HW 1: Face Detection Report

Student ID: 111550100 Name: 邱振源

Part I. Implementation

Part 1-1:

```python
# Begin your code (Part 1-1)
"""
1. Initialize the empty list (tarining_dataset, testing_dataset).
2. Use os.listdir() and for loop to traverse all the image.
3. Use os.path.join() to generate the file path of each image file.
4. Use cv2.imread() to read the image, and change it to gray scale image.
5. Append the image and its classification into dataset.
6. Since there are floders(face, non-face) in train and test, so do the above 4 times.
7. Return the dataset.
"""
tarining_dataset = []
testing_dataset = []

for file in os.listdir('data/data_small/train/face'):
    image_path = os.path.join('data/data_small/train/face', file)
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is not None:
        image_array = np.array(image)
        tarining_dataset.append((image_array, 1))

for file in os.listdir('data/data_small/train/non-face'):
    image_path = os.path.join('data/data_small/train/non-face', file)
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is not None:
        image_array = np.array(image)
        tarining_dataset.append((image_array, 0))

for file in os.listdir('data/data_small/test/face'):
    image_path = os.path.join('data/data_small/test/face', file)
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is not None:
        image_array = np.array(image)
        testing_dataset.append((image_array, 1))

for file in os.listdir('data/data_small/test/non-face'):
    image_path = os.path.join('data/data_small/test/non-face', file)
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is not None:
        image_array = np.array(image)
        testing_dataset.append((image_array, 0))

dataset = (tarining_dataset, testing_dataset)

# End your code (Part 1-1)
```

Part 1-2:

```python
# Begin your code (Part 1-2)
"""
1. Use `height` and `width` to be the height and width of the image.
2. Use while loop to check the nonface_box doesn't fully cover the face part.
3. Use np.random.randint() to check the reglon I cut is 19*19.
4. Use .copy() to copy the image.
"""
height, width = img_gray.shape

nonface_box = None

while nonface_box is None or any((not(nonface_box[0][0] > face_box[1][0] or nonface_box[1][0] < face_box[0][0] or nonface_box[0][1] > face_box[1][1] or nonface_box[1][1] < face_box[0][1])) for face_box in face_box_list):
    w = np.random.randint(19, width)
    h = np.random.randint(19, height)
    x = np.random.randint(0, width - w)
    y = np.random.randint(0, height - h)

    nonface_box = ((x, y), (x + w, y + h))
left_top, right_bottom = nonface_box
nonface_img_crop = img_gray[left_top[1]:right_bottom[1], left_top[0]:right_bottom[0]].copy()
# End your code (Part 1-2)
```

Part 2:

```
# Begin your code (Part 2)
"""
1. Initialize bestClf and bestError to none and infinity.
2. For every column of featureVals(each feature), initialize tmp=0.
3. Check the relation between every raw of the featureVals and labels:
    If (featureVals>=0, labels=1 or featureVals<0, label=0), add this weights to tmp.
4. After traversing the whole row, if tmp is less than bestError,
    change bestError to tmp and bestClf to the weakclassifier of the features.
5. Return the bestClf and bestError.
"""
bestClf = None
bestError = float('inf')

for i in range(len(features)):
    tmp = 0
    for j in range(len(iis)):
        if(featureVals[i][j]>=0 and labels[j]==1):
            tmp = tmp + weights[j]
        elif(featureVals[i][j]<0 and labels[j]==0):
            tmp = tmp + weights[j]
    if tmp<bestError:
        bestError = tmp
        bestClf = WeakClassifier(features[i])

# End your code (Part 2)
```

Part 4:

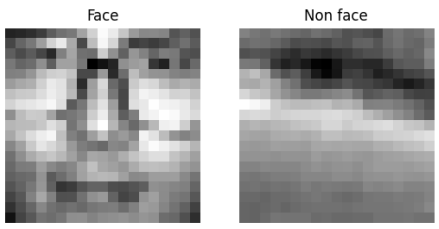
```
# Begin your code (Part 4)
"""
1. Read the txt file.
2. Use cv2.imread() and os.path.join() to open the image in the .txt file
3. Read the following lines of the .txt file and get the position of a face.
4. Change the image to a gray scale image.
5. Resize the image to the size of 19x19.
6. Use clf.classify() to detect faces.
   If it is face, the color of the box will be green, otherwise it will be red.
"""
with open(dataPath, 'r') as file:
  lines = file.readlines()

times = 0

for line in lines:
  if(times == 0):
    img_name, number_of_face = line.split()
    image = cv2.imread(os.path.join("data/detect", img_name))
  else:
    left, upper, width, height = map(int, line.split())
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    if clf.classify(cv2.resize(gray_image[upper:upper+height, left:left+width], (19, 19))):
      cv2.rectangle(image, (left, upper), (left+width, upper+height), (0, 255, 0), 3)
    else:
      cv2.rectangle(image, (left, upper), (left+width, upper+height), (0, 0, 255), 3)
  times = times + 1
  if(times == int(number_of_face)+ 1):
    times = 0
    cv2.imshow('image', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
# End your code (Part 4)
```

