# Homework 1: Face Detection

Name : 林伯偉 Student ID : 109612019

## Part I. Implementation (5%):

***Notice: Make sure you put both data_FDDB & data_small into the data folder to run the code properly.***

- **Part 1**

```
# Begin your code (Part 1-1)
"""
    This code segment loads image data for training and testing. It reads
    images from specified directories for faces and non-faces, assigning
    labels accordingly (1 for faces, 0 for non-faces).The images are
    stored as tuples with their corresponding labels.
"""
train_data = []  # List to store training data tuples (image, label)
test_data = []   # List to store testing data tuples (image, label)

# Path to training data
train_path = "data/data_small/train/"

# Load face images for training
face_train_path = os.path.join(train_path, "face/")
for image in os.listdir(face_train_path):
    img = cv2.imread(os.path.join(face_train_path, image), cv2.IMREAD_GRAYSCALE)
    train_data.append((img, 1))  # Add face image with label 1 to training data

# Load non-face images for training
non_face_train_path = os.path.join(train_path, "non-face/")
for image in os.listdir(non_face_train_path):
    img = cv2.imread(os.path.join(non_face_train_path, image), cv2.IMREAD_GRAYSCALE)
    train_data.append((img, 0))  # Add non-face image with label 0 to training data

# Path to testing data
test_path = "data/data_small/test/"

# Load face images for testing
face_test_path = os.path.join(test_path, "face/")
for image in os.listdir(face_test_path):
    img = cv2.imread(os.path.join(face_test_path, image), cv2.IMREAD_GRAYSCALE)
    test_data.append((img, 1))  # Add face image with label 1 to testing data

# Load non-face images for testing
non_face_test_path = os.path.join(test_path, "non-face/")
for image in os.listdir(non_face_test_path):
    img = cv2.imread(os.path.join(non_face_test_path, image), cv2.IMREAD_GRAYSCALE)
    test_data.append((img, 0))  # Add non-face image with label 0 to testing data
# End your code (Part 1-1)
```

```
# Begin your code (Part 1-2)
"""
    This segment generates non-face image samples by randomly selecting regions from the
    original image that do not overlap with the detected face regions. It ensures that the
    selected regions have a maximum allowed intersection with the face region, defined by
    the variable max_intersection. The selected non-face regions are resized to 19x19 pixels
    and appended to the nonface_dataset list along with the label 0 (indicating non-face).
"""
img_height, img_width = img_gray.shape
face_left_top, face_right_bottom = face_box_list[i]

# Define the maximum allowed intersection with the face region
max_intersection = 0.2

# Randomly choose non-face regions until a suitable one is found
while True:
    x1 = np.random.randint(0, img_width - 19)
    y1 = np.random.randint(0, img_height - 19)
    x2 = x1 + 19
    y2 = y1 + 19

    # Calculate intersection area with the face region
    intersection_area = max(0, min(face_right_bottom[0], x2) - max(face_left_top[0], x1)) * \
                        max(0, min(face_right_bottom[1], y2) - max(face_left_top[1], y1))

    # Check if the intersection area is small enough
    if intersection_area / (19 * 19) <= max_intersection:
        nonface_dataset.append((cv2.resize(img_gray[y1:y2, x1:x2], (19, 19)), 0))
        break
# End your code (Part 1-2)
```

- **Part 2**

```
# Begin your code (Part 2)
"""
    This segment is responsible for finding the best weak classifier among
    a set of features for the given weights. It iterates over each feature and calculates
    the error associated with each one based on the feature values, labels, and weights.
    The error is computed as the weighted sum of absolute differences between the predicted
    and actual labels. The feature that results in the lowest error is selected as the best
    weak classifier. Finally, it returns the best weak classifier and its corresponding error.
"""
features_num = featureVals.shape[0]
dataset_num = featureVals.shape[1]
errors = np.zeros(features_num)

for j in range(features_num):
    for i in range(dataset_num):
        errors[j] += weights[i] * abs((1 if featureVals[j][i] < 0 else 0) - labels[i])

bestError = errors[0]
bestClf = WeakClassifier(features[0])

for i in range(1, features_num):
    if errors[i] < bestError:
        bestClf = WeakClassifier(features[i])
        bestError = errors[i]
# End your code (Part 2)
```

