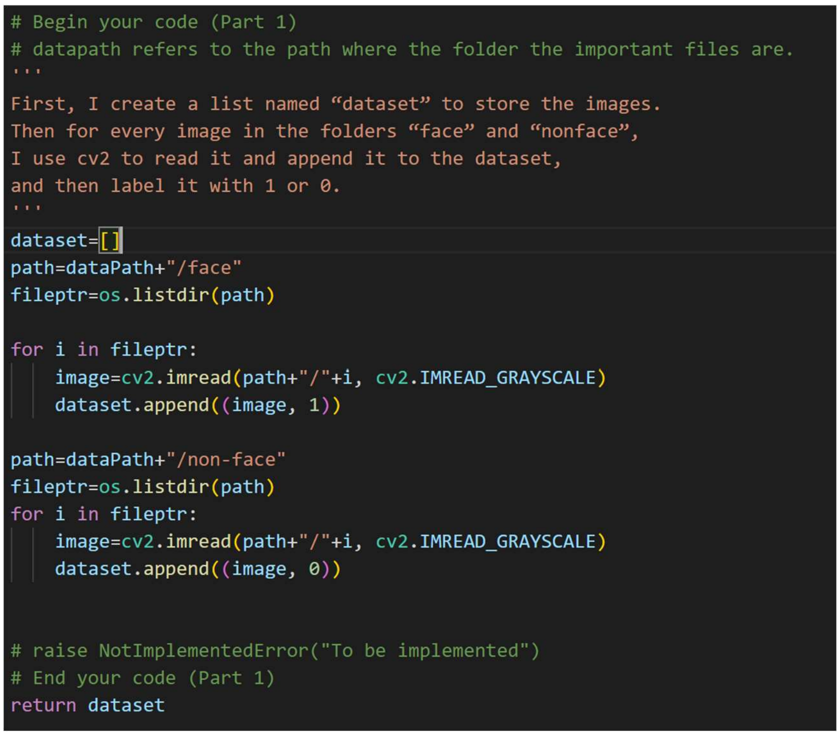
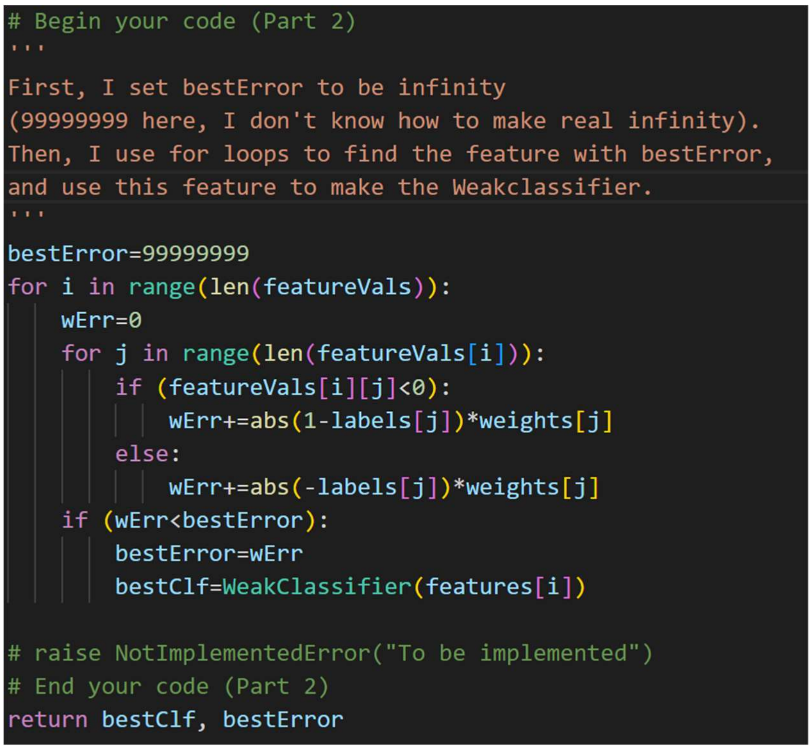
Homework 1: Face Detection

