Part I. Implementation (5%):

PART1-1:

```python
# Begin your code (Part 1-1)
"""
This code load images from come dirextories into dataset_train and dataset_test,combine this two set
into dataset and return it
"""
dataset=[]
dataset_train=[]
dataset_test=[]
dataPath_train='data/data_small/train'
dataPath_test='data/data_small/test'
n=1
for i in os.listdir(dataPath_train):
    r=dataPath_train+'/'+str(i)+'/'
    for k in os.listdir(r):
        dataset_train.append((cv2.imread(r+k, cv2.IMREAD_GRAYSCALE), n))
    n-=1
n=1
for i in os.listdir(dataPath_test):
    r=dataPath_test+'/'+str(i)+'/'
    for k in os.listdir(r):
        dataset_test.append((cv2.imread(r+k, cv2.IMREAD_GRAYSCALE), n))
    n-=1
dataset=[dataset_train,dataset_test]
# End your code (Part 1-1)

return dataset
```

PART1-2:

```python
# Here we set N equal to the number of faces to generate a balanced dataset
# Note that we have alreadly save the bounding box of faces into `face_box_list`, you can utilize it for non-face region cropping
for i in range(num_faces):
    # Begin your code (Part 1-2)
    """
    use np.random.randint() to random choose the coordinate if the choosen tube is on the area of the face,rechoose the coordinate
    if ok crop and resize it and put it into  nonface_dataset
    """
    ok=False
    while not ok:
        yrandomt = np.random.randint(0,img_gray.shape[0]-10)
        xrandoml = np.random.randint(0,img_gray.shape[1]-10)
        xrandomr = np.random.randint(xrandoml+1,img_gray.shape[1]-1)
        yrandomb = np.random.randint(yrandomt+1,img_gray.shape[0]-1)
        for lt, rb in face_box_list:
            x0=lt[0]
            y0=lt[1]
            x1=rb[0]
            y1=rb[1]
            if ((x0<=xrandoml and x1>=xrandoml) or ((x0<=xrandomr and x1>=xrandomr)) or
                (y0<=yrandomt and y1>=yrandomt) or (y0<=yrandomb and y1>=yrandomb)):
                continue
        ok=True

    img_crop = img_gray[yrandomt:yrandomb, xrandoml:xrandomr].copy()


    nonface_dataset.append((cv2.resize(img_crop, (19, 19)), 0))

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

PART2:

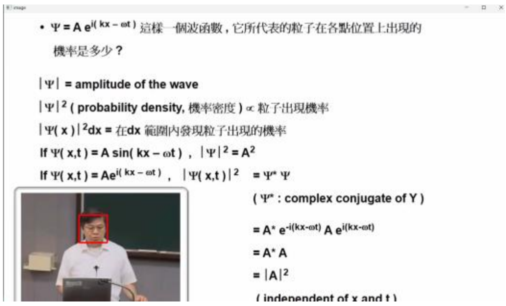
```python
        bestError: The error of the best classifer
    """
    # Begin your code (Part 2)
    """
    initialize the  bestclf and  besterro run every featurevals if (featureVals[i][j]<0 and labels[j]==0)
    or (featureVals[i][j]>=0 and labels[j]==1) error plus the weigth of the sample if erro of this
    erro < besterro update the besterro and the best clf
    """
    bestclf=None
    besterro=float('inf')
    featurenum=featureVals.shape[0]
    datasetnum=featureVals.shape[1]
    for i in range(featurenum):
        error=0
        for j in range(datasetnum):
            if(featureVals[i][j]<0 and labels[j]==0) or (featureVals[i][j]>=0 and labels[j]==1):
                error+=weights[j]
        if error<besterro:
            besterro=error
            bestclf=WeakClassifier(features[i])

    # End your code (Part 2)
    return bestclf, besterro
```

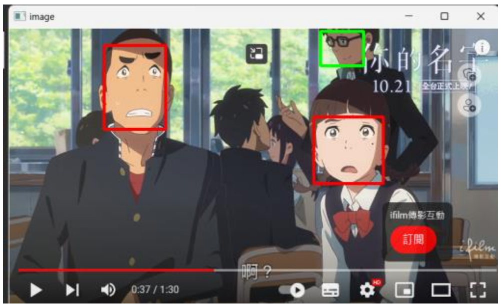
PART4:

```python
    # Begin your code (Part 4)
    # Read the detectData.txt file
    """
    direct path to the folder detect read the text into lines run lines and
    get the coordinate every time read pop the text put it in classify if
    classify is face use green as rectangle if nonface use red continue until the lines empty
    """
    detectpath = 'data/detect/'