Part 6:

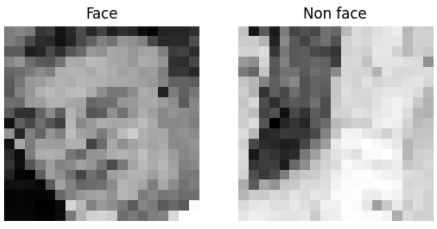
```
# Begin your code (Part 6)
"""
1. Initialize bestClf and bestError to none and infinity.
2. For every column of featureVals(each feature), initialize tmp=0.
3. Check the relation between every raw of the featureVals and labels:
    If (featureVals>=0, labels=1 or featureVals<0, label=0), add this weights*0.9 to tmp.
    Else add the weights*0.1 to tmp.
4. After traversing the whole row, if tmp is less than bestError,
    change bestError to tmp and bestClf to the weakclassifier of the features.
5. Return the bestClf and bestError.
!!
  The main difference from the original approach is that I no longer add or discard all the
  weights when comparing them. Instead, I use multiplication by 0.9 or 0.1 to reduce the
  likelihood of extreme values affecting the comparison of bestError.
!!
"""
bestClf = None
bestError = float('inf')

for i in range(len(features)):
    tmp = 0
    for j in range(len(iis)):
        if(featureVals[i][j]>=0 and labels[j]==1):
            tmp = tmp + weights[j]*0.9
        elif(featureVals[i][j]<0 and labels[j]==0):
            tmp = tmp + weights[j]*0.9
        else:
            tmp = tmp + weights[j]*0.1
    if tmp<bestError:
        bestError = tmp
        bestClf = WeakClassifier(features[i])
# End your code (Part 6)
```

Part II. Results & Analysis

Part 1



The result after load_data_small() is done



The result after load_data_FDDB() is done

Part 2

data small

```
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)
```

The result after selectBest() is done

data FDDB

```
Evaluate your classifier with training dataset
False Positive Rate: 113/360 (0.313889)
False Negative Rate: 43/360 (0.119444)
Accuracy: 564/720 (0.783333)

Evaluate your classifier with test dataset
False Positive Rate: 65/155 (0.419355)
False Negative Rate: 17/155 (0.109677)
Accuracy: 228/310 (0.735484)
```

The result after selectBest() is done

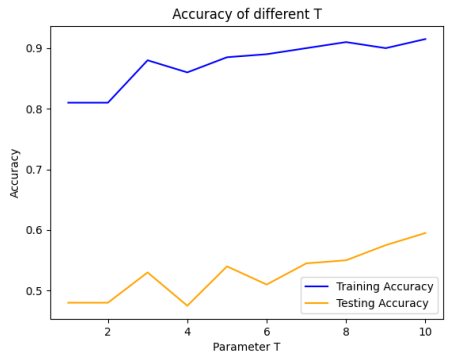
Part 3

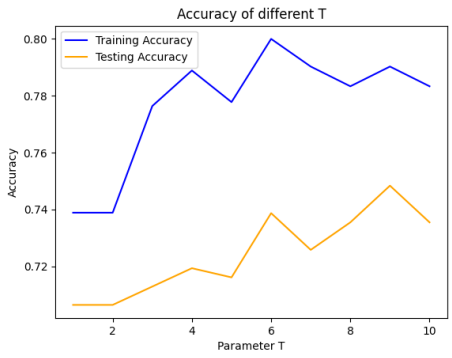
data small

T	Training data accuracy	Testing data accuracy
1	81.0%	48.0%
2	81.0%	48.0%
3	88.0%	53.0%
4	86.0%	47.5%
5	88.5%	54.0%
6	89.0%	51.0%
7	90.0%	54.5%
8	91.0%	55.0%
9	90.0%	57.5%
10	91.5%	59.5%

data FDDB

T	Training data accuracy	Testing data accuracy
1	73.8889%	70.6452%
2	73.8889%	70.6452%
3	77.6389%	71.2903%
4	78.8889%	71.9355%
5	77.7778%	71.6129%
6	80.0000%	73.8710%
7	79.0278%	72.5806%
8	78.3333%	73.5484%
9	79.0278%	74.8387%
10	78.1944%	73.5484%





Based on the results above, we can observe the following two points:

1.As the parameter T increases, the accuracy generally shows an upward trend.

