Homework 1: Face Detection

Part 1. Implementation(5%):

- Please screenshot your code snippets of Part 1, Part 2, Part 4, and explain your implementation.

```
    # Begin your code (Part 1-1)
    directory_path = "data/data_small/train/face"
    traindata = list(tuple())
    testdata = list(tuple())
    for file in os.listdir(directory_path):
        path = os.path.join(directory_path, file)
        image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        temp = ((image), 1)
        traindata.append(temp)

    directory_path = "data/data_small/train/non-face"
    for file in os.listdir(directory_path):
        path = os.path.join(directory_path, file)
        image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        temp = ((image), 0)
        traindata.append(temp)

    directory_path = "data/data_small/test/face"
    for file in os.listdir(directory_path):
        path = os.path.join(directory_path, file)
        image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        temp = ((image), 1)
        testdata.append(temp)

    directory_path = "data/data_small/test/non-face"
    for file in os.listdir(directory_path):
        path = os.path.join(directory_path, file)
        image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        temp = ((image), 0)
        testdata.append(temp)
    dataset = [traindata, testdata]

    # raise NotImplementedError("To be implemented") maybe done

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

Part 1-1

I do the same thing for train face, train non-face, test face, and test non-face respectively. I use os.path.join to specify the exactly position of all the images, and use cv2.imread to read them. The reason why I switch it in IMREAD_GRAYSCALE is that otherwise, there will be three components in image.shape, which doesn't fit the format in other code. Next, I assign a value of 1 to the face parts and a value of 0 to the non-face parts, following the order to organize the data into lists.

```
# Begin your code (Part 1-2)
img_height, img_weight = img_gray.shape
while True:
    overlap = False
    x, w = np.random.randint(img_weight, size=2)
    y, h = np.random.randint(img_height, size=2)
    for bound in face_box_list:
        if w==0 or h == 0:
            overlap = False
            break
        if x > bound[1][0] or (x+w) < bound[0][0] or y > bound[1][1] or (y+h) < bound[0][1]:
            overlap=True
        else:
            overlap=False
            break
    if overlap:
        break

left_top = (max(x, 0), max(y, 0))
right_bottom = (min(x+w, img_weight), min(y+h, img_height))
face_box_list.append([left_top, right_bottom]) #avoid two non-face overlap
# raise NotImplementedError("To be implemented")
img_crop = img_gray[left_top[1] : right_bottom[1], left_top[0]:right_bottom[0]].copy()
# End your code (Part 1-2)
```

Part 1-2

I use np.random.randint to generate four numbers: x, y, w and h. x and y represent the coordinates of the top-left point of the rectangle, and w and h are represent the width and height, respectively. I set the condition that if their values are not as

desired, I will regenerate them until the condition is met. The condition are as follow: first, w and h can't be 0 because them define a rectangle. Second, this rectangle can't intersect with any rectangles of faces generated before or any the non-face rectangles already generated. Once I obtain the desired rectangle, I copy the image bounded by the rectangle and append it to the non-face dataset.

```
# Begin your code (Part 2)

feature_count = 0
bestError = float('inf')
best_feature = None
for f, feature in zip(featureVals, features):
    cur_error = 0
    for i in range(len(f)):
        if f[i] < 0:
            h=1
        else:
            h=0
        cur_error += abs(h-labels[i])*weights[i]
    feature_count+=1
    if cur_error < bestError:
        best_feature = feature
        bestError = cur_error
bestClf = WeakClassifier(feature=best_feature)
# raise NotImplementedError("To be implemented")
# End your code (Part 2)
```

Part 2

Initially, I set the bestError to infinity, and create the best_feature. Depending on the feature f, I choose different h and put them into the formula to calculate the current error., similar to what's in the professor's presentation. Once done, I compare it with bestError. If the current error is better, I replace the bestError and the best_feature. Finally, I set the best feature in WeakClassifier to create the bestClf for return.

