Homework 1

Part I. Implementation (6%):

code for part. 1

```
# Begin your code (Part 1)
"""
load face & non-face images through the followings steps
1. create an empty tuple
2. store all the name of the pictures in folder 'face' into the variable 'images'
3. for all the images, read the image through cv2.imread(), put it into a np array, and append it into dataset with label 1
4. do the same thing for non-face as it was done in 2&3
"""
dataset = tuple()                                                    # step 1
images = os.listdir(dataPath+'/face')                                # step 2
for image in images:                                                 # step 3
  image_read = cv2.imread(dataPath+'/face/' + image, cv2.IMREAD_GRAYSCALE)   # read through imread() with the name of the file
  image_array = np.array(image_read)                                 # store the image into np array
  dataset = dataset + ((image_array,1),)                             # add label 1 and put it into the dataset

images = os.listdir(dataPath+'/non-face')                            # step 4
for image in images:
  image_read = cv2.imread(dataPath+'/non-face/' + image, cv2.IMREAD_GRAYSCALE)
  image_array = np.array(image_read)
  dataset = dataset + ((image_array,0),)                             # with label 0
# End your code (Part 1)
```

code for part. 2

```
150         # Begin your code (Part 2)
151         '''
152           Store the 1 or 0 for each feature of all images, indicating the prediction
153           for certain feature for each picture.
154           Calculate the error for each feature iteratively.
155           The error is calculated by summing the weight which was wrongly predicted.
156           Pick the feature with the least error and return with the error.
157         '''
158         bestf = features[0]
159         bestError = 1
160         featureAmount = len(features)                            # number of features
161         dataAmount = len(weights)                                # number of datas
162
163         featureValsTemp = np.empty((len(features), len(weights)))
164         for i in range (len(features)):
165           classifier = WeakClassifier(features[i])               # give predict results
166           for j in range (len(weights)):
167             featureValsTemp[i][j] = classifier.classify(iis[j])
168
169         for i in range (featureAmount):                          # for all features
170           sum = 0
171           for j in range (dataAmount):                           # sum to get error
172             sum += weights[j]*abs(featureValsTemp[i][j]-labels[j])
173           if(sum<bestError):                                     # update if error smaller
174             bestf = features[i]
175             bestError = sum
176         bestClf = WeakClassifier(bestf)                          # return as classifier
177         # End your code (Part 2)
178         return bestClf, bestError
```

code for part. 4

```python
    # Begin your code (Part 4)
    '''
    read the file, find the corresponding image of face & other data as described in the .txt.
    read the images, choose the portion needed, resize it, and convert into gray-scale.
    use the classifier to predict whether the image is a face.
    '''
    file= open(dataPath, 'r')                                                    # open .txt
    contents = file.read()
    elements = contents.split()
    i = 0
    while(i < len(elements)):
        imgName = elements[i]
        img = cv2.imread('data/detect/'+imgName)                                 # read image
        i += 1
        numberOfSquares = int(elements[i])
        i += 1
        for j in range(numberOfSquares):
            x1 , x2 = int(elements[i+4*j]), int(elements[i+4*j]) + int(elements[i+4*j+2])     # area needed
            y1 , y2 = int(elements[i+4*j+1]), int(elements[i+4*j+1]) + int(elements[i+4*j+3])
            cutImg = img[y1:y2,x1:x2]                                            # cut the area
            imgfordetect = cv2.resize(cutImg, (19, 19), interpolation=cv2.INTER_NEAREST)      # resize it
            imgfordetect = cv2.cvtColor(imgfordetect, cv2.COLOR_BGR2GRAY)
            image_array = np.array(imgfordetect)
            isFace = clf.classify(image_array)                                   # use classifier to predict
            if(isFace):                                                          # draw rectangle
              cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), thickness=2)
            else:
              cv2.rectangle(img, (x1, y1), (x2, y2), (0, 0, 255), thickness=2)
        i += numberOfSquares*4
        cv2.imwrite(imgName[:-4]+'.jpg', img)                                    # save the drawn img
    # End your code (Part 4)
```

Part II. Results & Analysis (12%):

- T analysis (part 3 in the HW)

