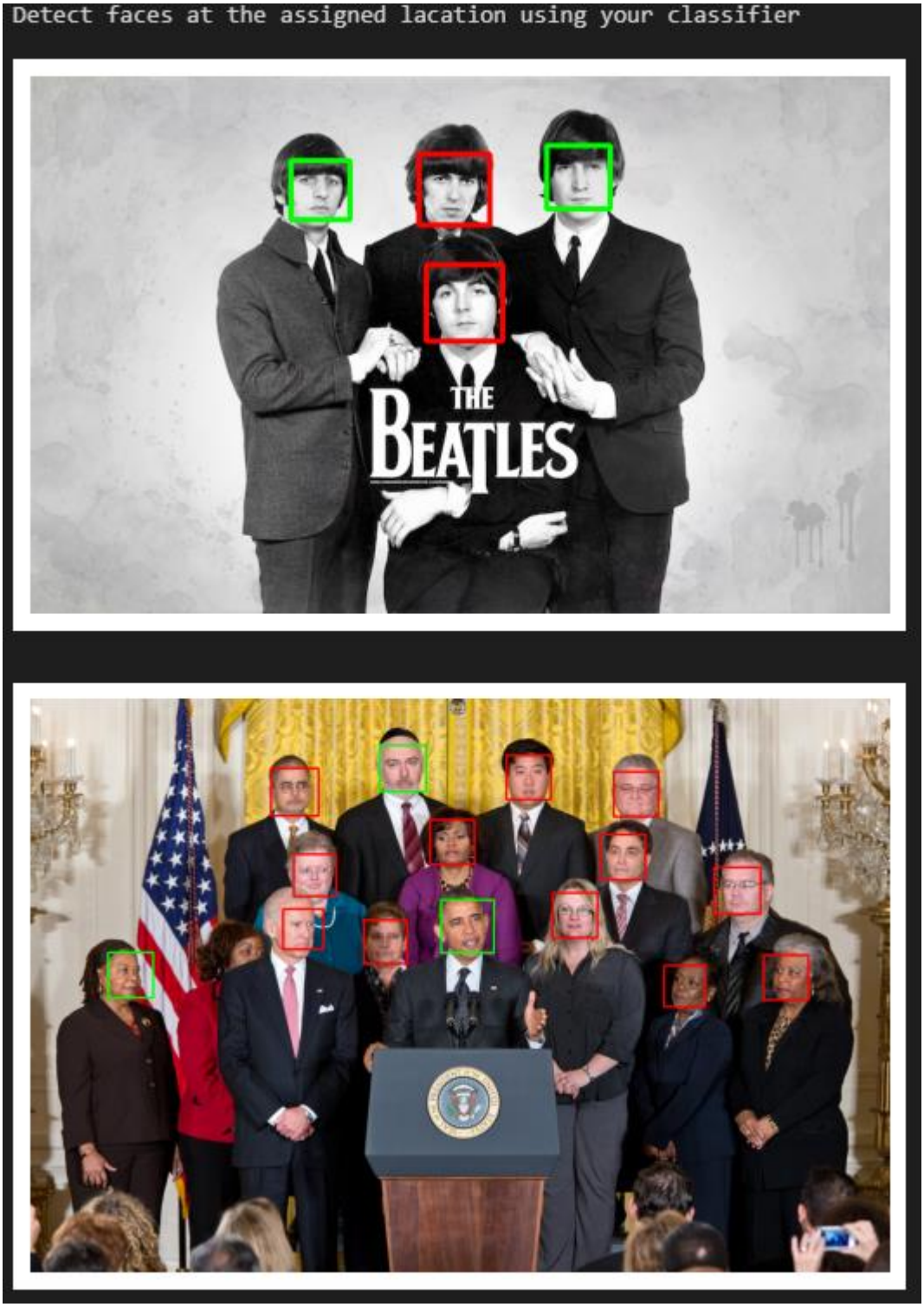
Change the parameter t from 1 to 10, I get the following data, and I plot the data into a line chart.



