Report For Intro. To AI HW1

~0812253 陳彥廷

Part 1

```python
import os
import matplotlib.image as mpimg
import numpy as np

def loadImages(dataPath):
    """
    load all Images in the folder and transfer a list of tuples. The first
    element is the numpy array of shape (m, n) representing the image.
    The second element is its classification (1 or 0)
      Parameters:
        dataPath: The folder path.
      Returns:
        dataset: The list of tuples.
    """

    # A list of tuple we need
    listOfImageTuples = []
    for subFolder in os.listdir(dataPath):
        # cd into the folder
        for imagesFileName in os.listdir(os.path.join(dataPath ,subFolder)):
            img = mpimg.imread(os.path.join(dataPath, subFolder, imagesFileName))
            if subFolder == "face" :
                tup = (img ,1) # We make 1 represent face
                listOfImageTuples.append(tup)
            else :
                tup = (img ,0) # We make 0 represent non-face
                listOfImageTuples.append(tup)

    return listOfImageTuples
```

- I use os path to get the images, and transfer them into numpy arrays by applying `matplotlib.image.imread` function.
- I add the label `1` and `0` for classification, and combine with numpy array in a tuple.
- For details, please take a look at the annotations.

Part 2

```python
    def selectBest(self, featureVals, iis, labels, features, weights):
        """
        Finds the appropriate weak classifier for each feature.
        Selects the best weak classifier for the given weights.
          Parameters:
            featureVals: A numpy array of shape (len(features), len(dataset)).
              Each row represents the values of a single feature for each training sample.
            iis: A list of numpy array with shape (m, n) representing the integral images.
            labels: A list of integer.
              The ith element is the classification of the ith training sample.
            features: A numpy array of HaarFeature class.
            weights: A numpy array with shape(len(dataset)).
              The ith element is the weight assigned to the ith training sample.
          Returns:
            bestClf: The best WeakClassifier Class
            bestError: The error of the best classifer
        """
        # Begin your code (Part 2)

        # Get a 2D array for the classified results
        featureClassifiedResult = np.zeros((len(features), len(iis)))
        for fi in range(len(features)):
            # Generate a classifier depend on feature
            classifier = WeakClassifier(features[fi])

            # Use it t oclassified given intergral images
            for imi in range(len(iis)):
                featureClassifiedResult[fi, imi] = classifier.classify(iis[imi])

        # Get the error of every classifiers
        errorOfFeature = []
        for fi in range(len(features)):
            sigmaValue = 0

            # Error formula
            for imi in range(len(iis)):
                sigmaValue += weights[imi] * abs(featureClassifiedResult[fi, imi] - labels[imi])

            errorOfFeature.append(sigmaValue)

        # Choose the classifier with the minimum error
        minimumError = 100000
        minimunErrorIndex = -1
        for i in range(len(errorOfFeature)):
            if errorOfFeature[i] < minimumError:
                minimumError = errorOfFeature[i]
                minimumErrorIndex = i

        # Return the best classifier and its error
        bestClf = WeakClassifier(features[minimumErrorIndex])
        bestError = minimumError

        # End your code (Part 2)
        return bestClf, bestError
```

- First, I create an 2D array to store the classification results.
- Then, I calculate error for each classifiers.
- Last but not least, I choose the one with the smallest error.
- For details, please take a look at the annotations.

Part 3

The following screenshots are the results of testing `T` from `1` to `10`.

- T = 1

```
Evaluate your classifier with training dataset
False Positive Rate: 28/100 (0.280000)
False Negative Rate: 10/100 (0.100000)
Accuracy: 162/200 (0.810000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 55/100 (0.550000)
Accuracy: 96/200 (0.480000)
```

- T = 2

```
Evaluate your classifier with training dataset
False Positive Rate: 28/100 (0.280000)
False Negative Rate: 10/100 (0.100000)
Accuracy: 162/200 (0.810000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 55/100 (0.550000)
Accuracy: 96/200 (0.480000)
```

