Homework 1 Report

Part I. Implementation (6%):

➢ Part 1:

```
# Begin your code (Part 1)
"""
1. Create a list named "dataset" to store the tuples.
2. For every image in the "face" folder, read it in grayscale so that
   the resulting numpy array will have only two dimensions.
3. If the image exists, append it to dataset with classification "1"
4. As for non-face images, the procedure is similar, while their classification should be "0"
"""
dataset = []

root = str(dataPath)+ "/face/"
for image in os.listdir(root):
    img = cv2.imread(os.path.join(root,image),cv2.IMREAD_GRAYSCALE)
    if img is not None:
        dataset.append( (img,1) )

root = str(dataPath)+ "/non-face/"
for image in os.listdir(root):
    img = cv2.imread(os.path.join(root,image),cv2.IMREAD_GRAYSCALE)
    if img is not None:
        dataset.append( (img,0) )
# End your code (Part 1)
```

➢ Part 2:

```
# Begin your code (Part 2)
"""
1. Initialize a numpy array named "h_array" of shape (len(features), len(dataset))
2. By the definition of Haar-like features given in Lecture 3 PPT,
   h_array[j,i] = 1 if featureVals[j,i] < 0, h_array[j,i] = 0 otherwise.
3. Calculate ε of each classifer.
4. Find the lowest ε and the corresponding classifer.
"""
feature_num,data_num = featureVals.shape

h_array = np.zeros((feature_num, data_num))
for j in range(feature_num):
    for i in range(data_num):
        if featureVals[j,i] < 0:
            h_array[j,i] = 1
        else:
            h_array[j,i] = 0

error = np.zeros(feature_num)
for j in range(feature_num):
    for i in range(data_num):
        error[j] += weights[i] * abs(h_array[j,i] - labels[i])

bestClf = WeakClassifier(features[0])
bestError = error[0]
for j in range(1,feature_num):
    if error[j] < bestError:
        bestError = error[j]
        bestClf = WeakClassifier(features[j])
# End your code (Part 2)
```

➢ Part 4:

```
# Begin your code (Part 4)
"""
1. Read every lines in the detectData.txt into the list "line".
2. For each image, first get its name and the number of detected area.
3. Read in the image, beside the original one, we also need to
   read in another in grayscale so that it has the same format as training data.
4. For each detected area, get its coordinates of the top-left corner, width, and height.
5. Cut and resize the area in order to have the same format as training data.
6. Call clf.classify() to detect the area, draw a green rectangle if it's a face,
   otherwise draw a red rectangle.
7. Show the outcome.
"""
with open(dataPath , 'r') as file:
    line = file.readlines()
    count = 0
    while count < len(line):
        name, num = map(str , line[count].split())

        img_ori = cv2.imread("data/detect/" + name)
        img = cv2.imread("data/detect/" + name , cv2.IMREAD_GRAYSCALE)

        for i in range(int(num)):
            count += 1
            info = list(map(int , line[count].split()))
            x1, y1 = info[0], info[1]
            x2, y2 = x1 + info[2], y1 + info[3]
            face = img[y1:y2 , x1:x2]
            resized_face = cv2.resize(face , (19, 19))

            if clf.classify(resized_face) == 1:
                cv2.rectangle(img_ori, (x1, y1), (x2, y2), (0, 255, 0), thickness=2)
            else:
                cv2.rectangle(img_ori, (x1, y1), (x2, y2), (0, 0, 255), thickness=2)

        plt.imshow(cv2.cvtColor(img_ori , cv2.COLOR_BGR2RGB))
        plt.show()
        count += 1
# End your code (Part 4)
```

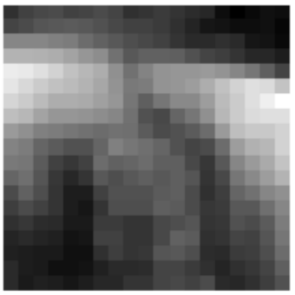
Part II. Results & Analysis (12%):

