HW1 : Report

Part I.

part1-1. screenshot

```python
# Begin your code (Part 1-1)
# Define paths for face and non-face images
train_face_path = 'data/data_small/train/face/*.pgm'
train_nonface_path = 'data/data_small/train/nonface/*.pgm'
test_face_path = 'data/data_small/test/face/*.pgm'
test_nonface_path = 'data/data_small/test/nonface/*.pgm'

# Load training dataset
train_data = []
for path, label in [(train_face_path, 1), (train_nonface_path, 0)]:
    for filename in glob.glob(path):
        img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
        train_data.append((img, label))

# Load testing dataset
test_data = []
for path, label in [(test_face_path, 1), (test_nonface_path, 0)]:
    for filename in glob.glob(path):
        img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
        test_data.append((img, label))

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

part1-2. screenshot

```python
    # Begin your code (Part 1-2)
    while True:
        intersects = False
        img_h, img_w = img_gray.shape

        # Randomly generate non-face bounding box
        # Ensure that the non-face bounding box does not intersect with any face bounding box

        # Generate random coordinates for non-face region
        x1 = np.random.randint(0, img_w - 19)  # Left coordinate of the bounding box
        y1 = np.random.randint(0, img_h - 19)  # Top coordinate of the bounding box
        x2 = x1 + 19  # Right coordinate of the bounding box
        y2 = y1 + 19  # Bottom coordinate of the bounding box

        # Check if the non-face bounding box intersects with any face bounding box
        for face_box in face_box_list:
            if (x1 < face_box[1][0] and x2 > face_box[0][0] and
                    y1 < face_box[1][1] and y2 > face_box[0][1]):
                intersects = True
                break

        if not intersects:
            img_crop = img_gray[y1:y2, x1:x2].copy()
            break

    # End your code (Part 1-2)
```

part2. screenshot

```
# Begin your code (Part 2)
# two slow
# train weak classifier for each feature
min_error = float('inf')
bestClf = None

for j in tqdm(range(len(features))):

    sorted_features = sorted(zip(featureVals[j], weights, labels))
    left_1_weight,  left_0_weight, right_1_weight, right_0_weight = 0, 0, 0, 0

    for threshold, weight, label in sorted_features:
        if label == 0:
            right_0_weight += weight
        else:
            right_1_weight += weight

    for threshold, weight, label in sorted_features:
        error = 0
        if label == 1:
            right_1_weight -= weight
            error += weight
        else:
            right_0_weight -= weight

         if left_1_weight + right_0_weight < left_0_weight + right_1_weight:
             polarity = -1
             error += left_1_weight + right_0_weight
         else:
             polarity = 1
             error += left_0_weight + right_1_weight

         if label == 1:
             left_1_weight += weight
         else:
             left_0_weight += weight

         if error < min_error:
             min_error = error
             bestClf = WeakClassifier(features[j], threshold, polarity)

bestError = min_error
# End your code (Part 2)
```

part4. screenshot

```python
# Begin your code (Part 4)
with open(dataPath, 'r') as file:
    line_list = [line.rstrip().split() for line in file]

line_idx = 0
while line_idx < len(line_list):
    img_gray = cv2.imread(os.path.join("data/detect", line_list[line_idx][0]),  cv2.IMREAD_GRAYSCALE)
    num_faces = int(line_list[line_idx][1])

    # Crop face region using the ground truth label
    box_list = []
    for i in range(num_faces):
        # get boxes
        x, y = int(line_list[line_idx + 1 + i][0]), int(line_list[line_idx + 1 + i][1])
        w, h = int(line_list[line_idx + 1 + i][2]), int(line_list[line_idx + 1 + i][3])
        img_crop = cv2.resize(img_gray[y:y + h, x:x + w].copy(), (19, 19))