```
# Begin your code (Part 4)
file_part = dataPath.split('/')
part_path = '/'.join(file_part[:-1])
with open(dataPath) as file:
  line_list = [line.rstrip() for line in file]
line_idx = 0
while line_idx < len(line_list):
  target_name, times = line_list[line_idx].split()
  img = cv2.imread(os.path.join(part_path, target_name), cv2.IMREAD_GRAYSCALE)
  img_show = cv2.imread(os.path.join(part_path, target_name), cv2.COLOR_GRAY2BGR)

  for i in range(1, int(times)+1):
    coord = [int(float(val)) for val in line_list[line_idx+i].split()]
    img_crop = img[coord[1]:(coord[1] + coord[3]), coord[0]:(coord[0] + coord[2])].copy()
    fin = cv2.resize(img_crop, (19, 19))
    jud = clf.classify(fin)
    if jud:
      obj = {"face":[coord[0], coord[1], coord[0]+coord[2], coord[1]+coord[3]]}
      cv2.rectangle(img_show, (obj['face'][0], obj['face'][1]), (obj['face'][2], obj['face'][3]), (0, 255, 0), 2)
    else:
      obj = {"nonface":[coord[0], coord[1], coord[0]+coord[2], coord[1]+coord[3]]}
      cv2.rectangle(img_show, (obj["nonface"][0], obj["nonface"][1]), (obj["nonface"][2], obj['nonface'][3]), (0, 0, 255), 2)
  line_idx += (int(times) + 1)
  cv2.imshow("Window", img_show)
  cv2.waitKey(0)

# raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

Part 4

The way I obtain the image path is by splitting the data path and ignoring the last element("detect.txt"). Then, I join it with image's name from the detect.txt file. After

using cv2.imread to read the image, following the number after the image's name, I set a for loop to retrieve coordinates and size of each rectangle. With this data, I apply clf.classify() to determine whether it is a face. If it is the face, I use a green line to bound this rectangle; Otherwise, I use a red line. Additionally, I read the same image twice, Once in grayscale mode, and once in RGB color mode. This is because I need the grayscale image for clf.classify(), but I also need the original colored image to display the final result.

```
# Begin your code (Part 2)
    bestError = float('inf')
    best_feature = None

    # Convert features and labels to numpy arrays
    X = np.array(featureVals).T  # Features
    y = np.array(labels)         # Labels

    # Initialize Random Forest classifier
    rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

    # Train Random Forest classifier
    rf_classifier.fit(X, y)

    # Find feature importance scores
    feature_importances = rf_classifier.feature_importances_

    # Find index of feature with highest importance score
    best_feature_index = np.argmax(feature_importances)

    # Select best feature
    best_feature = features[best_feature_index]

    # Calculate error based on feature importance score
    bestError = 1 - feature_importances[best_feature_index]

    bestClf = WeakClassifier(feature=best_feature)
    # raise NotImplementedError("To be implemented")
    # End your code (Part 2)
```

Part 6(bonus)

In this part, I utilize the random forest algorithm. Initially, I convert the lists of feature values and labels into np array. Then, I employ the Random Forest Classifier imported from sklearn.ensemble. I specify the number of estimators in the forest as 100. Following this, I use the fit function to train the model, evaluate the importance of features, and identify the feature with the highest score as the best feature. Subsequently, I calculate the best error and generate the bestClf.

The accuracy achieved by this method is lower than that of the original approach, both with data small and data FDDB. Furthermore, the detection results are inferior compared to the original method.

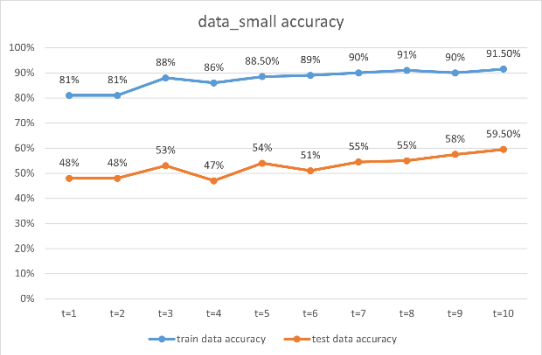
Part II. Results & Analysis (10%):

