Homework 1: Face Detection

110550108 施柏江

Part I. Implementation (6%):

↓ Part 1

```python
# Begin your code (Part 1)
# raise NotImplementedError("To be implemented")
"""
1. Create a list 'dataset' to store the data of all images
2. Use 'os.path.join()' to get the path of face and non-face folders
3. Use 'os.listdir()' to get all files in the folder
4. Use 'cv2.imread()' to read the image and
    use 'cv2.IMREAD_GRAYSCALE' to convert to greyscale
5. If current path is 'face', the second element will be 1.
    Otherwise, 0.
6. Finally, add the tuple to the 'dataset'
"""
dataset = []
path = os.path.join(dataPath, 'face')
files = os.listdir(path)
for file in files:
  img = cv2.imread(os.path.join(path, file), cv2.IMREAD_GRAYSCALE)
  dataset.append((img, 1))

path = os.path.join(dataPath, 'non-face')
files = os.listdir(path)
for file in files:
  img = cv2.imread(os.path.join(path, file), cv2.IMREAD_GRAYSCALE)
  dataset.append((img, 0))
# End your code (Part 1)
```

↓ Part 2

```python
        # Begin your code (Part 2)
        # raise NotImplementedError("To be implemented")
        """
        1. Create 'bestError' to store the error of the best weak classifier and
            'bestIndex' to store the index of it
        2. The first loop iterates through every weak classifier
        3. The second loop iterates through every integral image
        4. If featureVals[i][j] < 0, it means that the classifier identifies the image
            as a face. Otherwise, a non-face.
        5. If the image is identified wrongly, add its weight to 'error'
        6. Return the weak classifier with minimum error
        """
        bestError = float('inf')
        bestIndex = -1
        for i in range(len(features)):
            error = 0
            for j in range(len(iis)):
                isFace = (featureVals[i][j] < 0)
                error += weights[j] * (abs(isFace - labels[j]))
            if error < bestError:
                bestIndex = i
                bestError = error
        bestClf = WeakClassifier(features[bestIndex])
        return bestClf, bestError
        # End your code (Part 2)
```

↓ Part 4

```
# Begin your code (Part 4)
# raise NotImplementedError("To be implemented")
"""
1. Create 'locations' to store the location and size of each face,
   'filenames' to store the name of each image,
   and 'line_cnt' to store the number of faces of each image
2. Use 'open()' to open 'detectData.txt'
3. Use 'readlines()' to read in the information line by line
4. Use 'strip()' to remove any leading and trailing spaces
5. Use 'split()' to split the line into a list
6. If the first item in the list is a number,
   it means that this line is the information about location.
   Otherwise, it is about filename.
7. Use 'os.path.join()' to get the path of the image
8. Use 'cv2.imread()' to read the image
9. Because the image has to be grey when classifying,
   use 'cv2.IMREAD_GRAYSCALE' to convert it to greyscale
10. Because we have to draw green and red rectangles in the final result,
   create 'img_ori' to store the original image
11. Create 'crop_img' to store the cropped image,
   and then resize it to (19, 19)
12. After trying all of the interpolation methods,
   'cv2.INTER_NEAREST' perform the best, so I choose it.
13. If the cropped image is identified as a face, draw a green rectangle on it.
   Otherwise, draw a red one.
"""
```

```
GREEN = [0, 255, 0]
RED = [0, 0, 255]

locations = []
filenames = []
line_cnt = []

with open(dataPath, 'r') as f:
    i = 0
    for line in f.readlines():
        line = line.strip()
        strs = line.split(' ')
        if str.isdigit(strs[0]):
            locations.append([])
            for j in range(4):
                locations[i].append(int(strs[j]))
            i += 1
        else:
            filenames.append(strs[0])
            line_cnt.append(int(strs[1]))
```