- **Part 4**

```
# Begin your code (Part 4)
"""
  This code reads data from a file containing image names and coordinates of detected faces.
  It then loads each image, detects faces, and draws rectangles around them based on the detection
  results. The processed images are saved with rectangles drawn around the detected faces.
"""
with open(dataPath, "r") as file:
    # Iterate through each line in the file
    for line in file:
        # Split the line into image name and number of faces
        image_name, people_num = line.split()
        people = []
        # Iterate through each face in the image
        for _ in range(int(people_num)):
            # Extract face data (x, y, width, height)
            face_data = next(file).split()
            face = tuple(map(int, face_data))
            people.append(face)
        # Load the image
        image = cv2.imread(os.path.join("data/detect/", image_name))
        # Load the grayscale version of the image
        image_cmp = cv2.imread(os.path.join("data/detect/", image_name), cv2.IMREAD_GRAYSCALE)
        # Iterate through each face in the image
        for face in people:
            x, y, w, h = face
            # Crop and resize the face region
            face_image = cv2.resize(image_cmp[y:y+h, x:x+w], (19, 19), interpolation=cv2.INTER_LINEAR)
            # Classify the face image
            classification = clf.classify(face_image)
            # Determine the color of the rectangle based on classification result
            color = (0, 255, 0) if classification == 1 else (0, 0, 255)
            # Draw rectangle around the detected face
            cv2.rectangle(image, (x, y), (x + w, y + h), color, thickness=2)
        # Save the resulting image with rectangles drawn around the detected faces
        cv2.imwrite("result_" + image_name, image)
# End your code (Part 4)
```

- **Part 6 (bonus)**

```
# Begin your code (Part 6)
"""
    This implementation calculates the errors for all features using list comprehensions
    and finds the minimum error using the min function. Then, it retrieves the corresponding
    best classifier based on the index of the minimum error.
"""
def my_selectBest(self, featureVals, iis, labels, features, weights):

    errors = []

    # Iterate over each row of featureVals, weights, and labels to compute errors
    for feature_row, weight, label in zip(featureVals, weights, labels):
        error = sum(weight * abs((1 if value < 0 else 0) - label) for value in feature_row)
        errors.append(error)

    # Find the index of the minimum error
    min_error_index = errors.index(min(errors))
    bestError = errors[min_error_index]
    bestClf = WeakClassifier(features[min_error_index])

    return bestClf, bestError
# End your code (Part 6)
```

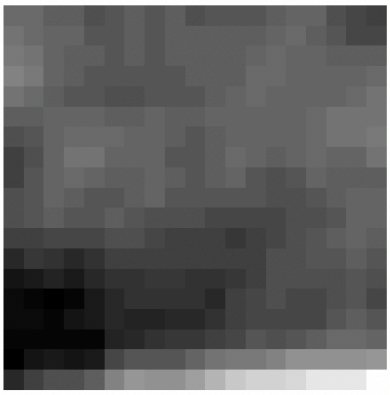
# Part II. Results & Analysis (10%):

- **Part 1**

  **These photos were generated after completing Part 1:**

Face

Non face



- **Part 2**

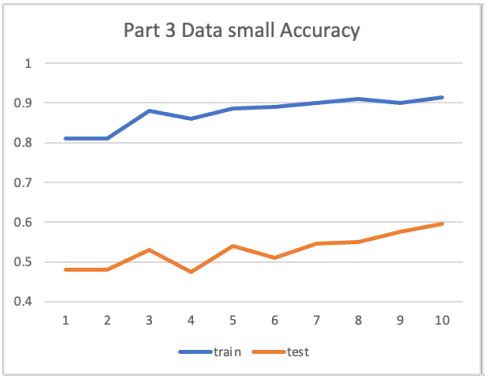
  **These data were generated after completing Part 2:**

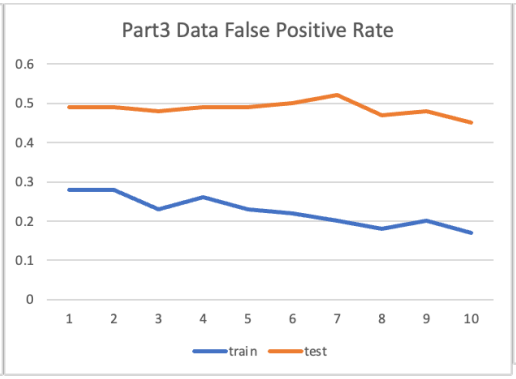
```
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 4, 1, 1)]
0.760000 and alpha: 0.707795
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 9, 2, 2),
2, 9, 2, 2), RectangleRegion(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)

Detect faces at the assigned location using your classifier

Detect faces on your own images
(test) (base) albertlin@Alberts-MacBook-Pro test %
```

