Homework 1: Face Detection Report Template

Part I. Implementation (5%):

- ### Part 1-1

```python
# Begin your code (Part 1-1)
'''
Line 31 ~ 35:   Defining paths to directories containing image files.
Line 37 ~ 41:   Iterating over each pair of path and label in the list.
                For each path, using glob.glob(path) to retrieve a list of filenames that match the specified pattern.
                Then, for each filename, loading the image using cv2.imread(), converting it to grayscale,
                and appending a tuple (img, label) to the train_data list. The label indicates the class of the image (1 for face, 0 for non-face).
Line 43 ~ 47:   Similar to loading training data.
Line 49:        Combining the train_data and test_data lists into a single dataset list
'''
path = "data/data_small"
test_path_nonface = path + '/test/non-face/*.pgm'
test_path_face = path + '/test/face/*.pgm'
train_path_nonface = path + '/train/non-face/*.pgm'
train_path_face = path + '/train/face/*.pgm'

train_data = []
for path, label in [(train_path_face, 1), (train_path_nonface, 0)]:
    for filename in glob.glob(path):
        img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
        train_data.append((img, label))

test_data = []
for path, label in [(test_path_face, 1), (test_path_nonface, 0)]:
    for filename in glob.glob(path):
        img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
        test_data.append((img, label))

dataset = [train_data, test_data]
# End your code (Part 1-1)
```

- ### Part 1-2

```python
    for i in range(num_faces):
        # Begin your code (Part 1-2)
        '''
        For each face, we continue to generate random crops of the image until we find a region that doesn't overlap with any of the faces.
        This is implemented by generating a random top-left coordinate (`rand_top_left`) and a random bottom-right coordinate (`rand_bottom_right`),
        ensuring that they have greater x and y values.
        We then check if this randomly generated region overlaps with any of the previously detected face regions stored in `face_box_list`.
        If the region does not overlap and its area is greater than 0, indicating a valid non-face region,
        we crop it from the grayscale image `img_gray` and append it, along with the label `0`, to the `nonface_dataset`.
        '''

        # Initialize a flag to check if the random cropping overlaps any of the face regions
        overlap = True

        # Generate random non-face regions until no overlap is found
        while overlap:
            # Generate random top-left and bottom-right coordinates for cropping
            rand_top_left = (np.random.randint(0, img_gray.shape[1] - 19), np.random.randint(0, img_gray.shape[0] - 19))
            rand_bottom_right = (rand_top_left[0] + 19, rand_top_left[1] + 19)

            # Check for overlap with any of the face regions
            overlap = False
            for face_box in face_box_list:
                if (rand_bottom_right[0] >= face_box[0][0] and rand_top_left[0] <= face_box[1][0] and
                        rand_bottom_right[1] >= face_box[0][1] and rand_top_left[1] <= face_box[1][1]):
                    overlap = True
                    break

        # Crop the non-face region and add it to the non-face dataset
        img_crop = img_gray[rand_top_left[1]:rand_bottom_right[1], rand_top_left[0]:rand_bottom_right[0]].copy()

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

- ## Part 2

```python
        # Begin your code (Part 2)
        """
        1. Initialize bestClf and bestError to none and infinity, respectively.
        2. For every column of featureVals, initialize eps=0.
        3. Check every raw whether featureVals and labels are correct
           (featureVals<=0, labels=1 or featureVals>0, label=0).
           If not, add its corresponding weights to eps.
        4. After scanning the whole row, if eps is less than bestError,
           make bestError equal to eps and bestClf become weakclassifier of the corresponding features.
        5. Return the bestClf and bestError.
        """

        bestClf, bestError=None, float('inf')
        col, row=featureVals.shape
        for i in range(col):
            eps=0
            for j in range(row):
                if (featureVals[i][j]<0 and labels[j]==0) or (featureVals[i][j]>=0 and labels[j]==1):
                    eps+=weights[j]
            if eps<bestError:
                bestError=eps
                bestClf=WeakClassifier(features[i])
        # End your code (Part 2)
```

- ## Part 4

