Part 1: Load and prepare your dataset (10%)

code

```
def loadImages(dataPath,byteorder='<'):
    # Begin your code (Part 1)
    dataset = []
    for class_dir_name in ["face","non-face"]:
        class_dir = os.path.join(dataPath, class_dir_name)
        images_in_class = os.listdir(class_dir)
        for image_file in images_in_class:
          with open(class_dir+'/'+image_file, 'rb') as f:
              buffer_ = f.read()
          try:
            header, width, height, maxval = re.search(
              b"(^P5\s(?:\s*#.*[\r\n])*"
              b"(\d+)\s(?:\s*#.*[\r\n])*"
              b"(\d+)\s(?:\s*#.*[\r\n])*"
              b"(\d+)\s(?:\s*#.*[\r\n]\s)*)", buffer_).groups()
          except AttributeError:
            raise ValueError("Not a raw PGM file: '%s'" % image_file)
          label= 1 if (class_dir_name=="face") else 0
          img = np.frombuffer(buffer_,dtype='u1' if int(maxval) < 256 else byteorder+'u2',count=int(width)*int(height),offset=len(header)).reshape(19, 19)
          dataset.append((img,label))
          #data augmentation
          if(dataPath[-5:]=="train" and label==1):
            img_flip = cv.flip(img,1)
            img_rot = rotate(img)
            dataset.append((img_flip ,label))
            dataset.append((img_rot ,label))
    return dataset
```

(1) os.listdir() returns a list of image names in the face/non-face directories.

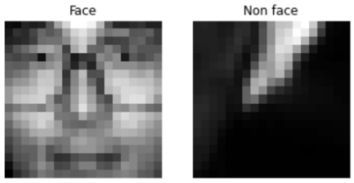
(2) Use regular expression to parse pgm images based on pgm format, and store the gray values in numpy array.

(3) The format of output is (np,label).

Reference : https://stackoverflow.com/questions/7368739/numpy-and-16-bit-pgm

result

```
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
```



Part 2: Implement Adaboost algorithm (30%)

code

```
def selectBest(self, featureVals, iis, labels, features, weights):
    # Begin your code (Part 2)
    bestError = float('inf')
    bestClf = WeakClassifier()
    # For each feature, set 3 kinds of Clf based on 3 thresholds
    for h in range(len(features)):
      mean_fVals = ( sum(featureVals[h]) / len(featureVals[h]) )
      sorted_fVals = np.sort(featureVals[h])
      mid_idx = (int)(len(featureVals[h]) / 2)
      mid_fVals = sorted_fVals[mid_idx]
      for t in [0, mean_fVals, mid_fVals]:
        #------------------------------initialize---------------------------------
        tmpClf = WeakClassifier(features[h],threshold = t,polarity = 1)
        error_h = 0
        #---------------------------sum up the error---------------------------
        for data in range(len(labels)):
          if((featureVals[h][data] < 0) and (t != 0)):
            tmpClf.set_polarity(-1)
          else:
            tmpClf.set_polarity(1)
          error = weights[data] * abs(tmpClf.classify(iis[data])-labels[data])
          error_h += error
        #---------------------select min(bestError, error_h)--------------------
        if(error_h < bestError):
          bestError=error_h
          bestClf = tmpClf
        #-----------------------------end of one Clf-----------------------------
    return bestClf, bestError
```

(1) For each feature indexed h, build 3 default classifier with respect to 3 kinds of threshold : zero, mean(featureVals[h]), and midian(featureVals[h]).

(2) The classifier will invert the polarity if (threshold!=0) and (featureVals <0).
(I know this is non-sense and would make the classifier unstable, but I still applied this since I the model would thereby perform better. QQ???)

(3) Under the feature[h], calculate the error of each data and sum them up.
That is, error_ h = $\sum_i w_i \left| h_j(x_i) - y_i \right|$

(4) If (error_h < bestError), set the current classifier as bestClf.

(5) Looping through all features, we will get a bestClf with bestError.