Screenshot for results of all Ts

T = 10

```
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 9,
 2, 2), RectangleRegion(2, 11, 2, 2)], negative regions=[RectangleRegion(2, 9, 2, 2), RectangleRegion(4, 1
1, 2, 2)]) with accuracy: 137.000000 and alpha: 0.811201

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)
```

T = 9

```
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 4
, 1, 1)], negative regions=[RectangleRegion(9, 4, 1, 1)]) with accuracy: 152.000000 and alpha: 0.707795

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)
```

T = 8

```
Run No. of Iteration: 8
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(12, 1
1, 5, 1)], negative regions=[RectangleRegion(12, 12, 5, 1)]) with accuracy: 72.000000 and alpha: 0.685227

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)
```

T = 7

```
Run No. of Iteration: 7
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 2,
 10, 2)], negative regions=[RectangleRegion(5, 4, 10, 2)]) with accuracy: 145.000000 and alpha: 0.719869

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)
```

T = 6

```
Run No. of Iteration: 6
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(7, 3,
 3, 8)], negative regions=[RectangleRegion(4, 3, 3, 8)]) with accuracy: 78.000000 and alpha: 0.769604

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)
```

T = 5

```
Run No. of Iteration: 5
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 8
, 1, 1)], negative regions=[RectangleRegion(9, 8, 1, 1)]) with accuracy: 155.000000 and alpha: 0.924202

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)
```

T = 4

```
Run No. of Iteration: 4
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 14
, 8, 2)], negative regions=[RectangleRegion(4, 16, 8, 2)]) with accuracy: 153.000000 and alpha: 0.908680

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)
```

T = 3

```
Run No. of Iteration: 3
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(16, 1
6, 1, 2)], negative regions=[RectangleRegion(15, 16, 1, 2)]) with accuracy: 155.000000 and alpha: 1.011738

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)
```

T = 2

```
Run No. of Iteration: 2
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 8,
 2, 9)], negative regions=[RectangleRegion(2, 8, 2, 9)]) with accuracy: 156.000000 and alpha: 1.286922

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)
```

T = 1

```
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)
```

The following graph is the relationship between T and the accuracy for both training set and the testing set.

From the graph, we can know that the accuracy for each two sets seems to have a positive relationship with T, and that is the thing we anticipated. One intersting thing is that when T is bigger than 5, the “false negative rate” in the training data becomes 0, which means that if the algorithm say a picture is not a face, it is not a face(only in training dataset). This might be caused by our traing data selection or the parr-feature we chose.

After trying 1 to 10, I made some experiments with bigger Ts.

Here are the records

```
Evaluate your classifier with training dataset
False Positive Rate: 13/100 (0.130000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 187/200 (0.935000)

Evaluate your classifier with test dataset
False Positive Rate: 46/100 (0.460000)
False Negative Rate: 41/100 (0.410000)
Accuracy: 113/200 (0.565000)
```

T = 15

```
Evaluate your classifier with training dataset
False Positive Rate: 6/100 (0.060000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 194/200 (0.970000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 36/100 (0.360000)
Accuracy: 115/200 (0.575000)
```

T = 20

```
Evaluate your classifier with training dataset
False Positive Rate: 3/100 (0.030000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 197/200 (0.985000)

Evaluate your classifier with test dataset
False Positive Rate: 50/100 (0.500000)
False Negative Rate: 42/100 (0.420000)
Accuracy: 108/200 (0.540000)
```

T = 30

```
Evaluate your classifier with training dataset
False Positive Rate: 0/100 (0.000000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 200/200 (1.000000)

Evaluate your classifier with test dataset
False Positive Rate: 46/100 (0.460000)
False Negative Rate: 46/100 (0.460000)
Accuracy: 108/200 (0.540000)
```

T = 50

```
Evaluate your classifier with training dataset
False Positive Rate: 0/100 (0.000000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 200/200 (1.000000)

Evaluate your classifier with test dataset
False Positive Rate: 43/100 (0.430000)
False Negative Rate: 43/100 (0.430000)
Accuracy: 114/200 (0.570000)
```