```python
# Begin your code (Part 4)
'''
This function loads the image paths and bounding box information from a text file specified by 'dataPath'.
It then iterates over each image, reads it, converts it to grayscale, and crops the regions specified by the bounding box.
For each cropped region, it resizes it to 19x19 and converts it to grayscale.
It then uses the provided classifier 'clf' to classify whether the region contains a face or not.
If the classification result is True, it draws a green rectangle around the face; otherwise, it draws a red rectangle.
The function displays the image with the drawn rectangles.
'''
folderPath='data/detect/'
with open(dataPath, 'r') as file:
    text=file.readlines()
    while len(text):
        name, num=text[0].split()
        text.pop(0)
        img=cv2.imread(folderPath+name)
        gray_img=cv2.imread(folderPath+name, cv2.IMREAD_GRAYSCALE)
        for i in range(int(num)):
            x0, y0, x1, y1=map(int, text[0].split())
            x1+=x0
            y1+=y0
            text.pop(0)
            if clf.classify(cv2.resize(gray_img[y0:y1, x0:x1], (19, 19))):
                cv2.rectangle(img, (x0, y0), (x1, y1), (0, 255, 0), thickness=3)
            else:
                cv2.rectangle(img, (x0, y0), (x1, y1), (0, 0, 255), thickness=3)
        cv2.imshow('image', img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
# End your code (Part 4)
```

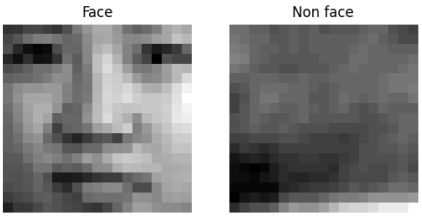
- **Part 6**

```
    '''
    Take t_pos and t_neg to accumulate the total weights of positive and negative samples.
    Iterates over each label and its corresponding weight, incrementing t_pos for positive labels and t_neg for negative labels.
    Then, iterating through each feature and its values, sorting the samples based on their values.
    For each feature, calculating the error considering the current feature, and updates the best error, best feature, best threshold,
    and best polarity if a smaller error is found. Also keeps track of the cumulative positive and negative weights and their counts.
    Finally, creating a WeakClassifier object with the best feature, threshold, and polarity,
    representing the best weak classifier based on the given features, feature values, labels, and weights.
    '''

    # Begin your code (Part 6)
    t_pos ,t_neg = 0, 0

    for label, w in zip(labels, weights):
        if label == 0:
            t_neg += w
        elif label == 1:
            t_pos += w

    best_feature, best_threshold, best_polarity, bestError = None, None, None, float('inf')
    for feature, vals in zip(features, featureVals):

        applied = sorted(zip( weights, vals, labels), key= lambda x: x[1])
        cur_pos_weight, cur_neg_weight, cur_pos_num, cur_neg_num = 0, 0, 0, 0

        for w, val, label in applied:

            error = min(cur_pos_weight + t_neg - cur_neg_weight,
                        cur_neg_weight + t_pos - cur_pos_weight)


            if error < bestError:
                bestError = error
                best_feature = feature
                best_threshold = val
                best_polarity = 1 if cur_pos_num >= cur_neg_num else -1
            if label == 1:
                cur_pos_num += 1
                cur_pos_weight += w
            else:
                cur_neg_num += 1
                cur_neg_weight += w

    bestClf = WeakClassifier(best_feature, best_threshold, best_polarity)

    # End your code (Part 6)
```

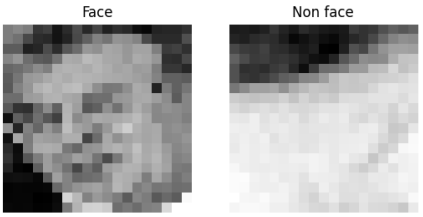
Part II. Results & Analysis (10%):

- **Part 1 loading data**

Part 1-1 **“load_data_small”**



Part 1-2 **“load_data_FDDB”**



- **Part 2**

The data after finishing selectBest () in adaboost.py -- Data small

