109550182 莊婕妤

AI Homework1 Report

Part I. Implementation (6%)

Part 1

```
1. I iterate over the files in directories, dataPath + "/face" and dataPath + "/non-face",
   and use cv2.imread(directory, 0) to read in the image and
   store it as a 2D numpy array of shape (m, n) in img.
2. I loaded img and its classification (1 for face and 0 for non-face) in a tuple and
   appended it to dataset.
"""
dataset = []
for file in os.listdir(dataPath + "/face"):
  img = cv2.imread(os.path.join(dataPath + "/face", file), 0)
  dataset.append((img, 1))

for file in os.listdir(dataPath + "/non-face"):
  img = cv2.imread(os.path.join(dataPath + "/non-face", file), 0)
  dataset.append((img, 0))
# End your code (Part 1)
```

Part 2

```
# Begin your code (Part 2)
"""
1. Declare a list called errors to store the errors
2. Calculate the errors for each weak classifier as follows:
   For all the integral images, calculate the absolute difference between its label and
   the value after classification. Next, multiply the difference by its weight.
   And add up all the results.
3. Select the weak classifier having the smallest error as bestClf
4. Return bestClf and its error as bestError
"""
errors = [0] * len(features)
for i in range(len(features)):
  weakClf = WeakClassifier(features[i])
  for j in range(len(iis)):
    errors[i] += weights[j] * abs(weakClf.classify(iis[j]) - labels[j])
bestError = min(errors)
bestClf = WeakClassifier(features[errors.index(min(errors))])
# End your code (Part 2)
return bestClf, bestError
```

109550182 莊婕妤

Part 4

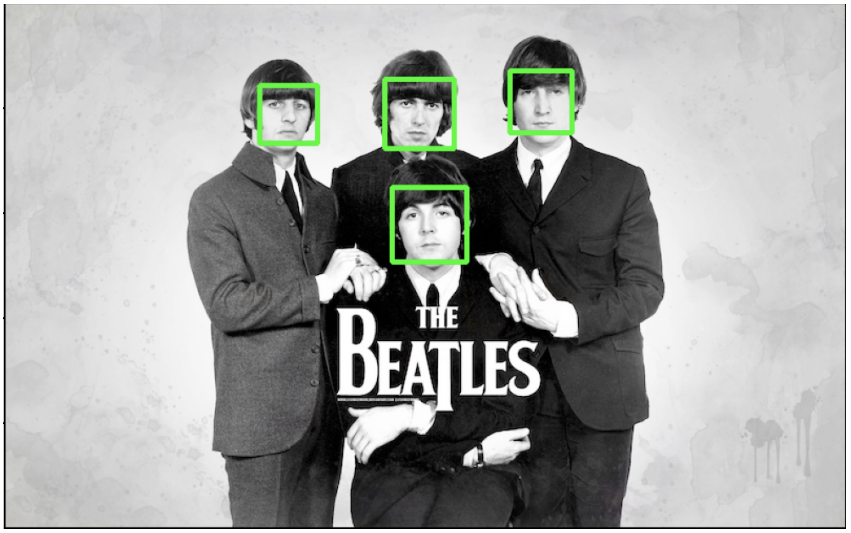
```
# Begin your code (Part 4)
"""
1. Read in the txt file in the given dataPath
2. For every image, store its file name and the number of faces to be detected
3. Read in the coordinates of the boxes
4. Use cv2 to read the image, crop the image with respect to the boxes' coordinate,
   resize the cropped image to 19x19 and then convert them to gray scale
5. Throw those cropped images into the classifier and draw red or green rectangles
   with respect to its output
"""
```