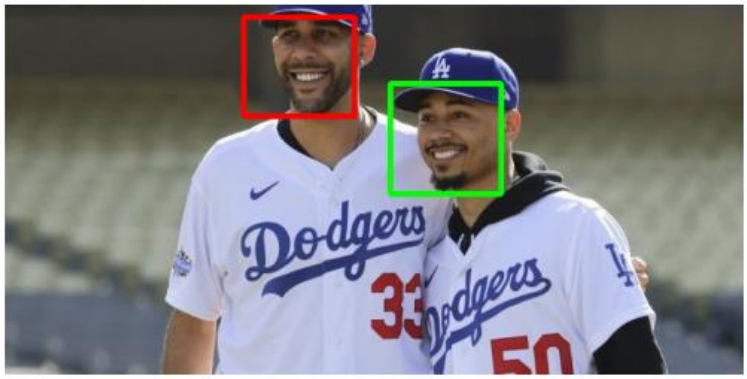
With an increase in T, there is a gradual improvement in the accuracy of the train data. This is due to the classifier being trained on this particular set of data. However, the real test lies in the ability of the classifier to accurately recognize faces in general images beyond the training data. The test data allows us to evaluate the effectiveness of the trained machine. If the results of the test data are not satisfactory, it indicates that the machine needs to be trained on more data and use a better way to find the classifier.

Part 4

After doing part 4, I will get the following images.

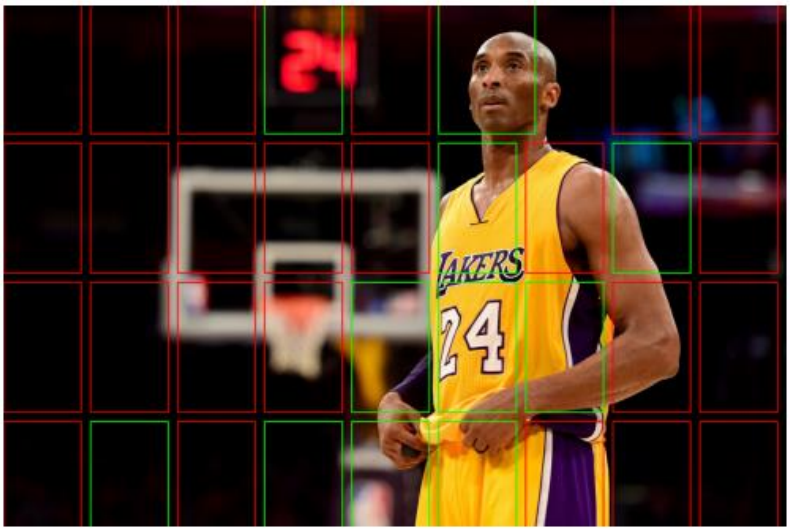


Detect faces on your own images



Part 5

At part 5, I used my own images to test, and the following images after part 5 is done.



Based on the results obtained from these images, it can be concluded that the machine cannot be used to recognize faces in general images. Face may be detected to non-face; non-face may be detected to face.

Part III. Answer the questions

1. Please describe a problem you encountered and how you solve it.
   - When I implement part 1, I cannot run my program because the raise function. Since I'm a beginner of Python , I don't know where the differences between Python and C/C++ are, I spent a lot of time to learn some basic Python and asked TA for help.
   - I don't use grayscale so my numpy array is 3 dimensions. And I referenced from the Internet, I solved it.
   - At first, I wanted to use the ''pyautogui'' package to get the parameters of square with my own image. But it is too complicated, so I used Microsoft Paint to get the pixel point.
2. What are the limitations of Viola-Jones' algorithm?
   - False positives: The algorithm may sometimes detect non-face regions as faces, especially if the image has complex backgrounds or contains objects with face-like features. This can lead to false alarms and reduce the accuracy of the detector.
   - Occlusion: The algorithm may fail to detect faces that are partially or fully occluded, such as faces with sunglasses, hats, or facial hair. This can reduce the overall accuracy of the detector.
   - Lighting and contrast: The algorithm may have difficulty detecting faces in images with poor lighting or low contrast, which can affect the accuracy of the detector.
3. Based on Viola-Jones' algorithm, how to improve the accuracy except increasing the training dataset and changing the parameter T?
   - Develop an effective method to identify the classifier, enabling the machine to utilize more critical classifiers.
4. Please propose another possible face detection method(no matter how good or bad, please discuss the pros and cons of the idea you proposed, compared to the Adaboost algorithm.)
   - After searching the Internet, I found another algorithm that can detect face, that is Convolutional Neural Networks (CNNs), a Deep Learning-based method.
     Pros:
     - High Accuracy: CNN-based face detection methods have achieved state-of-the art performance on face detection benchmarks.
     - Robustness: CNN-based face detection methods are robust to variations in pose, lighting, and occlusion.

     Cons:

- Training Data: CNN-based face detection methods require a large amount of training data, which can be difficult to obtain, especially for specific applications.
- Computationally Expensive: CNN-based face detection methods require a significant number of computational resources, which can make them slower than traditional methods.

Compared to Adaboost algorithm, CNN-based face detection methods have higher accuracy and more robust variations in pose, lighting, and occlusion. However, they require more training data and are computationally more expensive.