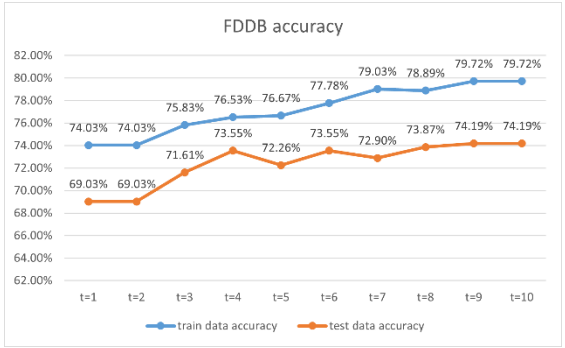
- Please screenshot the results.

```
Run No. of Iteration: 6
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 0, 12, 2)], negative regions=[RectangleRegion(5, 2,
12, 2)]) with accuracy: 0.656944 and alpha: 0.408637
Run No. of Iteration: 7
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 8, 1, 1)], negative regions=[RectangleRegion(10, 9,
 1, 1)]) with accuracy: 0.716667 and alpha: 0.404260
Run No. of Iteration: 8
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 16, 1, 1)], negative regions=[RectangleRegion(10, 1
7, 1, 1)]) with accuracy: 0.411111 and alpha: 0.335270
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(16, 9, 2, 2)], negative regions=[RectangleRegion(14, 9,
 2, 2)]) with accuracy: 0.681944 and alpha: 0.326487
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(11, 2, 7, 6), RectangleRegion(4, 8, 7, 6)], negative re
gions=[RectangleRegion(4, 2, 7, 6), RectangleRegion(11, 8, 7, 6)]) with accuracy: 0.644444 and alpha: 0.340554

Evaluate your classifier with training dataset
False Positive Rate: 111/360 (0.308333)
False Negative Rate: 35/360 (0.097222)
Accuracy: 574/720 (0.797222)

Evaluate your classifier with test dataset
False Positive Rate: 66/155 (0.425806)
False Negative Rate: 14/155 (0.090323)
Accuracy: 230/310 (0.741935)
```

These are generated under T=10 and with data_FDDB.

- Your analysis or observation. Please discuss the performance difference between the training and testing dataset, and present the results using a table or chart as follows. Moreover, compare the results of the two classifiers trained by different datasets.





Left side show the comparison between train data and test data in data_small, while the right side compares train data and test data in data_FDDB. According to these graphs, the accuracy of the train data consistently surpasses that of the test data. Moreover, although there's some fluctuation, the trend of accuracy becomes higher and higher when t is bigger.

Between FDDB and small, the accuracy of train data in data_small is higher than in FDDB, but the accuracy of test data exhibits the opposite trend. From my observations, I think that it's easier to detect the face in the picture with data_FDDB than data_small.

Part III. Answer the questions (15%):

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

The first problem I encountered was my lack of familiarity with Python, so it takes lots of time for me to understand what I'm doing. Next problem is that what I do in part 1-2, I think it can fit the command well, but it is not efficient. Unfortunately, I can't find the better way to replace it, so I keep this method. Additionally, I found that the mask may confuse the adaboost. I crop the mask in my own picture, and most of results showed in green line. I didn't try to fix it up because I thought that it was caused by inadequately data training.

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

I generate four number randomly, x, y, w, h, respectively. X and y for the left-top coordinate, w for the width and h for the height. After that, I set the condition for it. If this rectangle doesn't overlap with any rectangle of face and rectangle of non-face already generated, it can be used. Otherwise, generate other four number again. Moreover, I put the new rectangle into the face_box_list to prevent other rectangle overlapping with these rectangles.

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

First, training Viola-Jones' detector needs lots of calculation and intensive process to select and train a large number of weak classifiers. For the larger dataset, it will be time-consuming.

Second, because Viola-Jones' algorithm depends on Haar-like feature to distinguish. Therefore, if there is something, like hats, masks, etc, covering the part of face, it may not generate the correct result. Additionally, the complex backgrounds may cause the same problem.

Finally, the face size and rotation may lead to false detection under extreme variation.

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

We can select different feature type or even combine multiple type of feature to improve the accuracy, like LBP, HOG, etc.

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

I think we can input lots of data of facial features to machine, letting it to recognize how eyes, ears, mouth and nose look like. After that, it can detect the face

by finding the possible image's crop which looks like facial features. If what it detect are layout like the human face, it will assert it is the face.

I think the pros is that it may have higher accuracy comparing with Adaboost algorithm. Because it leverages detailed information about facial components to make more informed decisions. The cons is the sensitivity to angle. It may struggle if those features are obscured or appear differently due to changes in the angle of the face.