110550085 房天越

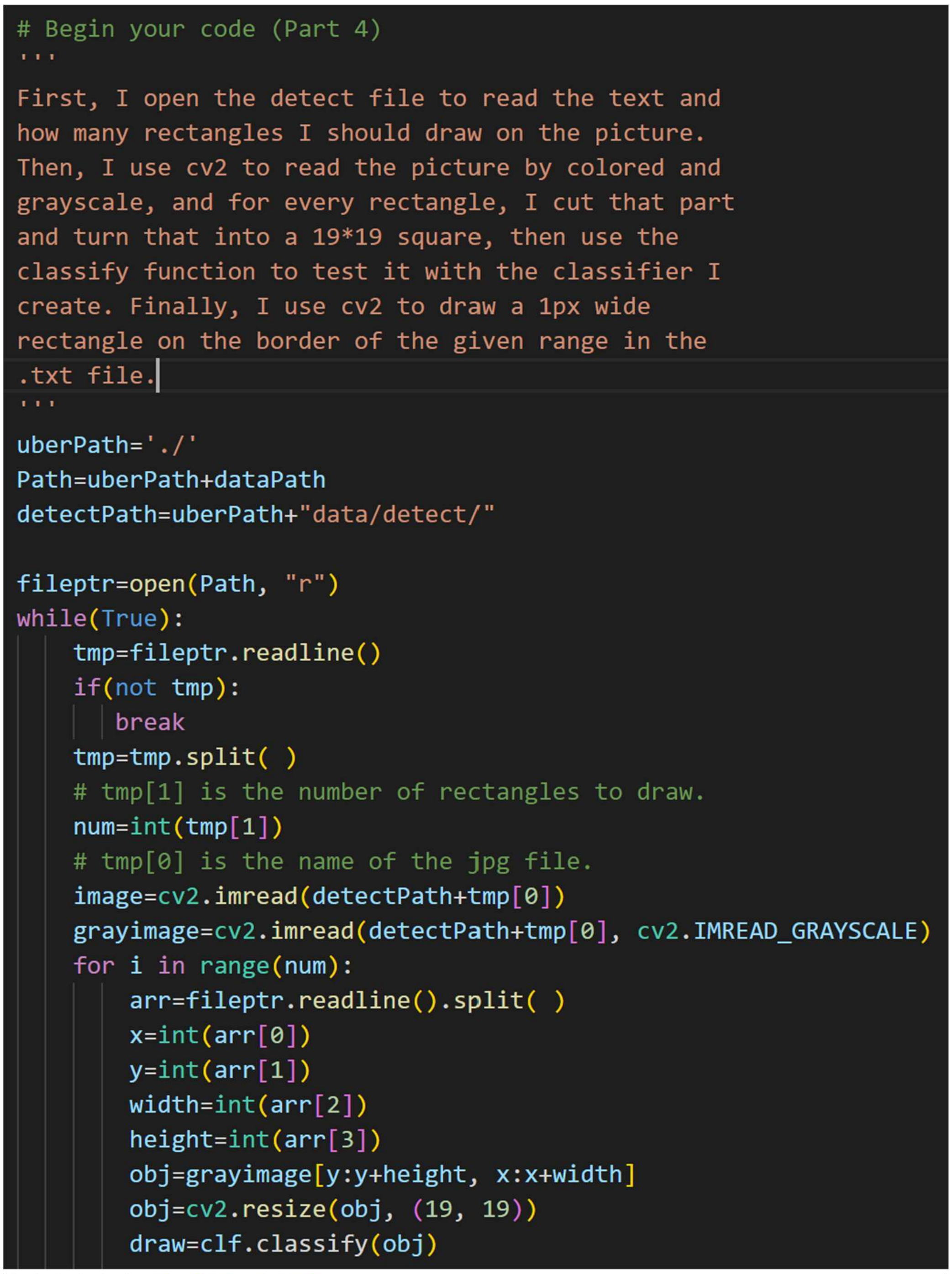
Part I. Implementation (6%):