result

```
========== T = 1 ==========

Start training your classifier
Computing integral images
Building features
Applying features to dataset
Selecting best features
Selected 5171 potential features
Initialize weights
Run No. of Iteration: 1
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(8, 0, 1, 3), RectangleRegion(7, 3, 1, 3)], negative regions=[RectangleRegion(7, 0, 1, 3), RectangleRegion(8, 3, 1, 3)]) with accuracy: 162.000000 and alpha: 1.450010

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 = 10 ==========

Start training your classifier
Computing integral images
Building features
Applying features to dataset
Selecting best features
Selected 5171 potential features
Initialize weights
Run No. of Iteration: 1
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(8, 0, 1, 3), RectangleRegion(7, 3, 1, 3)], negative regions=[RectangleRegion(7, 0, 1, 3), RectangleRegion(8, 3, 1, 3)]) with accuracy: 162.000000 and alpha: 1.450010
Run No. of Iteration: 2
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 8, 2, 9)], negative regions=[RectangleRegion(2, 8, 2, 9)]) with accuracy: 156.000000 and alpha: 1.286922
Run No. of Iteration: 3
Chose classifier: Weak Clf (threshold=-371, polarity=-1, Haar feature (positive regions=[RectangleRegion(13, 6, 1, 6)], negative regions=[RectangleRegion(13, 12, 1, 6), RectangleRegion(13, 0, 1, 6)]) with accuracy: 135.000000 and alpha: 1.181340
Run No. of Iteration: 4
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(10, 8, 1, 1)], negative regions=[RectangleRegion(9, 8, 1, 1)]) with accuracy: 155.000000 and alpha: 1.145447
Run No. of Iteration: 5
Chose classifier: Weak Clf (threshold=-64, polarity=1, Haar feature (positive regions=[RectangleRegion(9, 2, 3, 3)], negative regions=[RectangleRegion(9, 5, 3, 3)]) with accuracy: 150.000000 and alpha: 1.232193
Run No. of Iteration: 6
Chose classifier: Weak Clf (threshold=-64, polarity=1, Haar feature (positive regions=[RectangleRegion(9, 2, 3, 3)], negative regions=[RectangleRegion(9, 5, 3, 3)]) with accuracy: 150.000000 and alpha: 1.633147
Run No. of Iteration: 7
Chose classifier: Weak Clf (threshold=-522, polarity=1, Haar feature (positive regions=[RectangleRegion(6, 0, 9, 5)], negative regions=[RectangleRegion(6, 5, 9, 5)]) with accuracy: 155.000000 and alpha: 2.240964
Run No. of Iteration: 8
Chose classifier: Weak Clf (threshold=-522, polarity=1, Haar feature (positive regions=[RectangleRegion(6, 0, 9, 5)], negative regions=[RectangleRegion(6, 5, 9, 5)]) with accuracy: 155.000000 and alpha: 3.834358
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=-522, polarity=1, Haar feature (positive regions=[RectangleRegion(6, 0, 9, 5)], negative regions=[RectangleRegion(6, 5, 9, 5)]) with accuracy: 155.000000 and alpha: 6.613223
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=-522, polarity=1, Haar feature (positive regions=[RectangleRegion(6, 0, 9, 5)], negative regions=[RectangleRegion(6, 5, 9, 5)]) with accuracy: 155.000000 and alpha: 4.639874

Evaluate your classifier with training dataset
False Positive Rate: 24/100 (0.240000)
False Negative Rate: 21/100 (0.210000)
Accuracy: 155/200 (0.775000)

Evaluate your classifier with test dataset
False Positive Rate: 22/100 (0.220000)
False Negative Rate: 33/100 (0.330000)
Accuracy: 145/200 (0.725000)
```

Part 3: Additional experiments (20%)

code

(1) Test by changing T from 1 to 10

(2) In the meantime, record the results of utils.evaluate() in train_log and test_log.

```
# store utils.evaluate() info in train_log and test_log
train_log = []
test_log = []
for T in range(1,11):
  print('==================== T =',T,'============================\n')
  print('Start training your classifier')
  clf = adaboost.Adaboost(T = T)
  clf.train(trainData)

  print('\nEvaluate your classifier with training dataset')
  train_false_positives, train_all_negatives, train_false_negatives, train_all_positives, train_correct = utils.evaluate(clf, trainData)
  train_log.append([train_false_positives, train_all_negatives, train_false_negatives, train_all_positives, train_correct])

  print('\nEvaluate your classifier with test dataset')
  test_false_positives, test_all_negatives, test_false_negatives, test_all_positives, test_correct = utils.evaluate(clf, testData)
  test_log.append([test_false_positives, test_all_negatives, test_false_negatives, test_all_positives, test_correct])
  print('\n')
```