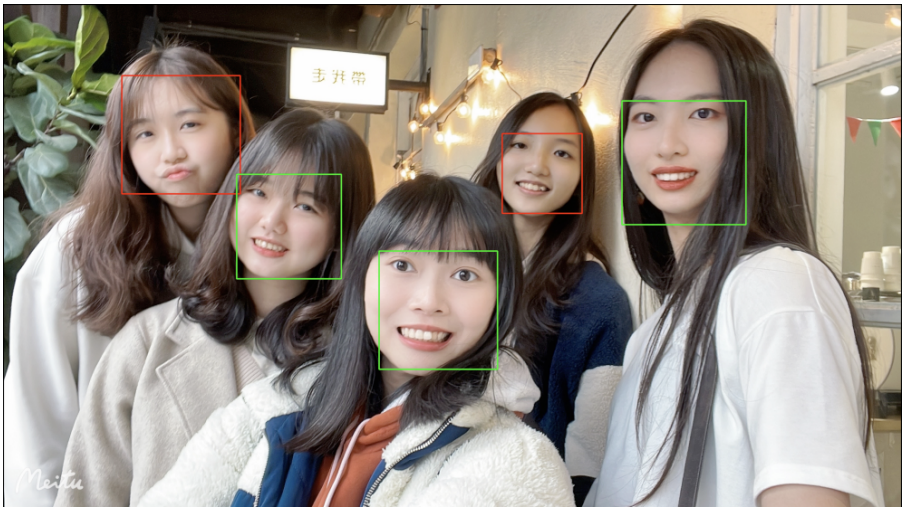
```
file = open(dataPath, 'r')
while True:
  info = file.readline().split(" ")
  if info == ['']: break

  boxes = []
  for i in range(int(info[1])):
    cdt = file.readline().split(" ")
    tmp = [int(cdt[0]), int(cdt[1]), int(cdt[2]), int(cdt[3])]
    boxes.append(tmp)

  images = []
  img = cv2.imread(os.path.join('data/detect/', info[0]))
  for i in boxes:
    cropped = img[i[1]:i[1]+i[3], i[0]:i[0]+i[2]]
    resized = cv2.resize(cropped, (19,19), interpolation=cv2.INTER_NEAREST)
    images.append(cv2.cvtColor(resized,cv2.COLOR_BGR2GRAY))

  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
  for i in boxes:
    if clf.classify(images[boxes.index(i)]) == 1:
      cv2.rectangle(img, (i[0],i[1]), (i[0]+i[2],i[1]+i[3]), (0,255,0), 3)
    else:
      cv2.rectangle(img, (i[0],i[1]), (i[0]+i[2],i[1]+i[3]), (255,0,0), 3)

  plt.imshow(img)
  plt.show()

file.close()
# End your code (Part 4)
```

109550182 莊婕妤

Part II. Results & Analysis (12%)

```
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)
```





109550182 莊婕妤

Analysis

	Train Data Accuracy	Test Data Accuracy
T = 1	81%	48%
T = 2	81%	48%
T = 3	88%	53%
T = 4	86%	47.5%
T = 5	88.5%	54%
T = 6	89%	51%
T = 7	90%	54.5%
T = 8	91%	55%
T = 9	90%	57.5%
T = 10	91.5%	59.5%

My Observation

I observed that as the number of iterations grows, the Train Data Accuracy becomes higher. However, the Test Data Accuracy doesn’t really have the same behavior. Since the test data isn't the set of data used for training, I guess it is because of overfitting. The model may be over trained so it may not perform well on generalized datasets.

109550182 莊婕妤

Part III. Answer the questions (12%)

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

   One major problem I encountered during this homework is that I am not familiar with handling images with Python (or OpenCV). The homework involved many operations with images, such as reading, cropping, resizing and so on. This took me a lot of time searching for documentation related to those operations. I'll also seek help from my classmates for some problems that I couldn't solve by searching the internet.

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

   1) Mainly effective on frontal-view faces
   2) Sensitive to the brightness of images
   3) Takes long time in training

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

   We can improve the accuracy by changing the way of calculating errors. For instance, in the bonus part of this homework, I used another method to calculate the errors (taking the square root of the sum of squared values). Although the accuracy is not higher than the original method, the face detection performance on the second image is better.

4. 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.

   We can derive an algorithm to first find the eyes of the face and then find the eye-brows, nose and mouth within relative positions. If they are all in the right position, then the picture might be a face. The advantage for this algorithm is that it may be faster than the Adaboost algorithm. The disadvantage is that the method is more restricted to detecting frontal-view faces only.

109550182 莊婕妤

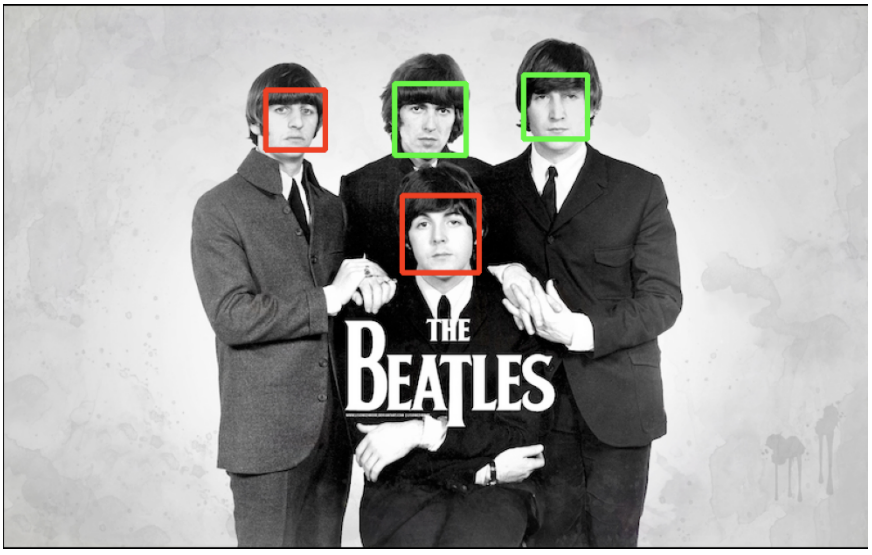
Part IV. Bonus Part (10%)