    with open(dataPath, 'r') as file:
        lines = file.readlines()
        for line_index in range(len(lines)):
            if len(lines)==0:
                break
            name, num = lines[0].split()
            num = int(num)
            img = cv2.imread(detectpath+name)
            gray_img = cv2.imread(detectpath + name, cv2.IMREAD_GRAYSCALE)
            lines.pop(0)
            for i in range(num):
                x0, y0, width, height = map(int, lines[0].split())
                x1 = x0 + width
                y1 = y0 + height
                if clf.classify(cv2.resize(gray_img[y0:y1, x0:x1], (19, 19))):
                    color = (0, 255, 0)  # Green color for correctly classified
                    cv2.rectangle(img, (x0, y0), (x1, y1), color, thickness=3)
                else:
                    color = (0, 0, 255)  # Red color for misclassified
                    cv2.rectangle(img, (x0, y0), (x1, y1), color, thickness=3)
                lines.pop(0)
            cv2.imshow('image', img)
            cv2.waitKey(0)
            cv2.destroyAllWindows()


    # End your code (Part 4)
```

Part II. Results & Analysis (10%):

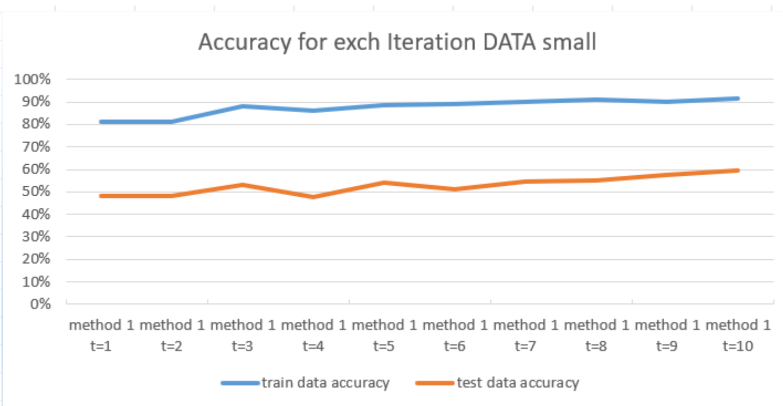
DATASMALL-method1:



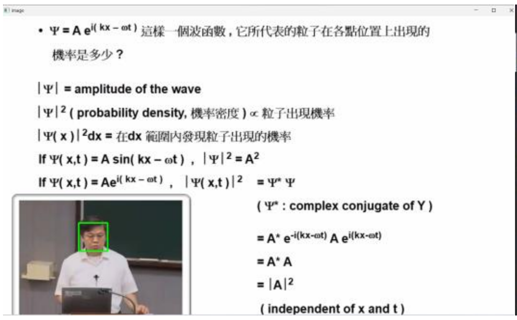


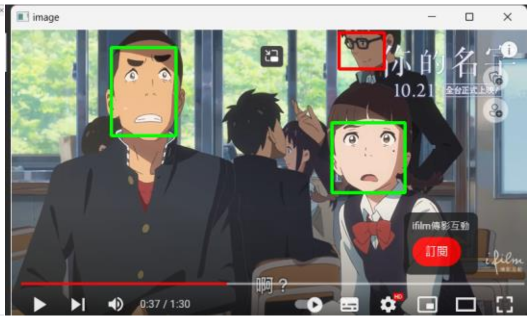