        # classify
        if clf.classify(img_crop) == 1:
            box_list.append(((x, y), (x + w, y + h), 1))
        else:
            box_list.append(((x, y), (x + w, y + h), 0))

    image = cv2.imread(os.path.join("data/detect", line_list[line_idx][0]))
    for left_top, right_bottom, label in box_list:
        if label == 1:
            cv2.rectangle(image, left_top, right_bottom, (0, 255, 0), 2)
        else:
            cv2.rectangle(image, left_top, right_bottom, (255, 0, 0), 2)

    line_idx += num_faces + 1

    # Show the image with face detections
    cv2.imshow('Face Detection Result', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
# End your code (Part 4)
```

Part II.

1. FDDB_dataset screenshot

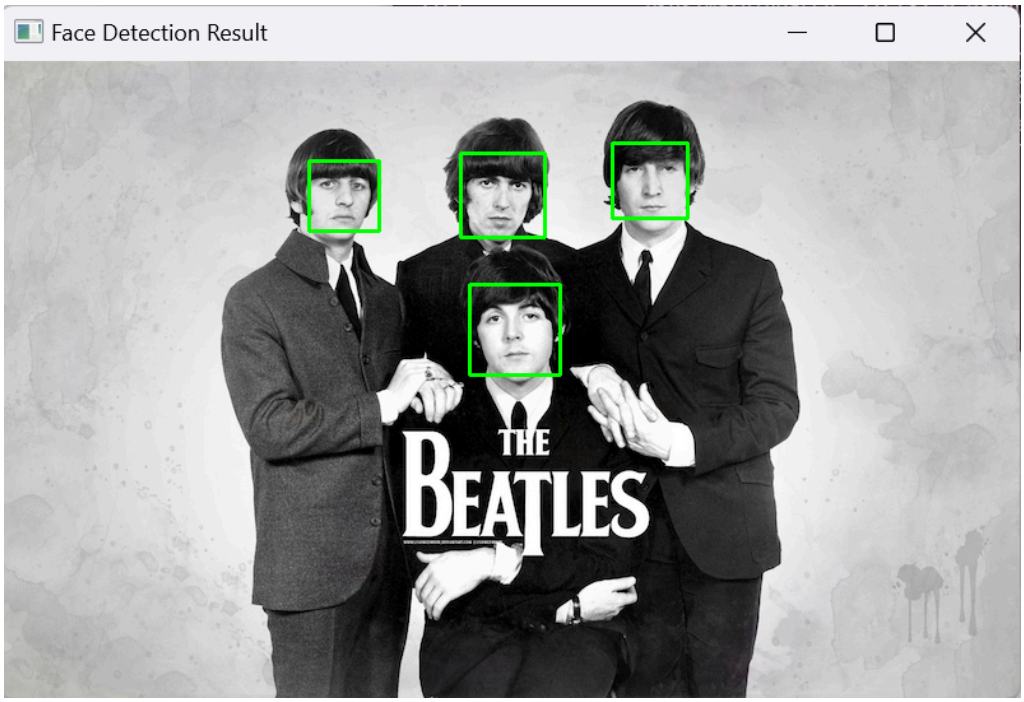
Case. T = 2

```
Run No. of Iteration: 2
100%|| 5171/5171 [00:05<00:00, 1014.23it/s]
Chose classifier: Weak Clf (threshold=20, polarity=-1, Haar feature (positive regions=[RectangleRegion(5, 16, 6, 1)], negative regions=[RectangleRegion(5, 17, 6, 1)]) with accuracy: 603.000000 and alpha: 1.414962

Evaluate your classifier with training dataset
False Positive Rate: 26/360 (0.072222)
False Negative Rate: 69/360 (0.191667)
Accuracy: 625/720 (0.868056)

Evaluate your classifier with test dataset
False Positive Rate: 13/155 (0.083871)
False Negative Rate: 34/155 (0.219355)
Accuracy: 263/310 (0.848387)
round 2is finished
```

Case. T = 8

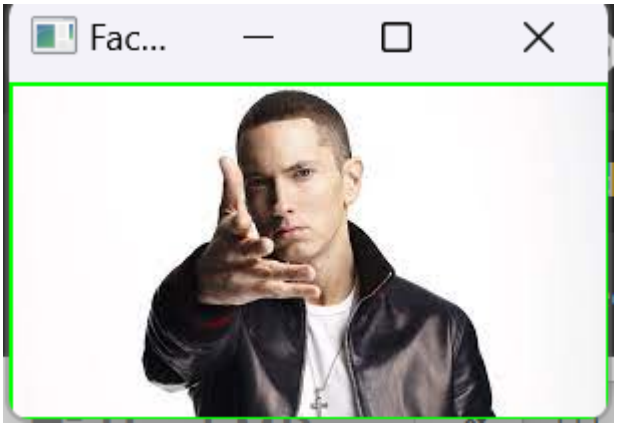
```
Run No. of Iteration: 8
100%|██████████| 5171/5171 [00:05<00:00, 999.04it/s]
Chose classifier: Weak Clf (threshold=-16, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 8, 3, 1)], negative regions=[RectangleRegion(5, 9, 3, 1)]) with accu
racy: 587.000000 and alpha: 1.117827