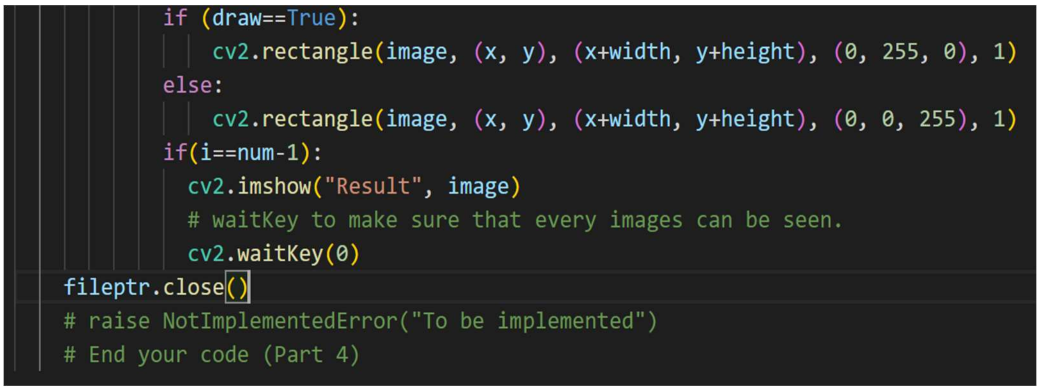
```python
# Begin your code (Part 1)
# datapath refers to the path where the folder the important files are.
'''
First, I create a list named “dataset” to store the images.
Then for every image in the folders “face” and “nonface”,
I use cv2 to read it and append it to the dataset,
and then label it with 1 or 0.
'''
dataset=[]
path=dataPath+"/face"
fileptr=os.listdir(path)

for i in fileptr:
    image=cv2.imread(path+"/"+i, cv2.IMREAD_GRAYSCALE)
    dataset.append((image, 1))

path=dataPath+"/non-face"
fileptr=os.listdir(path)
for i in fileptr:
    image=cv2.imread(path+"/"+i, cv2.IMREAD_GRAYSCALE)
    dataset.append((image, 0))


# raise NotImplementedError("To be implemented")
# End your code (Part 1)
return dataset
```



```python
# Begin your code (Part 2)
'''
First, I set bestError to be infinity
(99999999 here, I don't know how to make real infinity).
Then, I use for loops to find the feature with bestError,
and use this feature to make the Weakclassifier.
'''
bestError=99999999
for i in range(len(featureVals)):
    wErr=0
    for j in range(len(featureVals[i])):
        if (featureVals[i][j]<0):
            wErr+=abs(1-labels[j])*weights[j]
        else:
            wErr+=abs(-labels[j])*weights[j]
    if (wErr<bestError):
        bestError=wErr
        bestClf=WeakClassifier(features[i])

# raise NotImplementedError("To be implemented")
# End your code (Part 2)
return bestClf, bestError
```



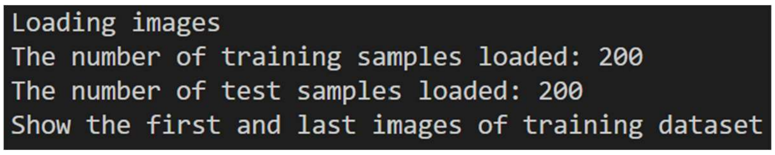
```python
# Begin your code (Part 4)
'''
First, I open the detect file to read the text and
how many rectangles I should draw on the picture.
Then, I use cv2 to read the picture by colored and
grayscale, and for every rectangle, I cut that part
and turn that into a 19*19 square, then use the
classify function to test it with the classifier I
create. Finally, I use cv2 to draw a 1px wide
rectangle on the border of the given range in the
.txt file.
'''
uberPath='./'
Path=uberPath+dataPath
detectPath=uberPath+"data/detect/"

fileptr=open(Path, "r")
while(True):
    tmp=fileptr.readline()
    if(not tmp):
        break
    tmp=tmp.split( )
    # tmp[1] is the number of rectangles to draw.
    num=int(tmp[1])