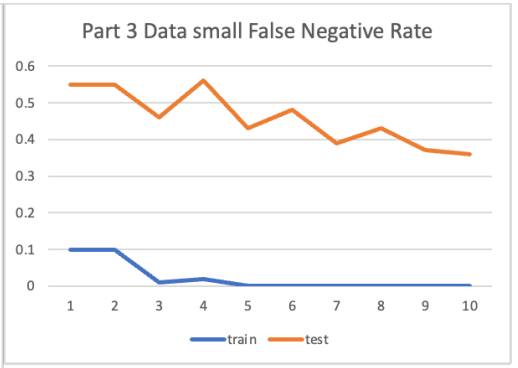
- **Part 3**

  **Parameter T tested between 1 ~ 10 generated the following data:**

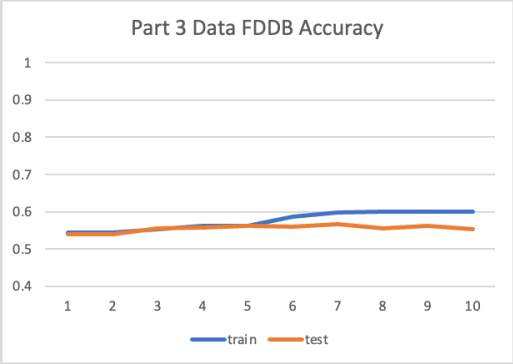
  **Data_small:**

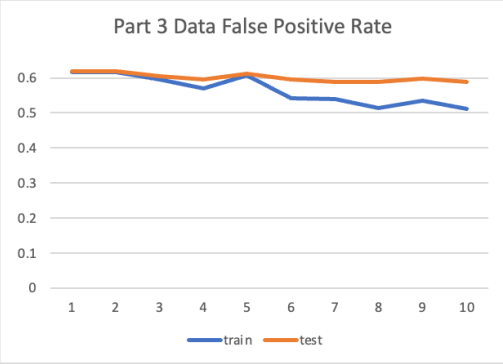


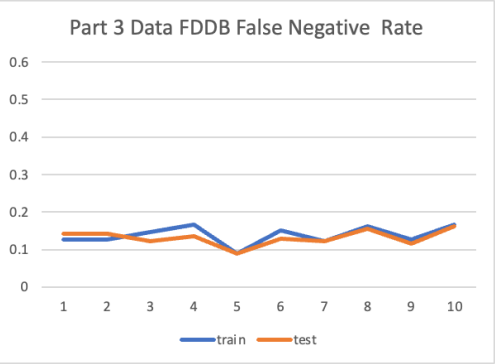



## Data_FDDB:




| small | | training | | | | test | | |
|---|---|---|---|---|---|---|---|---|
| | T | False Positiv | False Negati | Accuracy | | False Positiv | False Negati | Accuracy |
| | 1 | 0.28 | 0.1 | 0.81 | | 0.49 | 0.55 | 0.48 |
| | 2 | 0.28 | 0.1 | 0.81 | | 0.49 | 0.55 | 0.48 |
| | 3 | 0.23 | 0.01 | 0.88 | | 0.48 | 0.46 | 0.53 |
| | 4 | 0.26 | 0.02 | 0.86 | | 0.49 | 0.56 | 0.475 |
| | 5 | 0.23 | 0 | 0.885 | | 0.49 | 0.43 | 0.54 |
| | 6 | 0.22 | 0 | 0.89 | | 0.5 | 0.48 | 0.51 |
| | 7 | 0.2 | 0 | 0.9 | | 0.52 | 0.39 | 0.545 |
| | 8 | 0.18 | 0 | 0.91 | | 0.47 | 0.43 | 0.55 |
| | 9 | 0.2 | 0 | 0.9 | | 0.48 | 0.37 | 0.575 |
| | 10 | 0.17 | 0 | 0.914 | | 0.45 | 0.36 | 0.595 |
| | | | | | | | | |
| | | | | | | | | |
| FDDB | | training | | | | test | | |
| | T | False Positiv | False Negati | Accuracy | | False Positiv | False Negati | Accuracy |
| | 1 | 0.617 | 0.1278 | 0.545 | | 0.618 | 0.142 | 0.54 |
| | 2 | 0.617 | 0.127 | 0.545 | | 0.618 | 0.142 | 0.54 |
| | 3 | 0.595 | 0.147 | 0.554 | | 0.605 | 0.122 | 0.556 |
| | 4 | 0.571 | 0.167 | 0.563 | | 0.595 | 0.135 | 0.558 |
| | 5 | 0.607 | 0.09 | 0.563 | | 0.611 | 0.09 | 0.563 |
| | 6 | 0.543 | 0.152 | 0.586 | | 0.595 | 0.129 | 0.56 |
| | 7 | 0.54 | 0.122 | 0.598 | | 0.588 | 0.122 | 0.566 |
| | 8 | 0.513 | 0.161 | 0.6 | | 0.588 | 0.155 | 0.556 |
| | 9 | 0.5355 | 0.127 | 0.6 | | 0.598 | 0.116 | 0.562 |
| | 10 | 0.511 | 0.166 | 0.6 | | 0.588 | 0.161 | 0.554 |