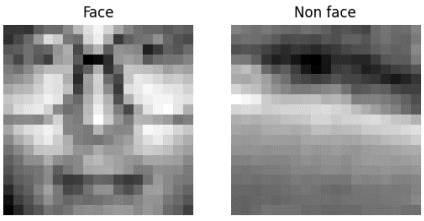
```python
file_cnt = 0
line_idx = 0
for line in line_cnt:
    path = os.path.join('data/detect', filenames[file_cnt])
    file_cnt += 1
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    img_ori = cv2.imread(path)
    for _ in range(line):
        x = locations[line_idx][0]
        y = locations[line_idx][1]
        w = locations[line_idx][2]
        h = locations[line_idx][3]
        crop_img = img[y:y+h, x:x+w]
        crop_img = cv2.resize(crop_img, (19, 19), interpolation=cv2.INTER_NEAREST)
        ans = clf.classify(crop_img)
        if ans:
            cv2.rectangle(img_ori, (x, y), (x+w, y+h), GREEN, 2)
        else:
            cv2.rectangle(img_ori, (x, y), (x+w, y+h), RED, 2)
        line_idx += 1
    cv2.imshow('result', img_ori)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# End your code (Part 4)
```

Part II. Results & Analysis (12%):

1. Results:

↓ Part 1



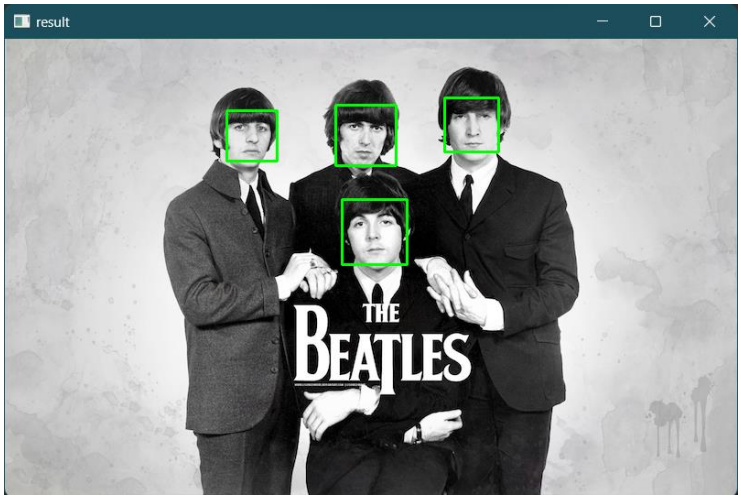
↓ Part 2

```
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
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 9, 2, 2), RectangleRe
gion(2, 11, 2, 2)], negative regions=[RectangleRegion(2, 9, 2, 2), RectangleRegion(4, 11, 2, 2)]) with accuracy: 137.000000 a
nd 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)
```

↓ Part 4


result
THE
BEATLES


result

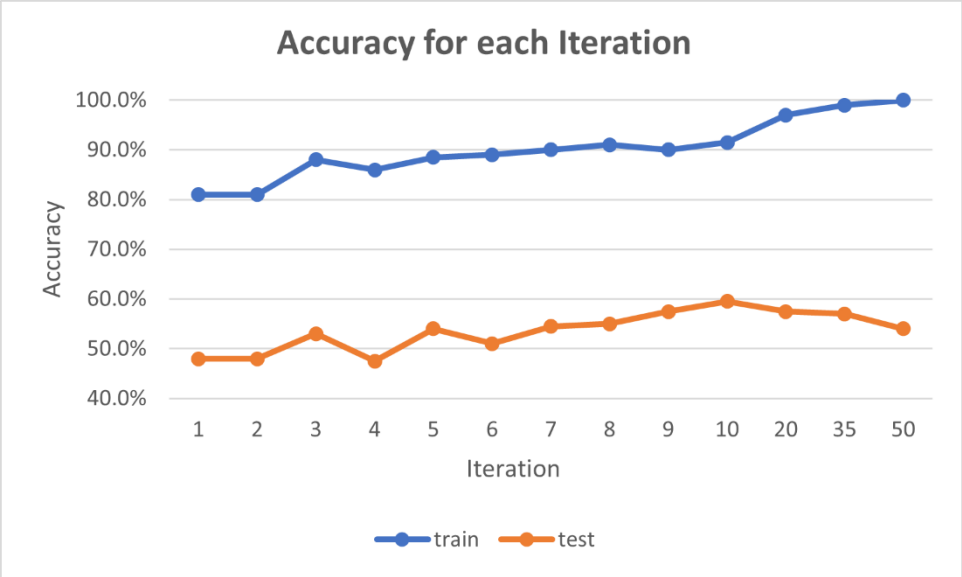
↓ Part 5


CTBC BANK
BASEBALL
CLASSIC
中華隊成軍記者會

2. Analysis:

↓ Part 3

200張	train data accuracy	test data accuracy
method 1 t=1	81.0%	48.0%
method 1 t=2	81.0%	48.0%
method 1 t=3	88.0%	53.0%
method 1 t=4	86.0%	47.5%
method 1 t=5	88.5%	54.0%
method 1 t=6	89.0%	51.0%
method 1 t=7	90.0%	54.5%
method 1 t=8	91.0%	55.0%
method 1 t=9	90.0%	57.5%
method 1 t=10	91.5%	59.5%
method 1 t=20	97.0%	57.5%
method 1 t=35	99.0%	57.0%
method 1 t=50	100.0%	54.0%