```
]) with accuracy: 0.390000 and alpha: 0.769604
Run No. of Iteration: 7
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 2, 10, 2)], negative regions=[RectangleRegion(5, 4, 10, 2)]) with accuracy: 0.725000 and alpha: 0.719869
Run No. of Iteration: 8
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(12, 11, 5, 1)], negative regions=[RectangleRegion(12, 12, 5, 1)]) with accuracy: 0.360000 and alpha: 0.685227
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: 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), RectangleRegion(2, 11, 2, 2)], negative regions=[RectangleRegion(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
```

200張	train data accuracy	test data accuracy
method 1 t=1	81%	48%
method 1 t=2	81%	48%
method 1 t=3	88%	53%
method 1 t=4	86%	47.50%
method 1 t=5	88.50%	54%
method 1 t=6	89%	51%
method 1 t=7	90%	54.50%
method 1 t=8	91%	55%
method 1 t=9	90%	57.50%
method 1 t=10	91.50%	59.50%

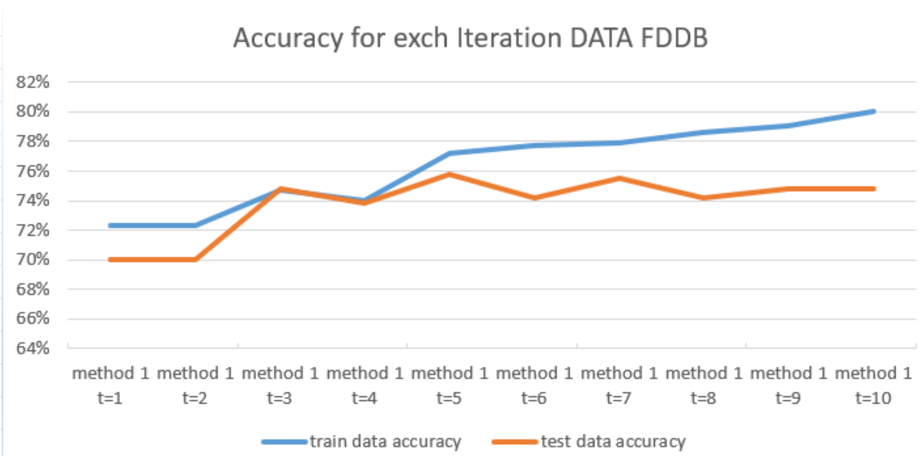


DATAFDDB-method1:





720張/310張	train data accuracy	test data accuracy
method 1 t=1	72%	70%
method 1 t=2	72%	70%
method 1 t=3	75%	75%
method 1 t=4	74%	73.87%
method 1 t=5	77.22%	76%
method 1 t=6	78%	74%
method 1 t=7	78%	75.48%
method 1 t=8	79%	74%
method 1 t=9	79%	74.83%
method 1 t=10	80.00%	74.83%



Based on the results, we can draw the following conclusions:

1. **Training for longer periods generally leads to higher accuracy.**
2. **The accuracy of the training data is typically higher than that of the test data since the classifier is specifically trained on the training data.**
3. **We observe that the accuracy of the training data from "data_small" is higher than that of "data_FDDB." However, the overall performance of the model trained on "data_ FDDB " is better on average.**

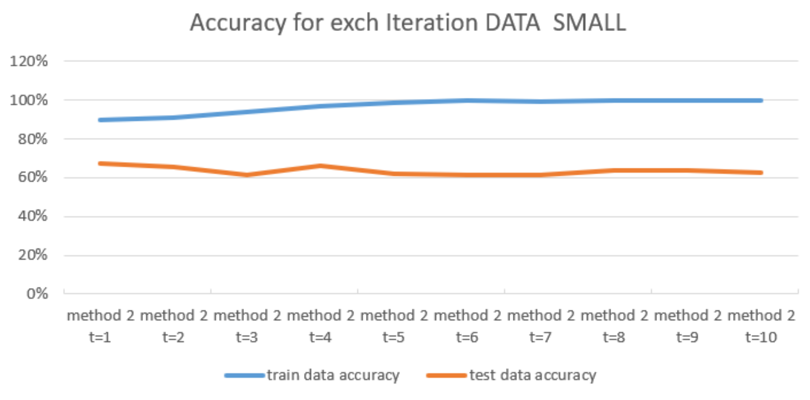
BONUS:

I calculates the error of each weak classifier based on the absolute distance between the normalized feature value and the threshold, considering Adaboost's weights.

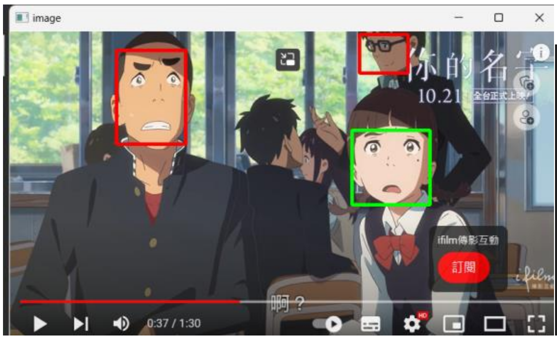
```
This code snippet calculates the weighted error sum by measuring the absolute distance between
the normalized feature value and the threshold. It then combines this with Adaboost's weights
to compute the overall error. In other words, it assesses the performance of each weak classifier
and selects the best one based on its error value and weight in the Adaboost ensemble.
  """
bestnerror = float("inf")
besterro = float("inf")
bestclf = None
for i in tqdm(range(len(features))):
    maxi = np.max(featureVals[i])
    mini = np.min(featureVals[i])
    r = maxi-mini
    Val = (featureVals[i] - mini) / r
    thresholds = np.unique(Val)

    for threshold in thresholds:
        for polarity in [1, -1]:
            pred = polarity * \
                Val < polarity * threshold
            t = weights * (pred != labels)
            error = np.sum(t * np.abs(Val-threshold))