    # tmp[0] is the name of the jpg file.
    image=cv2.imread(detectPath+tmp[0])
    grayimage=cv2.imread(detectPath+tmp[0], cv2.IMREAD_GRAYSCALE)
    for i in range(num):
        arr=fileptr.readline().split( )
        x=int(arr[0])
        y=int(arr[1])
        width=int(arr[2])
        height=int(arr[3])
        obj=grayimage[y:y+height, x:x+width]
        obj=cv2.resize(obj, (19, 19))
        draw=clf.classify(obj)
```



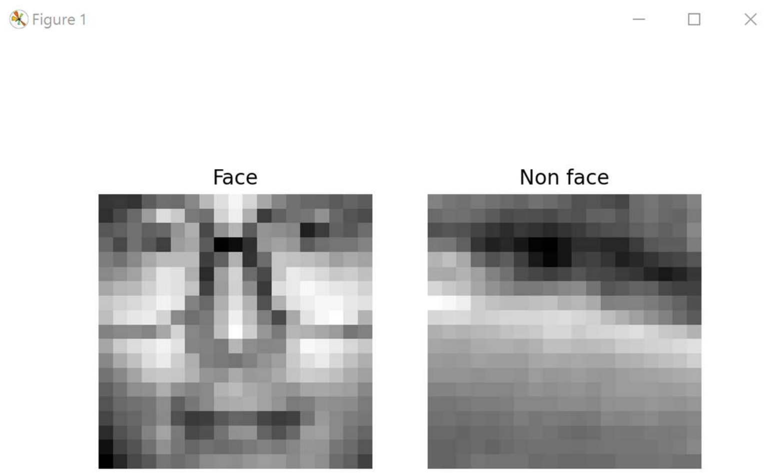
```python
        if (draw==True):
            cv2.rectangle(image, (x, y), (x+width, y+height), (0, 255, 0), 1)
        else:
            cv2.rectangle(image, (x, y), (x+width, y+height), (0, 0, 255), 1)
        if(i==num-1):
            cv2.imshow("Result", image)
            # waitKey to make sure that every images can be seen.
            cv2.waitKey(0)
fileptr.close()
# raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

Part II. Results and Analysis (12%):

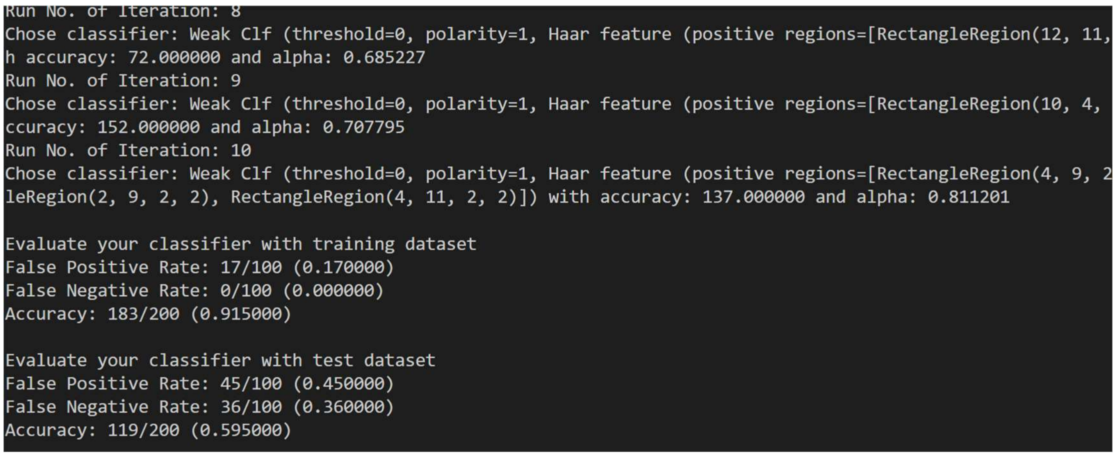
Part1:



```
Loading images
The number of training samples loaded: 200
The number of test samples loaded: 200
Show the first and last images of training dataset
```

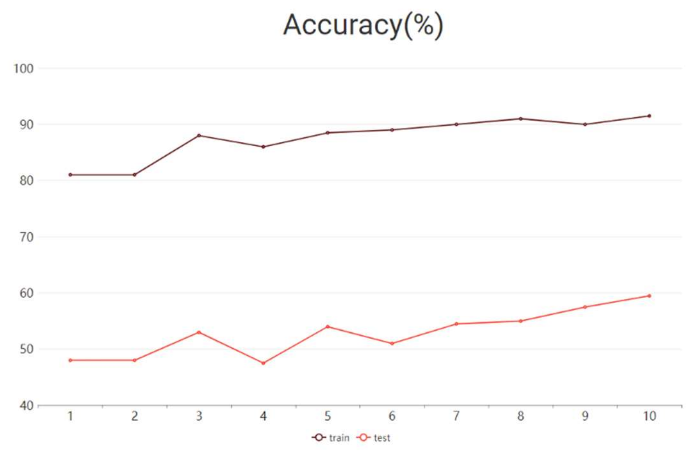


Part2:

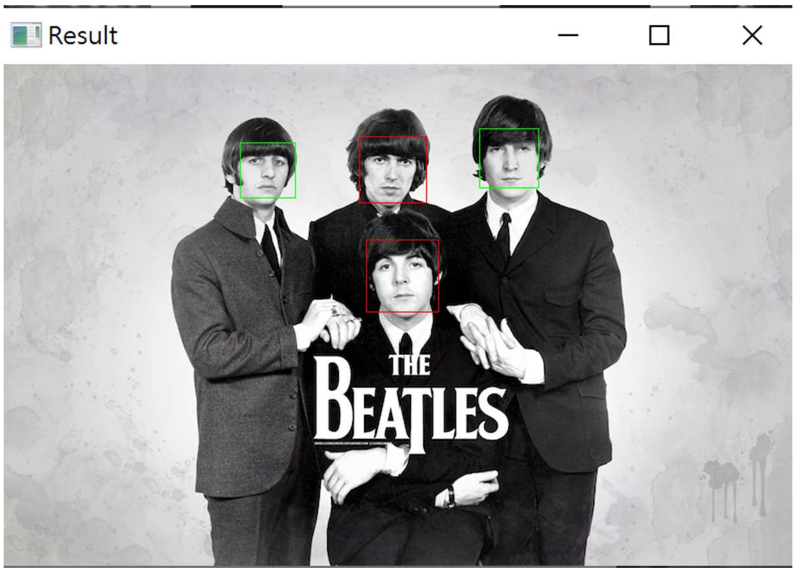


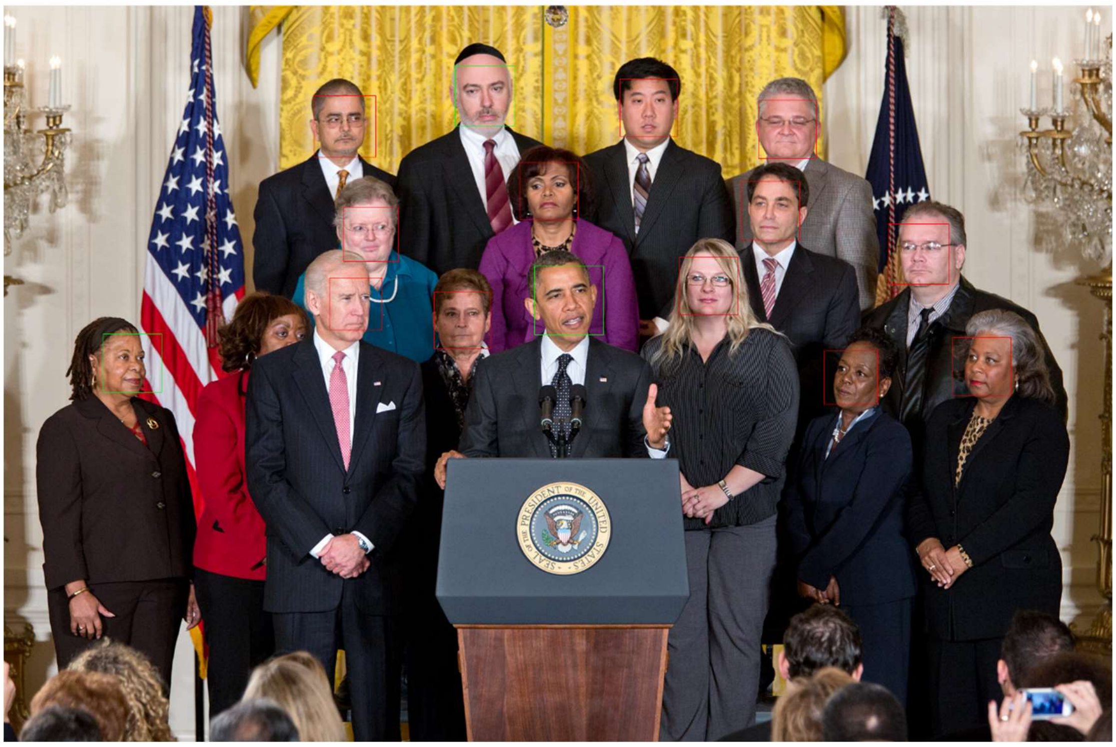
```
Run No. of Iteration: 8
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(12, 11,
h accuracy: 72.000000 and alpha: 0.685227
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 4,
ccuracy: 152.000000 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