Evaluate your classifier with training dataset
False Positive Rate: 22/360 (0.061111)
False Negative Rate: 24/360 (0.066667)
Accuracy: 674/720 (0.936111)

Evaluate your classifier with test dataset
False Positive Rate: 11/155 (0.070968)
False Negative Rate: 22/155 (0.141935)
Accuracy: 277/310 (0.893548)
round 8is finished
```

Part4. beatles



Part4. presidents

myimage



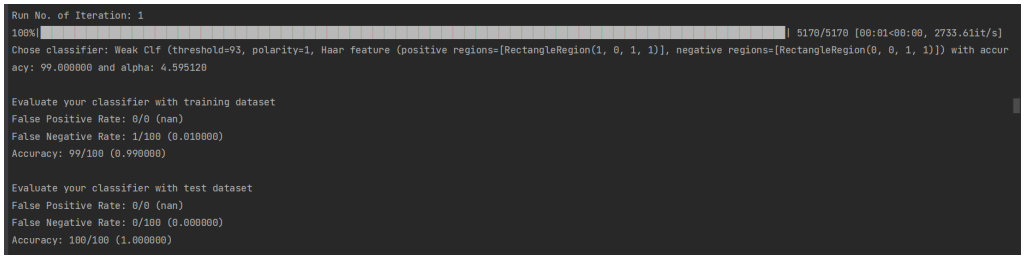
Analysis

times	train Accuracy	test Accuracy
T = 2	0.868	0.848
T = 3	0.868	0.848
T = 4	0.908	0.874
T = 5	0.915	0.870
T = 6	0.923	0.890
T = 7	0.926	0.890
T = 8	0.936	0.893
T = 9	0.943	0.887
T = 10	0.954	0.883

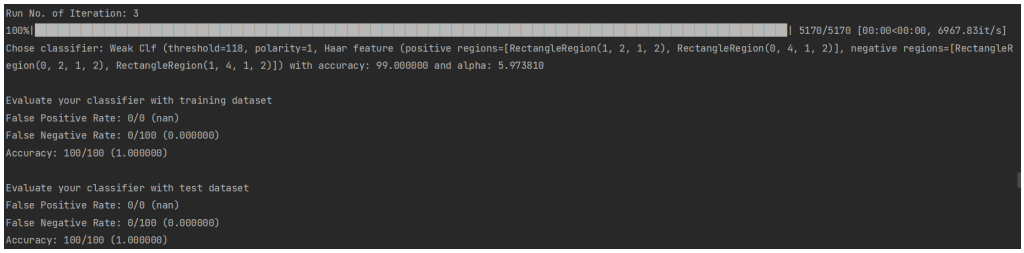
- **No overfitting**
  Observing that the train accuracy and test accuracy are closed,
  it means that there is no overfitting in these model
  It conf
- **Opimization issue**
  I also find that train accuracy haven't converge to 100% at T = 10.
  Maybe I could improve the selectBest() function to get a better result
  it could be still optimized better
- **High accuracy**
  I think that I have a high accuracy on my model. It is because I iterate every threshold to ensure I get the best weakclassifier

2. small dataset

Case. T = 2



Case. T = 3



times	train Accuracy	test Accuracy
T = 1	0.99	1.00
T = 2	0.99	1.00
T = 3	1.00	1.00
T = 4	1.00	1.00
T = 5	1.00	1.00
T = 6	1.00	1.00
T = 7	1.00	1.00
T = 8	1.00	1.00
T = 9	1.00	1.00
T = 10	1.00	1.00

- **perform well on small dataset compared to FDDB**
  I found that my model work well on small dataset, but it works relatively not well on FDDB dataset
  I believed that it is because datas in small dataset are not diverge, so we will have a better result on this dataset

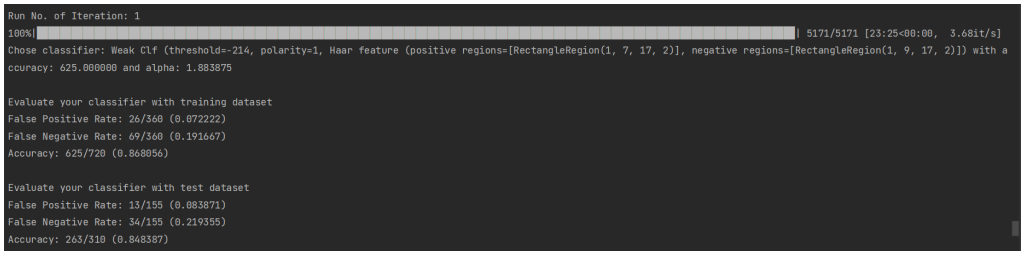
Bonus : Part 6

I implement it selectBest in the following form. I think if we have GPU, or some parallelized computing method. This method will swifter than my code now.
But now, the code works slowly due to its complexity O(n^2)

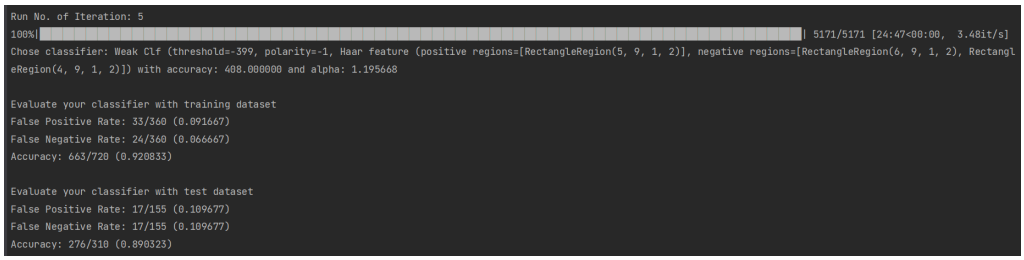
```
for j in tqdm(range(len(features))):
    for i in range(len(iis)):
        threshold = featureVals[j][i]
        for polarity in [-1, 1]:
            # Compute predictions based on the threshold and polarity
            predictions = np.where(polarity * (threshold - featureVals[j]
            # Calculate the error using weighted sum of incorrect predict
            error = np.dot(weights, predictions != labels)
            # Update the best classifier if the current error is lower
            if error < min_error:
                min_error = error