➢ 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
```

Face

Non face



➢ Part 2:

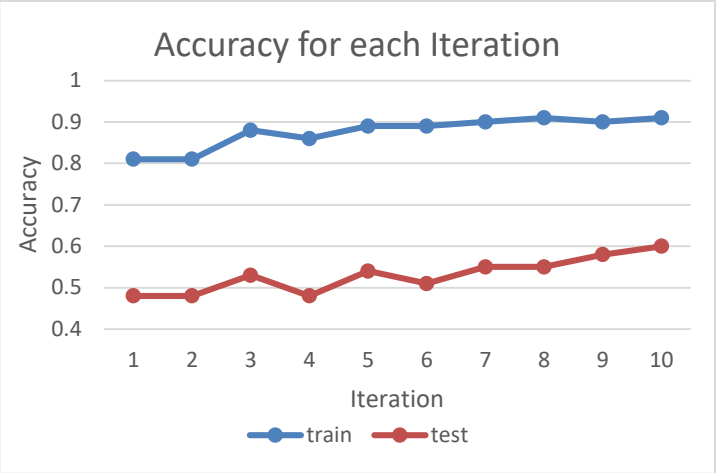
```
Start training your classifier
Computing integral images
Building features
Applying features to dataset
Selecting best features
Selected 5171 potential features
Initialize weights
Run No. of Iteration: 1
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(8,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

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)
```

```
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
```

➢ Part 3:



From the line chart, we can observe that both training and testing data have higher accuracy rate as the iteration increases. However, there is still a big gap between their performance. Training data has at most 92% accuracy while testing data has only around 60%.

We can conclude that our trained model fits well with the training data, but it's still not general enough to deal with new situation, that is, testing data.

If looking at the right end of this chart, we can see that the accuracy of training data seems to oscillate around 0.9, while that of testing data keeps growing. I guess maybe training data has reached its limitation, but chances are that the testing data can performance better under more than ten iterations.

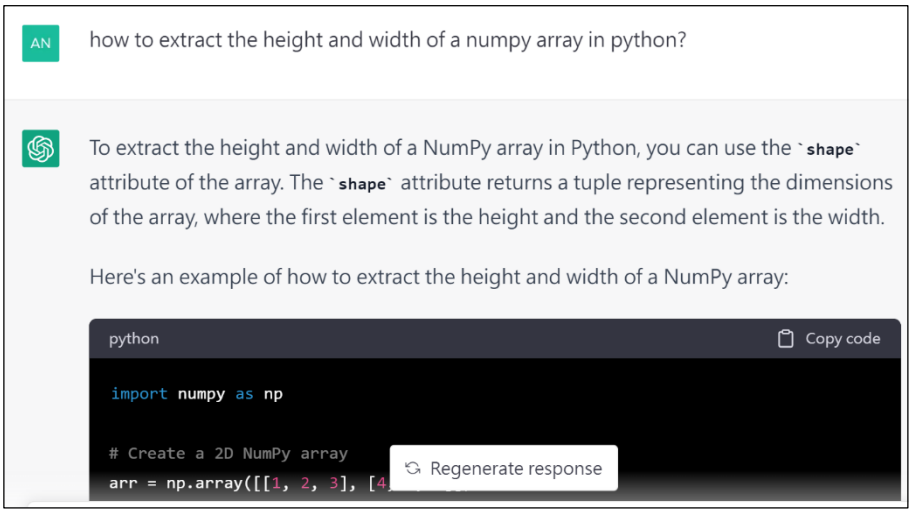
➢ Part 4:

(t = 10)

- **Part 5:**
  (t=10)

Part III. Answer the questions (12%):

1. Please describe a problem you encountered and how you solved it.
   I. The most serious problem I have encountered was that I never learned how to write Python. My solution was to ask ChatGPT with questions beginning like: “how to...”. It then suggested me which function I might need.



   II. My Colab cannot respond to new code immediately, that is, it still executed the former one even if I updated the code. If I did nothing, it took a day waiting for it to connect with the updated code, which was very time-consuming. At last, I found I could solve this problem by pressing the bottom “restart execution”. Then I could keep working on my project!

2. What are the limitations of the **Viola-Jones’ algorithm**?
   I. Can only do binary classification (ex. face or non-face)
   II. Cannot adapt to new conditions (ex. side face, overexposed and underexposed image)
   III. Have a hard time detecting part of the object (ex. people wearing a mask or glasses)

3. Based on **Viola-Jones’ algorithm**, how to improve the accuracy except changing the training dataset and parameter T?
   I. Include various features, as discriminative as possible
   II. Pre-adjust the test data so that they are in consistent condition (ex. exposure)
   III. When plotting the detected area, put the boundary close enough to the object, avoid including too many irrelevant information.
   For example, after I replotted the box closer to the face, the accuracy raised.

Before	After
	

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 can use "Bagging".

The idea of bagging is to create a few subsets by randomly sampling with replacement from the original training data. Each subset then trains their model independently. Finally, combine all results to produce a stronger classifier. In the face detection case, which is a binary classification, use "voting" to combine those weaker classifiers.

Pros:

I. reduce overfitting, so that the final model has better generalization
II. reduce the variance of the final model, because it's constructed by averaging the predictions of many models

Cons:

I. computationally expensive when dealing with large datasets, since it has to combine many models