```
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positiv
lpha: 0.707795
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positiv
ectangleRegion(4, 11, 2, 2)]) with accuracy: 0.685000 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)
```

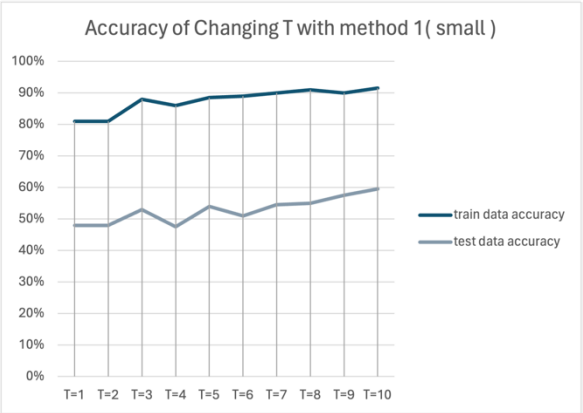
The data after finishing selectBest () in adaboost.py -- Data FDDB

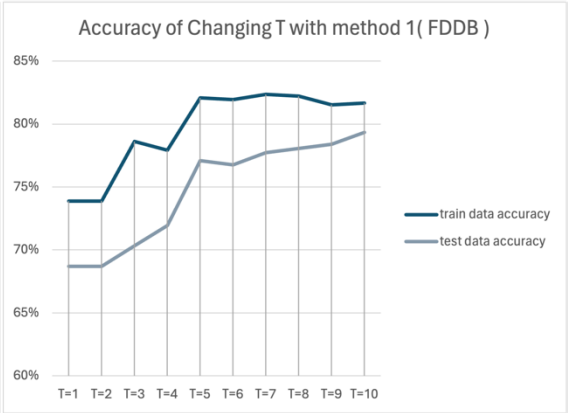
```
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive
 RectangleRegion(5, 15, 4, 2)]) with accuracy: 0.627778 and alpha: 0.383727
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive
 RectangleRegion(15, 6, 3, 2)]) with accuracy: 0.620833 and alpha: 0.304851

Evaluate your classifier with training dataset
False Positive Rate: 102/360 (0.283333)
False Negative Rate: 30/360 (0.083333)
Accuracy: 588/720 (0.816667)