2.The accuracy of the training data is higher than that of the testing data, suggesting that the classifier is trained on the training data.

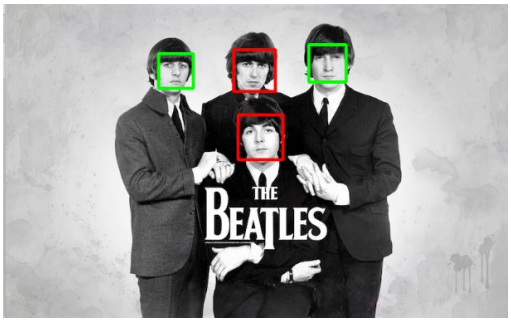
However, comparing the results of the two classifiers trained on different datasets, it can be observed that there is a significant difference between the training data accuracy and testing data accuracy for the "small" dataset. Although the training data accuracy for the "FDDB" dataset is lower, the classifier trained on the "FDDB" dataset still exhibits a testing data accuracy close to that of the training data.

Furthermore, in the "FDDB" dataset, it is evident that increasing the parameter T is not necessarily better, and it is still necessary to select an appropriate value for T based on the results.

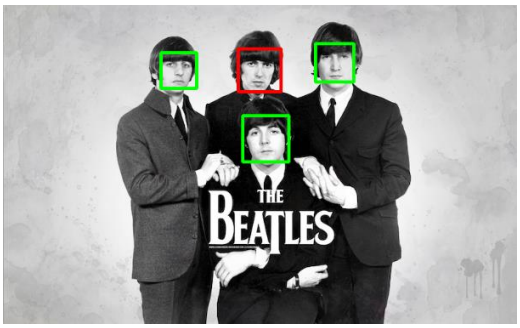
Part 4

The result after finishing the detect() function:

data small



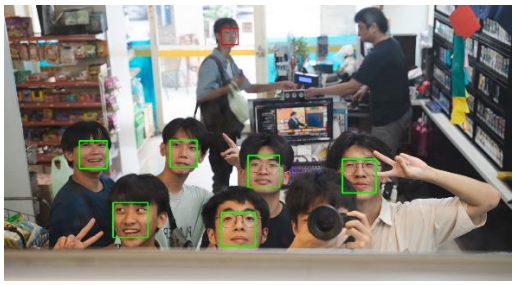
data FDDB



Part 5

I use Microsoft Paint to check the region of face, and use detect() to test:

data small



data FDDB

Part 6

The approach I use in Part 6 is like Part 2. However, this time I no longer add or discard all the weights when comparing them. Instead, I use multiplication by 0.9 or 0.1 to reduce the likelihood of extreme values affecting the comparison of bestError.

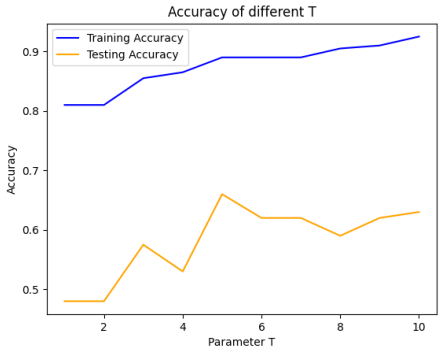
data small

T	Training data accuracy	Testing data accuracy
1	81.0%	48.0%
2	81.0%	48.0%
3	85.5%	57.5%
4	86.5%	53.0%
5	89.0%	66.0%
6	89.0%	62.0%
7	89.0%	62.0%
8	90.5%	59.0%
9	91.0%	62.0%
10	92.5%	63.0%

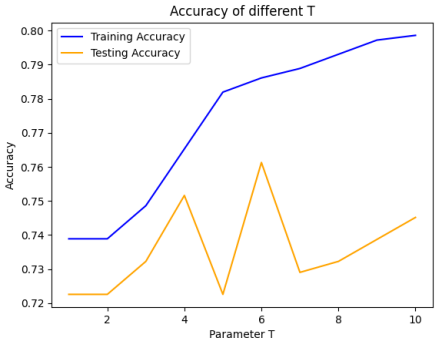
data FDDB

T	Training data accuracy	Testing data accuracy
1	73.8889%	72.2581%
2	73.8889%	72.2581%
3	74.8611%	73.2258%
4	76.5278%	75.1613%
5	78.1944%	72.2581%
6	78.6111%	76.1290%
7	78.8889%	72.9032%
8	79.3056%	73.2258%
9	79.7222%	73.8710%
10	79.8611%	74.5161%

data small



data FDDB



Through this method, it can be observed that there is a slight improvement of accuracy in both the training and testing data compared to the results from Part 2. It can be inferred that the reduced influence of extreme values on the classifier selection leads to better performance of the classifier.