- T = 3

```
Evaluate your classifier with training dataset
False Positive Rate: 23/100 (0.230000)
False Negative Rate: 1/100 (0.010000)
Accuracy: 176/200 (0.880000)

Evaluate your classifier with test dataset
False Positive Rate: 48/100 (0.480000)
False Negative Rate: 46/100 (0.460000)
Accuracy: 106/200 (0.530000)
```

- T = 4

```
Evaluate your classifier with training dataset
False Positive Rate: 26/100 (0.260000)
False Negative Rate: 2/100 (0.020000)
Accuracy: 172/200 (0.860000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 56/100 (0.560000)
Accuracy: 95/200 (0.475000)
```

- T = 5

```
Evaluate your classifier with training dataset
False Positive Rate: 23/100 (0.230000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 177/200 (0.885000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 43/100 (0.430000)
Accuracy: 108/200 (0.540000)
```

- T = 6

```
Evaluate your classifier with training dataset
False Positive Rate: 22/100 (0.220000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 178/200 (0.890000)

Evaluate your classifier with test dataset
False Positive Rate: 50/100 (0.500000)
False Negative Rate: 48/100 (0.480000)
Accuracy: 102/200 (0.510000)
```

- T = 7

```
Evaluate your classifier with training dataset
False Positive Rate: 20/100 (0.200000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 180/200 (0.900000)

Evaluate your classifier with test dataset
False Positive Rate: 52/100 (0.520000)
False Negative Rate: 39/100 (0.390000)
Accuracy: 109/200 (0.545000)
```

- T = 8

```
Evaluate your classifier with training dataset
False Positive Rate: 18/100 (0.180000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 182/200 (0.910000)

Evaluate your classifier with test dataset
False Positive Rate: 47/100 (0.470000)
False Negative Rate: 43/100 (0.430000)
Accuracy: 110/200 (0.550000)
```

- T = 9

```
Evaluate your classifier with training dataset
False Positive Rate: 20/100 (0.200000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 180/200 (0.900000)

Evaluate your classifier with test dataset
False Positive Rate: 48/100 (0.480000)
False Negative Rate: 37/100 (0.370000)
Accuracy: 115/200 (0.575000)
```

- T = 10

```
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)
```

- It shows that the accuracy of the classifier will increase with the number of iterations in range `1 ~ 10` .

Part 4

```python
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import os
import numpy as np
from PIL import Image

# Formula from numpy form image to change from rgb to gray scale
def rgbToGray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

def detect(dataPath, clf):
    """
    Please read detectData.txt to understand the format. Load the image and get
    the face images. Transfer the face images to 19 x 19 and grayscale images.
    Use clf.classify() function to detect faces. Show face detection results.
    If the result is True, draw the green box on the image. Otherwise, draw
    the red box on the image.
      Parameters:
        dataPath: the path of detectData.txt
      Returns:
        No returns.

    """
    # Begin your code (Part 4)

    # Read lines from txt file
    with open(dataPath,'r') as f:
        txtLines = f.readlines()

    path = "data/detect"
    l = 0

    while l < len(txtLines):
        fileName, people = txtLines[l].split(" ")
        people = int(people)
        image = mpimg.imread(os.path.join(path, fileName))

        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        for n in range(1, people + 1):
```

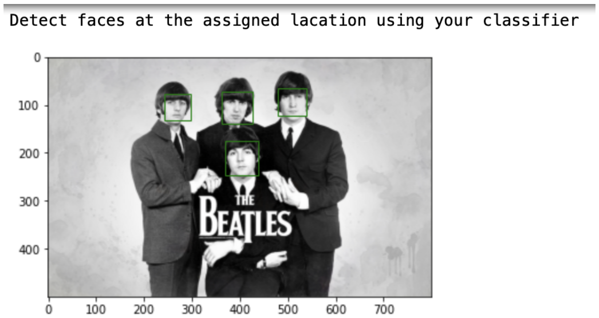
```python
            # Get part of the image which was chosen by the txt file
            x, y, width, height = txtLines[l + n].split(" ")
            x = int(x)
            y = int(y)
            width = int(width)
            height = int(height)

            # make it a small image
            imageCrop = image[y:y+height, x:x+width, :]

            # Apply gray scale
            imageCrop = rgbToGray(imageCrop)

            # Resize the small image
            imageTran = Image.fromarray(imageCrop)
            imageTran = imageTran.resize((19, 19))
            imageCrop = np.array(imageTran)

            # Apply our classifier
            result = clf.classify(imageCrop)

            if result == 1:
                rect = plt.Rectangle((x, y), width, height, fill=False, edgecolor = 'green',linewidth=1)
            else:
                rect = plt.Rectangle((x, y), width, height, fill=False, edgecolor = 'red',linewidth=1)

            # Show the result on the image
            ax.add_patch(rect)
            plt.imshow(image)

        plt.show()
        l += people + 1

    # End your code (Part 4)
```