T = 100

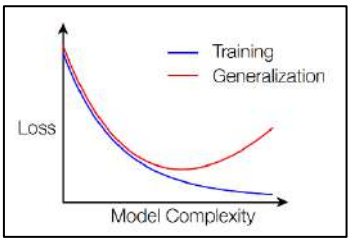
```
Evaluate your classifier with training dataset
False Positive Rate: 0/100 (0.000000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 200/200 (1.000000)

Evaluate your classifier with test dataset
False Positive Rate: 47/100 (0.470000)
False Negative Rate: 42/100 (0.420000)
Accuracy: 111/200 (0.555000)
```

T = 200

Here is the graph

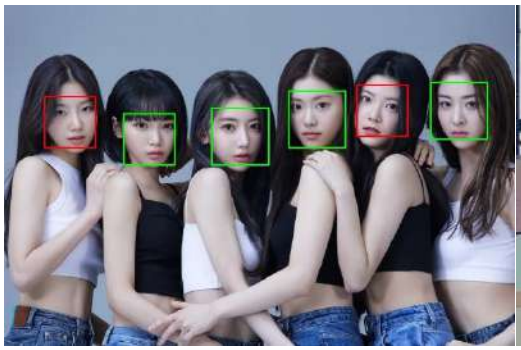
From the graph, you can see that the accuracy of the training dataset is 100% when we have a large T. However, the accuracy of the testing dataset drops a little. The decrease can be explained by the figure below, you can see that when the model is not complex (T is small), when we make it more complex (increase T), the loss for training data and gerneral data both decrease (when T<10, both accuracies increased), but when the model is too complex (T is large), it starts to overfit, and the loss for general cases starts to increase (accuracy for testing set drops).



- Photo detection results

photos given(part 4 in HW)


Photos I chose(part 5 in HW)




In this part, I use T=10 as the classifier to predict the result. I tried to find the characteristic which might lead our classifier to make mistakes, but it seems that there our no such characteristic. I think it is just a matter of the position and the size of the rectangle you choose. If your classifier made a mistake, you can get the correct result with a little fine-tune.

Part III. Answer the questions (12%):

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

   At first, I tried to only write the code in part 2 without reading all the .py, so I had troubles dealing with types, parameters, and the functions. It was my first time writing a python code written in the OOP style, and I spent quite a few time reading and searching what all the thing means. Upon all the things, the two classify() function confused me the most. Due to the lack of experience, when I saw the classify() in the code, I look for the classify() function directly without noticing which object the classify() function is under, and that confused me a lot cause its not the way I think the algorithm should be like… The way I solve it was stop being careless and everything went fine after I found the mistake I made.

2. **What are the limitations of the Viola-Jones' algorithm?**
   - ✓ The accuracy will drop significantly if the face you choose was tilted or turned around because the algorithm mainly support frontal face detection.
   - ✓ Haar-like features are sensitive to changes in lighting conditions. Since Viola-Jones's algorithm is based on harr-like features, it is prone to lighting changes.

3. **Based on Viola-Jones' algorithm, how to improve the accuracy except changing the training dataset and parameter T?**
   We can change the harr-like features, or maybe we can separate the training set into training set and validation set and use these two sets to acquire better result.

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**
   Here are my thoughts:
   1. First, find the lines of the face, and we can get a matrix giving only the lines of the face (identity where the lines is through the change of color, since there might be a slight change on the color of your skin, maybe we should add a buffer for that).
   2. After that, we can let the machine to learn how the distribution of lines of "an element of a face" should be like. For example, we can train the identifier of nose, eyes, mouth, and the shape of a face.
   3. Then, we can identify these elements separately, and learn the distribution of these elements. The result can be determined by the voting of the elements, or maybe use some weights and add up the distribution result and the detection of elements.

   The strength of the algorithm I proposed is the accuracy. Since the computer leared lots of things (combine all the things together, a distribution test), my algorithm might get better accuracy than the adaboost algorithm if it is well-desinged, and it maybe this method can be used for tilted face in some way.

   The shortcoming of the algorithm is that it takes lots of computation since it must train lots of things, which will comsume more time and power. Finding the positions of the elements is also a difficulty, so it's harder to implement than the adaboost algorithm.