Part III. Answer the questions

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

1. At the beginning, I encountered failures when passing data into the dataset using imread(). After researching online and discussing with classmates, I discovered that using imread() directly reads files as color images by default. It requires specifically using cv2.IMREAD_GRAYSCALE to convert them into grayscale images to meet our requirements.
2. When segmenting non-face images for the "FDDB" dataset, I initially had no idea how to proceed. After thinking for a long time, I came across a hint in the PDF file, which suggested cutting only a small portion of the face area rather than the entire image to generate the non-face region.
3. Initially, I attempted various methods to write the selectBest() function but did not achieve satisfactory results. It was only after diligently studying the class PowerPoint slides and thoroughly reading the function introductions in each .py file in the zip archive that I began to have some ideas for implementing the selectBest() function, ultimately leading to an improvement in accuracy.
4. When initially running the detect() function, I mistakenly assumed that the file was storing coordinates of the top-left and bottom-right corners. This misconception led to spending a lot of time debugging. However, after understanding the meaning of each number in the .txt file, I successfully completed the detect() function.

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

   By following these steps:

   1. Use height and width to represent the height and width of the image.
   2. Use a while loop to ensure that the nonface_box does not fully cover the face region provided in the dataset.
   3. Use np.random.randint() to ensure that the region I crop is 19*19 in size.
   4. Use .copy() to create a copy of the image.

   We can generate "nonface" data by cropping images that have small intersections with the face regions.

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

   After observing the result and searching the Internet, I find that the Viola-Jones’ algorithm has the following limitation:

   1. Sensitivity to variations in lighting conditions:
      Since the algorithm relies on simple rectangular features, which may not be robust to changes in lighting conditions, leading to false positives or false negatives.
   2. Limited detection of non-frontal faces.
   3. Performance degradation with scale and rotation.
   4. Computational complexity:
      The algorithm's feature extraction and selection process can be computationally intensive, especially when using a large number of features or processing high-resolution images.

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

   1. Use the other way to write the selectBest() function:
      Instead of relying solely on the default set of rectangular Haar-like features, custom feature extraction techniques can be explored. This may involve designing more complex features or using different types of feature descriptors to capture more discriminative information about faces.
   2. Post-processing techniques:
      Applying post-processing techniques such as non-maximum suppression to refine the detected face regions can help improve accuracy by reducing false positives and refining the localization of detected faces.

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.**

Convolutional Neural Network (CNN) Approach.

CNNs are deep learning models which designed to automatically and adaptively learn spatial hierarchies of features from the input images. In the context of face detection, a CNN can be trained on a large dataset of labeled face images to learn discriminative features directly from the pixel values.

Pros:

1. End-to-End Learning:
   CNNs learn features directly from the raw input data, eliminating the need for handcrafted feature extraction. This allows CNNs to adapt to a wide range of face variations, including pose, expression, and illumination.
2. Flexibility:
   CNN architectures can be customized and scaled to suit the complexity of the face detection task.

Cons:

1. Data Dependency:
   CNNs typically require a large amount of labeled training data to generalize well to unseen faces. Acquiring and annotating such datasets can be time-consuming and resource-intensive.
2. Computationally Intensive:
   Training CNN models and performing inference on large images can be computationally expensive, especially on resource-constrained devices.

Comparison with Adaboost Algorithm:

1. Adaptability:
   CNNs have the advantage of learning features directly from the data, allowing them to adapt more effectively to variations in facial appearance compared to handcrafted features used in Adaboost.
2. Performance:
   CNNs often achieve state-of-the-art accuracy when provided with sufficient training data.
3. Computational Complexity:
   While CNNs may offer superior accuracy, they are typically more computationally intensive than Adaboost algorithms, especially during training and inference phases.

All in all, while the CNN approach may offer higher accuracy and better adaptability to complex face variations, it comes with the trade-offs of increased computational complexity, and data dependency compared to the Adaboost algorithm.