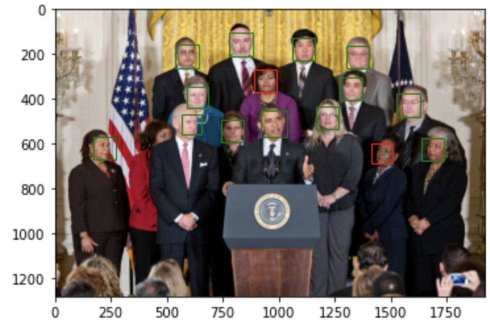
- Let me explain the structure in the txt file first:

```
the-beatles.jpg 4
242 78 55 55
368 175 71 71
480 64 59 59
361 72 66 66
p110912sh-0083.jpg 15
588 347 94 94
1526 376 104 104
744 494 97 97
1166 433 100 100
537 155 106 106
1302 163 100 100
892 270 99 99
917 449 118 118
1272 303 101 101
780 104 103 103
1063 126 99 99
173 566 101 101
1637 572 100 100
1415 595 92 92
564 469 90 90
```

- The first line contain the `image file name` and `the number of face` in the picture.
- In the following n lines `(n = the number of face)` ,we have `x1` , `x2` , `x3` , `x4`.
    - `x1` : Upper left corner x coordinate.
    - `x2` : Upper left corner y coordinate.
    - `x3` : Width of the rectangle.
    - `x4` : Height of the rectangle.

- After getting the require informations, I crop the images into small rectangle , resize it, and send it into the given classifier to get the result.
- I show the result in green box if the return value equal 1, and show in red box if the return value equal to 0.
- The following screenshots are the result :



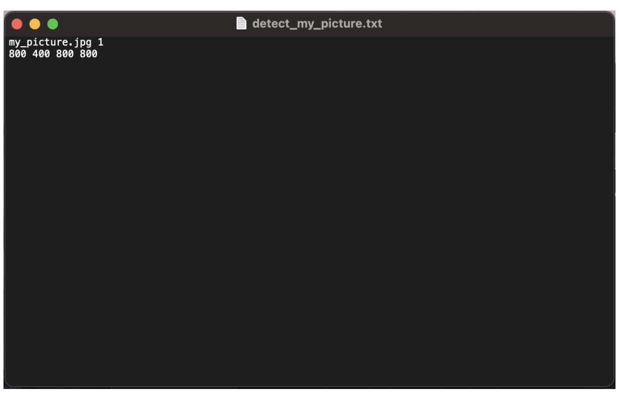


- For details, please take a look at the annotations.

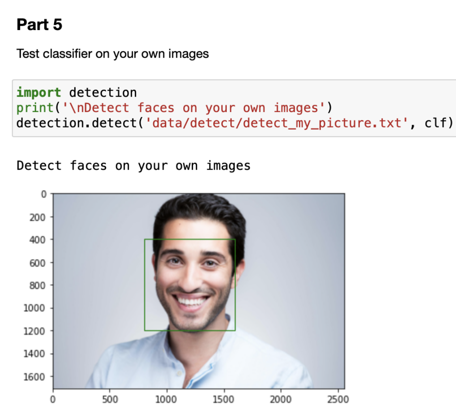
Part 5

- The following screenshot are the result of my own picture, and the txt file I use to input requirements.

- detect_my_picture.txt



- my_picture.jpg result



The Problems I Meet

- Prob: Google Colab sometime cannot print the image, because we use matplotlib for showing image.
- Sol: I switch to anaconda and use jupyter notebook ~