Implementation

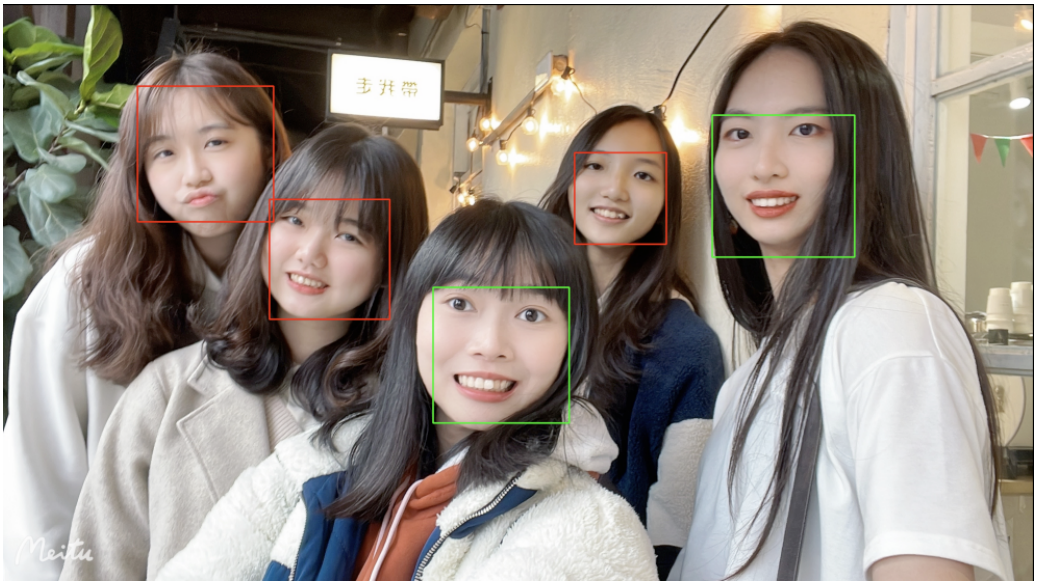
```
"""
Bonus Part Explanation:
I changed the way of getting the errors by
taking the square root of the sum of squared values
"""
errors = [0] * len(features)
for i in range(len(features)):
  weakClf = WeakClassifier(features[i])
  for j in range(len(iis)):
    # errors[i] += weights[j] * abs(weakClf.classify(iis[j]) - labels[j])
    # Bonus: Squared Error
    errors[i] += (weights[j] * abs(weakClf.classify(iis[j]) - labels[j])) ** 2
  errors[i] = math.sqrt(errors[i])
bestError = min(errors)
bestClf = WeakClassifier(features[errors.index(min(errors))])
# End your code (Part 2)
return bestClf, bestError
```

Results

```
Evaluate your classifier with training dataset
False Positive Rate: 30/100 (0.300000)
False Negative Rate: 17/100 (0.170000)
Accuracy: 153/200 (0.765000)

Evaluate your classifier with test dataset
False Positive Rate: 45/100 (0.450000)
False Negative Rate: 62/100 (0.620000)
Accuracy: 93/200 (0.465000)
```



109550182 莊婕妤



109550182 莊婕妤

Analysis

	Train Data Accuracy	Test Data Accuracy
T = 1	81%	48%
T = 2	81%	48%
T = 3	79.5%	57.5%
T = 4	79.5%	57.5%
T = 5	78.5%	50.5%
T = 6	79.5%	50.5%
T = 7	79%	47.5%
T = 8	77.5%	47%
T = 9	82%	51.5%
T = 10	76.5%	46.5%

My Observation

I observed that there is little difference in the Train Data Accuracy as the number of iteration increases, so as the Test Data Accuracy. Also, the accuracy for this new method is lower than the original method. However, it performs better on detecting the second given image.