As t increases, train data accuracy gradually increases. When t=50, it even reaches 100%. This is because the model learned based on the train data, so if the train data is trained multiple times, train data accuracy obtained will be higher and higher. However, for test data accuracy, it initially increases as t increases, but after t=10, it starts to drop. This is due to overfitting. When the model is trained too many times on the same train data, it loses the ability to distinguish generalized datasets to fit outliers.

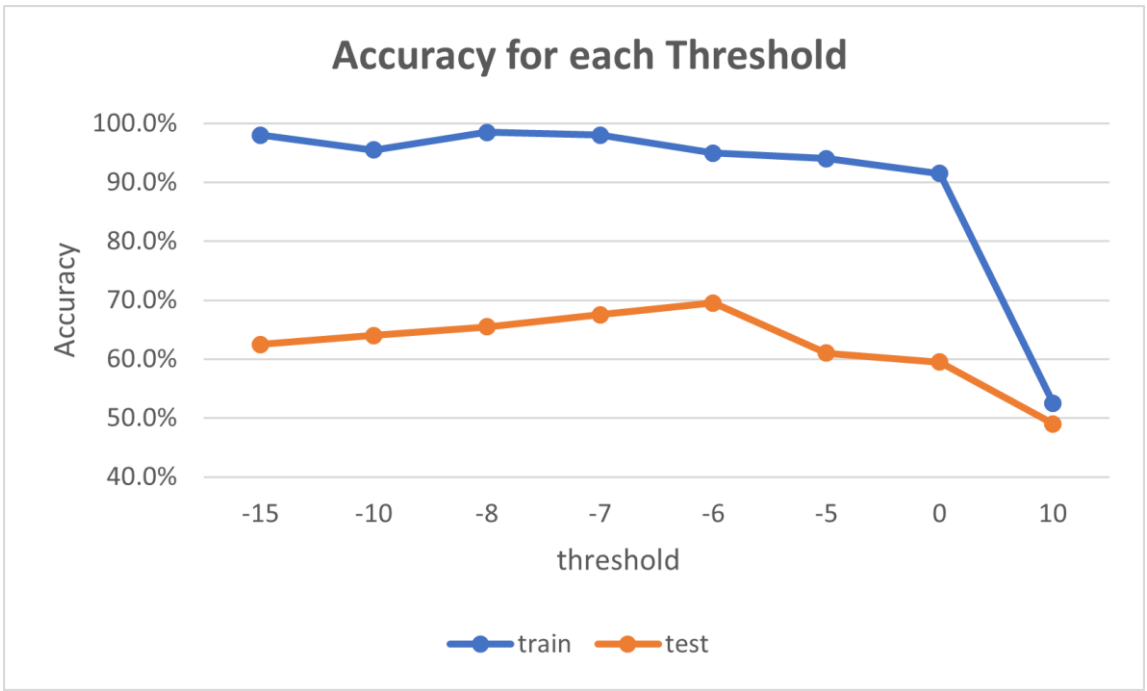
3. Bonus:

↓ Part 6

```
177             # Begin your code (Part 6)
178             bestError = float('inf')
179             bestIndex = -1
180             for i in range(len(features)):
181                 error = 0
182                 for j in range(len(iis)):
183                     isFace = (featureVals[i][j] < 0)
184                     error += weights[j] * (abs(isFace - labels[j]))
185                 if error < bestError:
186                     bestIndex = i
187                     bestError = error
188             bestClf = WeakClassifier(features[bestIndex], threshold=-6)
189             # End your code (Part 6)
190             return bestClf, bestError
```