            if error < bestnerror:
                bestnerror = error
                besterro = np.sum(t)
                bestclf = WeakClassifier(
                    features[i], threshold*r+mini, polarity)
""""""
```

100張/100張	train data accuracy	test data accuracy
method 2 t=1	90%	68%
method 2 t=2	91%	66%
method 2 t=3	94%	62%
method 2 t=4	97%	66.00%
method 2 t=5	98.50%	62%
method 2 t=6	100%	62%
method 2 t=7	100%	61.50%
method 2 t=8	100%	64%
method 2 t=9	100%	63.50%
method 2 t=10	100.00%	62.50%

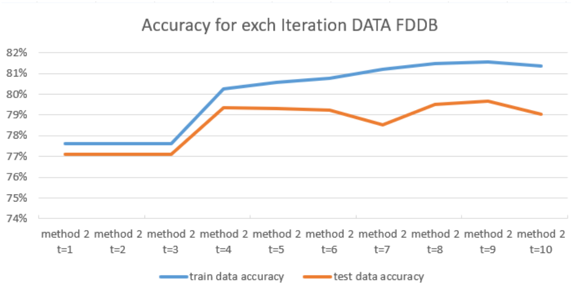
```
ns=[RectangleRegion(0, 5, 18, 3)]) with accuracy: 0.905000 and alpha:
Run No. of Iteration: 10
100%|
hose classifier: Weak Clf (threshold=146, polarity=-1, Haar feature
s=[RectangleRegion(10, 7, 3, 4)]) with accuracy: 0.820000 and alpha:

valuate your classifier with training dataset
alse Positive Rate: 0/100 (0.000000)
alse Negative Rate: 0/100 (0.000000)
ccuracy: 200/200 (1.000000)

valuate your classifier with test dataset
alse Positive Rate: 5/100 (0.050000)
alse Negative Rate: 70/100 (0.700000)
ccuracy: 125/200 (0.625000)
```



720張/310張	train data accuracy	test data accuracy
method 2 t=1	78%	77%
method 2 t=2	78%	77%
method 2 t=3	78%	77%
method 2 t=4	80%	79.35%
method 2 t=5	80.56%	79%
method 2 t=6	81%	79%
method 2 t=7	81%	78.51%
method 2 t=8	81%	80%
method 2 t=9	82%	79.66%
method 2 t=10	81.38%	79.03%



```
Run No. of Iteration: 10
100%|
Chose classifier: Weak Clf (threshold=42, polarity=1, Haar feature
=[RectangleRegion(1, 17, 16, 1)]) with accuracy: 0.305556 and alph

Evaluate your classifier with training dataset
False Positive Rate: 66/360 (0.183333)
False Negative Rate: 68/360 (0.188889)
Accuracy: 586/720 (0.813889)

Evaluate your classifier with test dataset
False Positive Rate: 39/155 (0.251613)
False Negative Rate: 26/155 (0.167742)
Accuracy: 245/310 (0.790323)
```



BY this method the accuracy is more average than the method one. However the accuracy is really high in data small of the training dataset .But I didn't figure out the reason.

Part III. Answer the questions (15%):

1. Please describe a problem you encountered and how you solved it.
   - First, I don't know how to start the Part2 . However I search a lot of information to solve this problem.
   - I don't know how to get the coordinate of myownimage. Finally I use"小畫家"to get the coordinate of myownimage.
2. How do you generate "nonface" data by cropping images?
   - Generating "nonface" data involves selecting background images without faces, cropping out nonface regions from these images, and assigning appropriate labels (e.g., "0" for nonface) for training a nonface classifier.
3. What are the limitations of the Viola-Jones' algorithm?
   - Fixed-size Window: The algorithm works based on a fixed-size window for feature extraction, which can lead to issues with scale variation. Faces at different distances or sizes may not be detected accurately.

   - Sensitive to Lighting Conditions: Viola-Jones is sensitive to variations in lighting conditions, making it less robust in environments with drastic lighting changes or shadows.

   - Limited to Frontal Faces: The algorithm is primarily designed for detecting frontal faces and may struggle with detecting faces in profile or at different angles, reducing its versatility in real-world applications.

4. Based on Viola-Jones' algorithm, how to improve the accuracy except changing the training dataset and parameter T?
    - Optimize the classifier can reduce computation time.
    - Pre-processing the image such as improving the quality og input images and enhance extraction.
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
    - Histogram of Oriented Gradients (HOG) is a technique used for feature extraction in image processing and computer vision.

      Pros:

        1. HOG features provide an efficient representation of image gradients.
        2. : SVM can handle non-linear decision boundaries effectively.

      Cons:

      1.Calculating HOG features can be computationally intensive.