Evaluate your classifier with test dataset
False Positive Rate: 47/155 (0.303226)
False Negative Rate: 17/155 (0.109677)
Accuracy: 246/310 (0.793548)
```

- **Part 3**

Different accuracy by testing parameter T from 1 to 10 including small and FDDB (face =1, non-face=0)

Small	train data accuracy	test data accuracy		FDDB	train data accuracy	test data accuracy
Method 1, T=1	81.00%	48.00%		Method 1, T=1	73.89%	68.71%
Method 1, T=2	81.00%	48.00%		Method 1, T=2	73.89%	68.71%
Method 1, T=3	88.00%	53.00%		Method 1, T=3	78.61%	70.32%
Method 1, T=4	86.00%	47.50%		Method 1, T=4	77.92%	71.94%
Method 1, T=5	88.50%	54.00%		Method 1, T=5	82.08%	77.10%
Method 1, T=6	89.00%	51.00%		Method 1, T=6	81.94%	76.77%
Method 1, T=7	90.00%	54.50%		Method 1, T=7	82.36%	77.74%
Method 1, T=8	91.00%	55.00%		Method 1, T=8	82.22%	78.06%
Method 1, T=9	90.00%	57.50%		Method 1, T=9	81.53%	78.39%
Method 1, T=10	91.50%	59.50%		Method 1, T=10	81.67%	79.35%





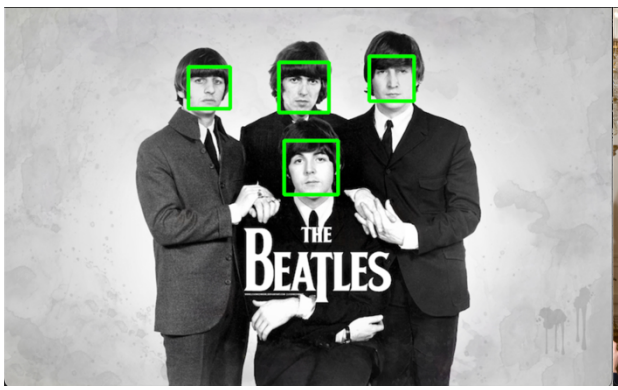
According to the results, we can reach the following conclusions:

1. For the test data in both dataset, the more we train, the accuracy will gradually grown up .
2. The accuracy of training data is always higher than the test data because the classifier is trained from training data.
3. Test data can tell whether the machine good or bad; hence, from the accuracy of test data we know the machine need to train more times or optimize its method for selecting classifier.
4. The increment in FDDB data is more obvious than in small data, which shows that if we use larger dataset to train the model, the improvement will be better.

- **Part 4**

The data after finishing detect () in detection.py – data small

(method 1, face =1, non-face=0, T=1)

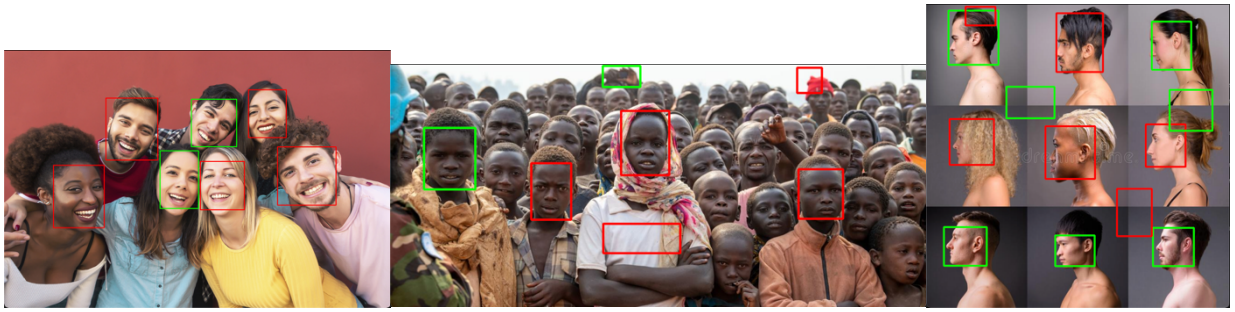


The data after finishing detect () in detection.py – data FDDB

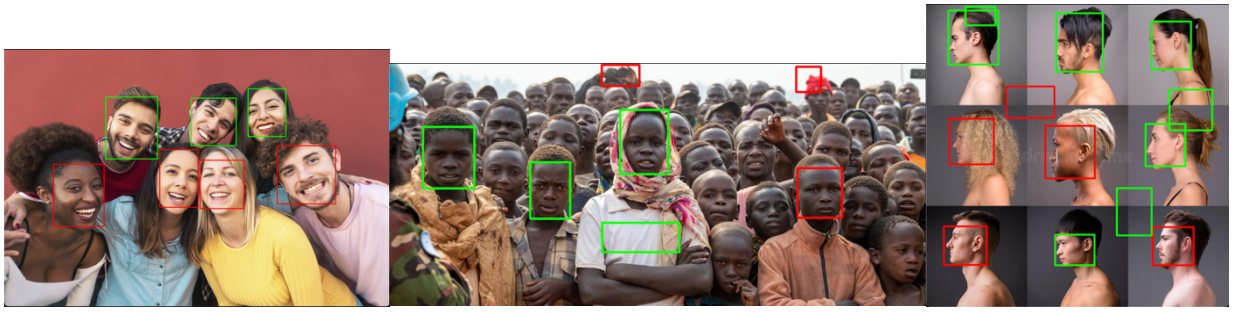
(method 1, face =1, non-face=0, T=10)

- **Part 5**

Data small (method 1, face =1, non-face=0, T=10)



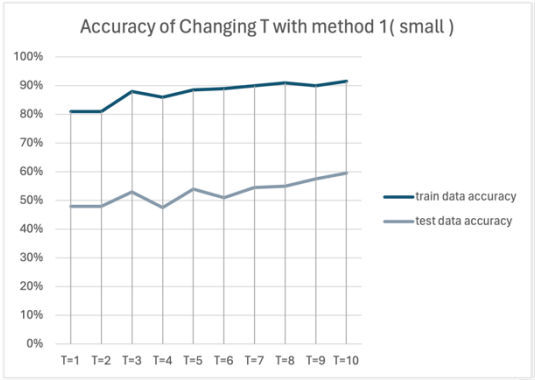
Data FDDB (method 1, face =1, non-face=0, T=10)

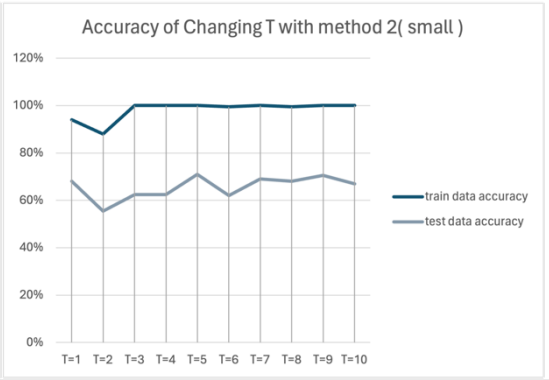


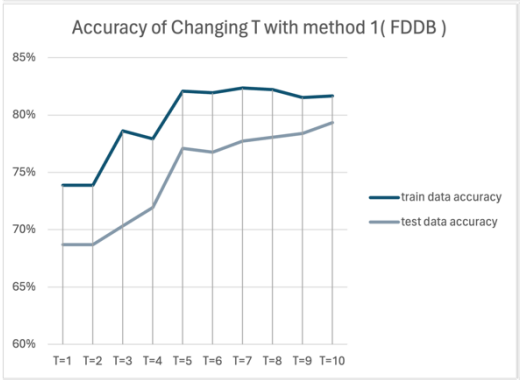
- **Part 6**

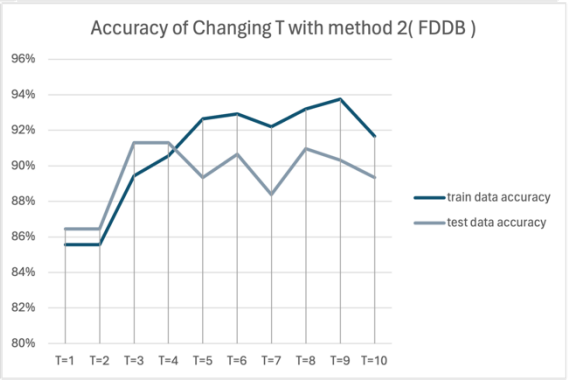
In the selectBest function, I calculate the best threshold and polarity of each classifier first, and then calculate their errors to find the smallest one.

Small	train data accuracy	test data accuracy		FDDB	train data accuracy	test data accuracy
Method 2, T=1	94.00%	68.00%		Method 2, T=1	85.56%	86.45%
Method 2, T=2	88.00%	55.50%		Method 2, T=2	85.56%	86.45%
Method 2, T=3	100.00%	62.50%		Method 2, T=3	89.44%	91.30%
Method 2, T=4	100.00%	62.50%		Method 2, T=4	90.56%	91.30%
Method 2, T=5	100.00%	71.00%		Method 2, T=5	92.64%	89.35%
Method 2, T=6	99.50%	62.00%		Method 2, T=6	92.92%	90.65%
Method 2, T=7	100.00%	69.00%		Method 2, T=7	92.20%	88.39%
Method 2, T=8	99.50%	68.00%		Method 2, T=8	93.19%	90.97%
Method 2, T=9	100.00%	70.50%		Method 2, T=9	93.75%	90.32%
Method 2, T=10	100.00%	67.00%		Method 2, T=10	91.67%	89.35%









According to the results, it indicates that we can get higher accuracy if we give each classifier its suitable threshold and polarity. We can tell that in method 2, both small and FDDB' s accuracy is higher than method 1. By giving the best thresholds for each classifier, we do get better performance. Still, the limit of the test data accuracy is at 71%. We cannot get higher result even increasing the parameter T. This might be occurring overfitting problem.