**Comparison:**

- **Data_small:**

  The accuracy of the machine learning model improves with more trains. Though it shows higher accuracy in training data, it needs further training iterations to recognize faces beyond the training set. The model has a high true detection rate but also a high false positive rate, suggesting that the classification criteria may be too lenient. However, it accurately identifies actual faces, demonstrating a robust understanding of facial features.

- **Data_FDDB:**

  This classifier struggled to improve accuracy during testing, even with faster learning during training. Additionally, This classifier had difficulty reducing the false positive rate when compared to the Data_small dataset.

- **Part 4**

  These photos were generated after completing part 4:

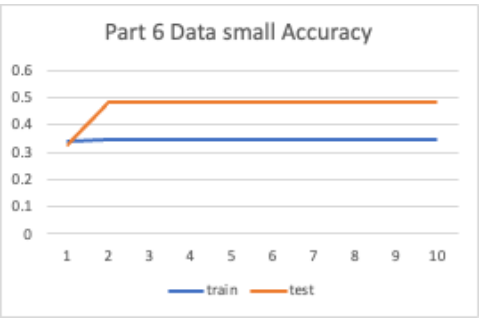

- **Part 5**

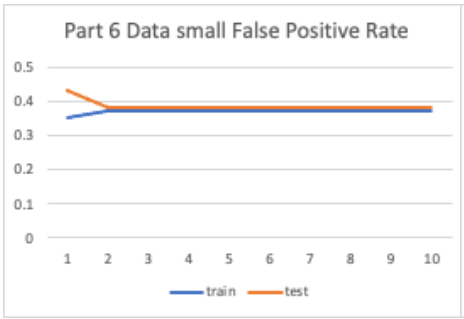
  This photo was generated after part 5 was done:

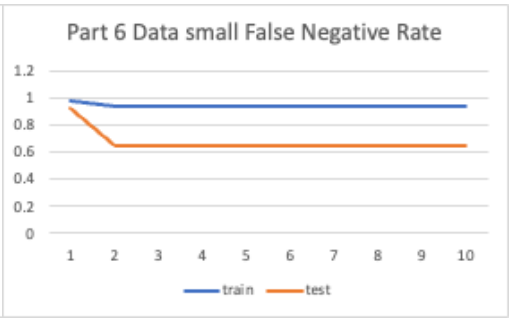
- **Part 6**

**Parameter T tested between 1 ~ 10 generated the following data:**

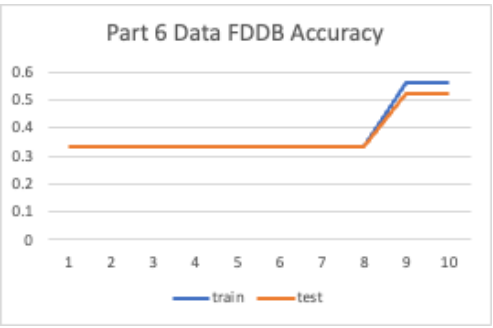
**Data_small:**

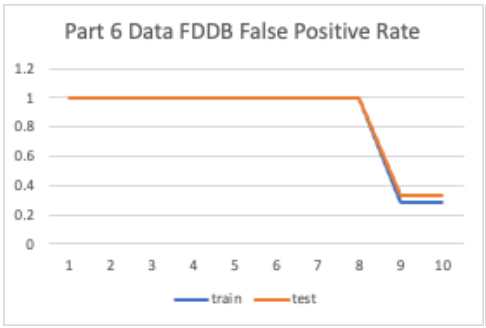


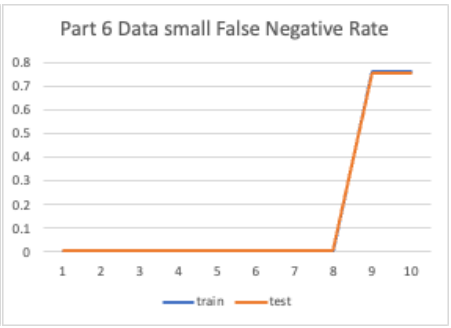




**Data_FDDB:**




| small | | training | | | | test | | |
|---|---|---|---|---|---|---|---|---|
| | T | False Positiv | False Negati | Accuracy | | False Positiv | False Negati | Accuracy |
| | 1 | 0.35 | 0.97 | 0.34 | | 0.43 | 0.92 | 0.325 |
| | 2 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | 3 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | 4 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | 5 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | 6 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | 7 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | 8 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | 9 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | 10 | 0.37 | 0.94 | 0.345 | | 0.38 | 0.65 | 0.485 |
| | | | | | | | | |
| FDDB | | training | | | | test | | |
| | T | False Positiv | False Negati | Accuracy | | False Positiv | False Negati | Accuracy |
| | 1 | 0.997 | 0 | 0.335 | | 0.997 | 0.003 | 0.336 |
| | 2 | 0.997 | 0 | 0.335 | | 0.997 | 0.003 | 0.336 |
| | 3 | 0.997 | 0 | 0.335 | | 0.997 | 0.003 | 0.336 |
| | 4 | 0.997 | 0 | 0.335 | | 0.997 | 0.003 | 0.336 |
| | 5 | 0.997 | 0 | 0.335 | | 0.997 | 0.003 | 0.336 |
| | 6 | 0.997 | 0 | 0.335 | | 0.997 | 0.003 | 0.336 |
| | 7 | 0.997 | 0 | 0.335 | | 0.997 | 0.003 | 0.336 |
| | 8 | 0.997 | 0 | 0.335 | | 0.997 | 0.003 | 0.336 |
| | 9 | 0.28 | 0.758 | 0.56 | | 0.336 | 0.754 | 0.523 |
| | 10 | 0.28 | 0.758 | 0.56 | | 0.336 | 0.754 | 0.523 |

**Comparison:**

This implementation calculates the errors for all features using list comprehensions and finds the minimum error using the min function. Then, it retrieves the corresponding best classifier based on the index of the minimum error. However, the result showed that my classifier was struggling to achieve accuracy as well as the original classifier, and it is not learning when there is little iteration, especially shown in the FDDB data set while showing a tendency to consider face images as non-face images.

## Part III. Answer the questions (15%):

1. Please describe a problem you encountered and how you solved it.
   - I can't read the data initially because I put the data_FDDB folder in the wrong folder structure. I spent a lot of time finding out why the images were not loading in and I solved this by looking up our specs closely.

2. How do you generate "nonface" data by cropping images?
   - We randomly select regions from an image that do not overlap with detected face regions. It checks the selected regions for maximum allowed intersection with face regions, crops and resizes them to 19x19 pixels, and labels them as non-face images. This process is repeated for each detected face to generate a balanced dataset of non-face images.

3. What are the limitations of the **Viola-Jones' algorithm**?
   - Sensitivity to variations in lighting conditions: Viola-Jones' algorithm relies heavily on the Haar-like features which can be affected by changes in lighting.
   - Difficulty in handling occlusions: When parts of the face are obscured or partially hidden, the algorithm may struggle to detect the entire face accurately.
   - Limited in detecting faces at different scales: Although the algorithm uses a cascading approach to detect faces at multiple scales, it may still miss faces that are significantly smaller or larger than the training data.
   - Susceptibility to false positives: Viola-Jones can produce false positives, especially in complex backgrounds or scenes with objects that resemble faces.

4. Based on **Viola-Jones' algorithm**, how to improve the accuracy except changing the training dataset and parameter T?
   - Cascade optimization: Fine-tune the cascade parameters to achieve a better balance between detection accuracy and speed.
   - Post-processing techniques: Apply techniques like non-maximum suppression to reduce false positives and improve overall detection accuracy.
   - Integration with other algorithms: Combine Viola-Jones with other complementary methods, such as deep learning-based approaches, for improved performance in challenging scenarios.

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 Networks (CNNs):**

**Pros:**

- CNNs can automatically learn features, potentially reducing the need for manual feature engineering.
- They tend to have higher accuracy and better generalization capabilities, especially in complex and varied datasets.

**Cons:**

- CNNs require more computational resources for training and inference compared to the faster cascade-based approach of Viola-Jones.
- Training CNNs often requires a larger amount of labeled data and more extensive parameter tuning.
- CNNs may lack the transparency and interpretability of the cascade approach used in Viola-Jones.