                bestClf = WeakClassifier(features[j], threshold, polarity
```

Anaysis

Case T = 1



Case T = 5



times	train Accuracy	test Accuracy
T = 1	0.868	0.848
T = 2	0.868	0.848
T = 3	0.893	0.890
T = 4	0.898	0.896
T = 5	0.920	0.890
T = 6	0.931	0.904

- **Waste too much time**
  In this selectBest() function, I find that we should take care of the time complexity of code. Selecting a weak classifier takes 15 to 20 mins in this code, while it is 9 sec in our previous code.
- **a little more accurate**
  However, this function performs better than our previous code. It shows that we may weight between time and accuracy when choosing model.

Part III.

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

The problem that I encounter is that my model training is too slow.
Actually, the original code of selectBest() function is O(n^2), whereas the new code is O(n*log(n)).
I optimize the code by sorting (the new code refers to PartI. part2)

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

I generate them by randomly selecting an area of pictures. It will try to select an area unitl there exist a area with no overlap with bounding box.

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

Viola-Jones assumes all the features of picture could be catagorized into 5 groups of Hear features. It relatively oversimplify the picture detection problem.
In reality, it couldn't memorize the pattern of object in its network. Nor could It do prediction by considering multiple pattern. (because it only predict by weighted sum of few weak classifiers)

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

I think consider all of the thresholds would be a good way to improve Viola-Jones. The more precise the threshold, the more accurate the model will be.

5. Other than Viola-Jones' algorithm, please propose another possible face detection method . Please discuss the pros and cons of the idea you proposed, compared to the Adaboost algorithm

We could use transformer. I thought transformer is good at doing sequence to sequence problem. Then, it might do well in face detection (we could stretch a picture to a vector)

- pros: transformer detect the similarity of adjacent pixels with a more precise way (with self-attention). It could have a better result than Viola-jones (which use Haar features)
- cons: training a transformer envolving matrix multiplication, which is required with GPU.
  However, training Viola-jones is faster. Moreover, predicting with Viola-jones is also faster than predicting with neural network .