Part III. Answer the questions (15%):

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

   Initially, the complexity of the algorithm and inconsistent variable data types caused me to run into issues. The program was larger than I had anticipated, thus it was difficult for me to execute it with correctness and clarity. I took the time to carefully go over the code and record the inputs and outputs of every function to remedy this. To comprehend the complexities of Adaboost and Viola-Jones' algorithms and the code's overall operation better, I also did a lot of research on them. This thorough approach improved my understanding of the algorithm's workflow, assisted me in identifying and resolving variable data type-related problems, and ultimately resulted in a more reliable implementation.

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

   To generate "nonface" data by cropping images, I first identify regions in images that do not contain faces. This can be done by using existing face detection algorithms or manually annotating regions that contain faces. Once non-face regions are identified, I randomly select coordinates within these regions to define the top-left and bottom-right corners of cropping boxes. These coordinates are selected to ensure that the cropping box does not overlap with any face regions. Finally, I crop the images using the selected coordinates to generate non-face image samples.

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

   a. Sensitivity to lighting conditions: The Viola-Jones algorithm relies on analyzing the contrast between different regions of an image to detect objects. This means that it can be sensitive to changes in lighting conditions, which can affect the contrast and make it more difficult to detect objects accurately.

   b. Limited accuracy for non-frontal faces: As the professor discussed in the lecture, the algorithm was designed to detect frontal faces, and it may not be as accurate for non-frontal faces. This can be a limitation in applications where accurate detection of non-frontal faces is important.

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

    a. Use more complex features: The Viola-Jones algorithm uses Haar-like features to detect objects, but we can use more complex features to improve the accuracy of face detection.

    b. Use multiple detectors: Instead of relying on a single detector detect the whole face, multiple detectors can be trained to detect different parts of the object being detected. For example, in face detection, one detector can be trained to detect the eyes, another to detect the nose, and another to detect the mouth. Combining the output of these detectors can improve the accuracy of overall detection.

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.

    RetinaFace, a deep learning-based method for face recognition which was published in 2019.

    Comparison:

Algorithm / Comparison	Viola - Jones	RetinaFace
Accuracy	60%~80%	Higher than Viola-Jones
Accuracy under challenging condition	Low	High
Required dataset for training	less	more
Detect faces at different scales and orientations	Mostly impossible	Able