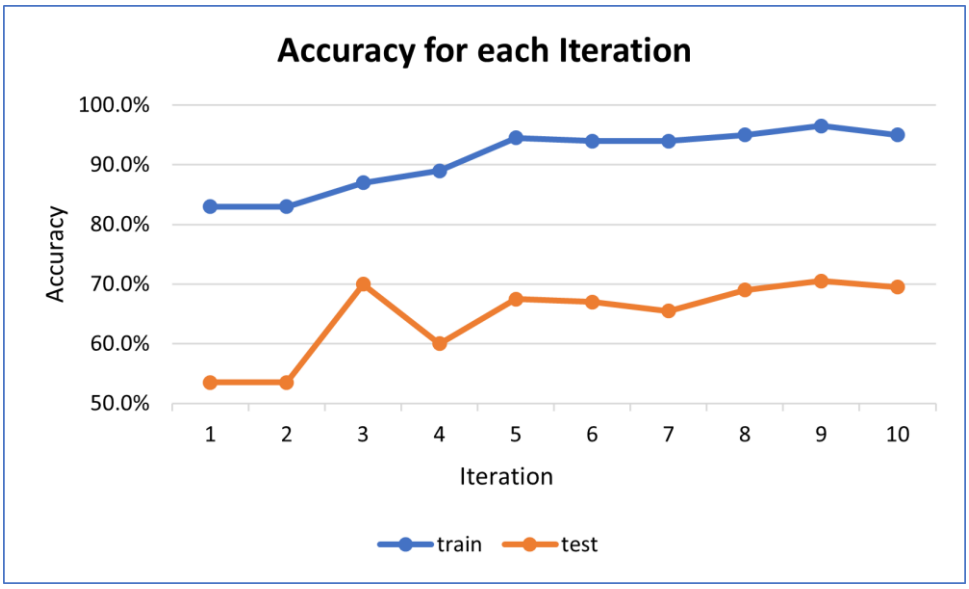
Unlike in part 2, I changed the value of the threshold. I tested values ranging from -15 to 10 and found that the test data accuracy was highest at around 70% when the threshold was set to -6.

threshold(T=10)	train data accuracy	test data accuracy
-15	98.0%	62.5%
-10	95.5%	64.0%
-8	98.5%	65.5%
-7	98.0%	67.5%
-6	95.0%	69.5%
-5	94.0%	61.0%
0	91.5%	59.5%
10	52.5%	49.0%



Therefore, I decided to test the accuracy of T values ranging from 1 to 10 using the threshold of -6.

200張	train data accuracy	test data accuracy
method 2 t=1	83.0%	53.5%
method 2 t=2	83.0%	53.5%
method 2 t=3	87.0%	70.0%
method 2 t=4	89.0%	60.0%
method 2 t=5	94.5%	67.5%
method 2 t=6	94.0%	67.0%
method 2 t=7	94.0%	65.5%
method 2 t=8	95.0%	69.0%
method 2 t=9	96.5%	70.5%
method 2 t=10	95.0%	69.5%



Compared to parts 2 and 3, not only did the train data accuracy increase in part 6, but the test data accuracy also reached around 70%, which is an increase of about 10%. This indicates that the approach taken in part 6 has better classification ability for generalized datasets.

Part III. Answer the questions (12%):

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

Ans: A problem I faced while completing this assignment was my unfamiliarity with image processing using Python. The assignment involved numerous image operations, including reading, cropping, resizing, among others. As a result, I spent an amount of time searching for relevant documentation on these operations, and I finally figured it out.

2. What are the limitations of the Viola-Jones' algorithm?

Ans: First, the algorithm relies on a fixed set of features to detect faces, so it is not sufficient to accurately detect all kinds of faces. Second, the algorithm is designed to work best on frontal faces, so it may not perform well when detecting side view of faces. Third, the algorithm can be computationally intensive, so it may take a long time processing the results.

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

Ans: Changing the value of threshold of weak classifier can improve the accuracy as I done in part 6. Also, increasing the scale of the image can help to improve the accuracy because larger images provide more detail.

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.

Ans: A possible algorithm to detect faces involves first identifying the eyes, and then using their relative positions to locate the ears, nose, and mouth. If these facial features are all detected in the correct positions, then the algorithm can conclude that a face is present in the image. One advantage of this approach is that it may be faster than the Adaboost algorithm. However, a limitation of this method is that it is more restricted to detecting faces that are directly facing the camera and the accuracy may be lower than the Adaboost algorithm.