leRegion(2, 9, 2, 2), RectangleRegion(4, 11, 2, 2)]) with accuracy: 137.000000 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)
```

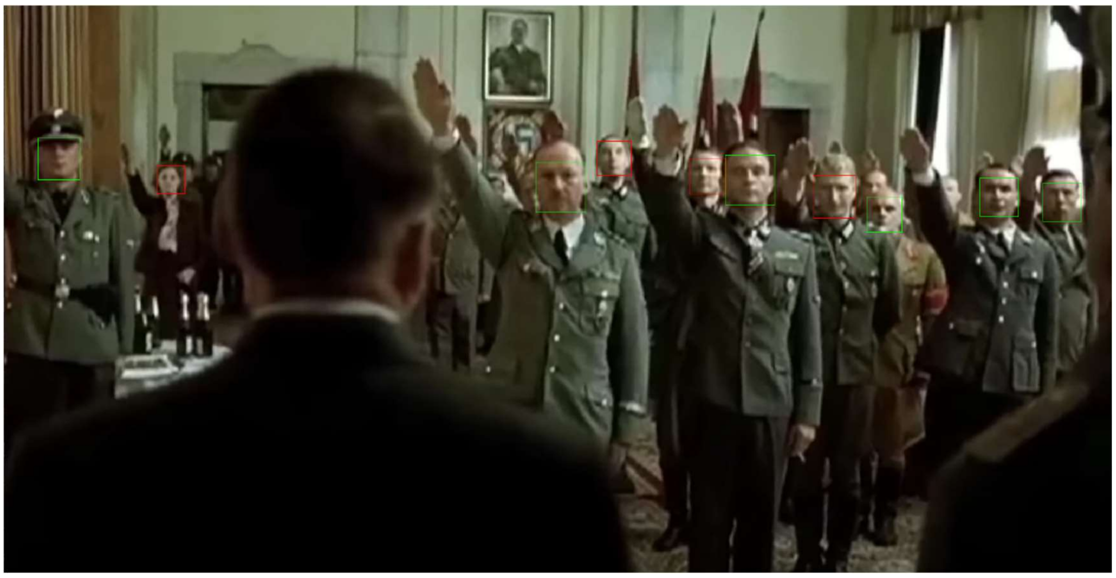
Part3: (The number below represents the number of T)



Part4:







Analysis:

From part 3, we could know that the accuracy tends to increase by the increase of T, however, it also has somewhere to fall.

In addition, I tried once at T=50, the accuracy in the training data went to 100%, but it got only 54% in test data, so larger T doesn't always mean good performance.

PartIII. Answer the questions (12%):

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

   Since I am a completely noob in Python, I had a lot of problem finishing this homework. The problem that took me the longest time was how to read the data. I first run it on MacOS, but I found it difficult to orient the file, so I tried again to do it on Colab, this time the code was able to run, but for unknown reason the picture it showed in Part1 is not the first and the last pictures. Lastly, I switched to Windows, and it could finally go well.

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

   I. It takes a really long time to finish.
   II. It is restricted to binary classification.
   III. If something covers the face then it would go wrong.

IV. Hard to detect faces that are not in frontal view.

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

   In Viola-Jones' algorithm, there are two parameters we could modify, which are Threshold and Polarity. By modifying these two parameters, we could better know the properties of the WeakClassifiers, and make every classifier do the detection more accurately, thus we could improve the accuracy.

4. **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.**

   A popular face detection method is called Deep Convolutional Neural Networks, or DCNN/CNN for short. The concept of this method is relied on convolution, when we want to determine the similarity between a part of the picture and the whole picture, we can simply multiply the similarity of pixels, add them up, and then divide it by the number of pixels, let white=1, black=-1, if it is completely the same, then we would get the result=1, if completely different, we would get -1. By repeating this method with different features, we could finally finish the convolution.
   After convolution, we also need to do Pooling, after pooling the number of pixels would be downgraded to 1/4 of the original one, however, it could still save the range and the features of the original figure. That is, after pooling, we could focus more on whether the test data contain the features we want, not only focus on whether the features are on the according places.

   Compared to Adaboost algorithm, the pros and cons are as follows:
   Pros:
   1. DCNN gives a very high accuracy, even may be higher than human eyes, so it clearly has a better performance than Adaboost.
   2. DCNN can handle even the situations in lighting, pose and expression, etc.
   3. DCNN can be used in multi-scale problems.

   Cons:
   1. DCNN requires a large number of training data.

2. DCNN is more computer power-consumed than Adaboost. Which implies a higher cost when performing the algorithm.