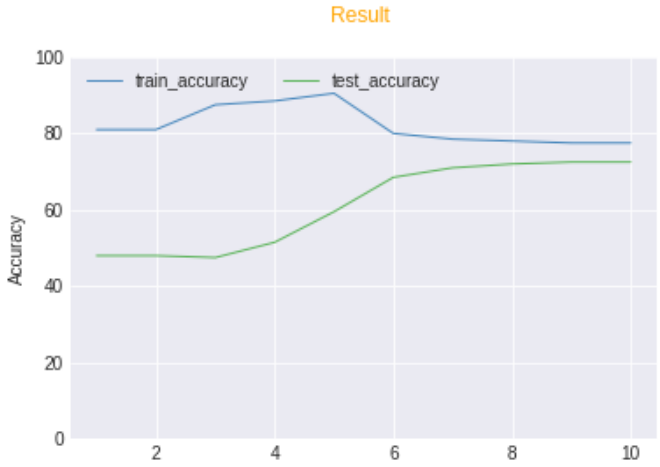
(3) Plot the line chart of the false positive/negative rate of train and test data.

```
import matplotlib.pyplot as plt
import pandas as pd

# Make data frame
train_false_positive_list = []
train_false_negative_list = []
test_false_positive_list = []
test_false_negative_list = []
for i in range(10):
  train_false_positive_list.append(train_log[i][0]*100/train_log[i][1])
  train_false_negative_list.append(train_log[i][2]*100/train_log[i][3])
  test_false_positive_list.append(test_log[i][0]*100/test_log[i][1])
  test_false_negative_list.append(test_log[i][2]*100/test_log[i][3])
df=pd.DataFrame({ 'T': range(1,11), 'train_false_positive_rate': train_false_positive_list, 'train_false_negative_rate': train_false_negative_list, 'test_false_positive_rate': test_false_positive_list, 'test_false_negative_rate': test_false_negativ

# Change the style of plot and create a color palette
plt.style.use('seaborn-darkgrid')
palette = plt.get_cmap('Set1')

# set axis
axes = plt.gca()
axes.set_ylim([-5,100])
# Plot multiple lines
num=0
for column in df.drop('T', axis=1):
    num+=1
    plt.plot(df['T'], df[column], marker='', color=palette(num), linewidth=1, alpha=0.9, label=column)
# Add legend and title
plt.legend(loc=2, ncol=2)
plt.title("Result \n", loc='center', fontsize=12, fontweight=0, color='orange')
plt.xlabel("T")
plt.ylabel("Rate")

# Show the graph
plt.show()
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', -1)
print(df)
```

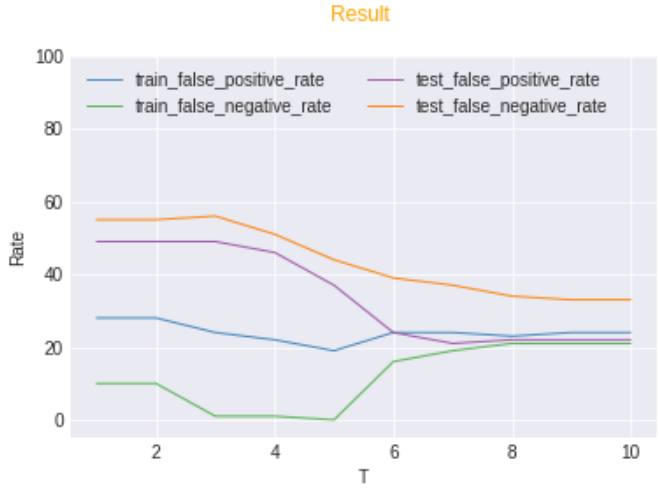
(4) Plot the line chart of the accuracy of train and test data.

```
train_accuracy_list = []
test_accuracy_list = []
# Make a data frame
train_len = len(trainData)
test_len = len(testData)

for i in range(0,10):
  train_accuracy_list.append(train_log[i][4]*100/train_len)
  test_accuracy_list.append(test_log[i][4]*100/test_len)
df=pd.DataFrame({ 'T': range(1,11), 'train_accuracy': train_accuracy_list, 'test_accuracy': test_accuracy_list })

# Change the style of plot and create a color palette
plt.style.use('seaborn-darkgrid')
palette = plt.get_cmap('Set1')

# set axis
axes = plt.gca()
axes.set_ylim([0,100])
# Plot multiple lines
num=0
for column in df.drop('T', axis=1):
    num+=1
    plt.plot(df['T'], df[column], marker='', color=palette(num), linewidth=1, alpha=0.9, label=column)

# Add legend and title
plt.legend(loc=2, ncol=2)
plt.title("Result \n", loc='center', fontsize=12, fontweight=0, color='orange')
plt.xlabel("T")
plt.ylabel("Accuracy")

# Show the graph
plt.show()
print(df)
```

result



T	train_accuracy	test_accuracy
1	81.0	48.0
2	81.0	48.0
3	87.5	47.5
4	88.5	51.5
5	90.5	59.5
6	80.0	68.5
7	78.5	71.0
8	78.0	72.0
9	77.5	72.5
10	77.5	72.5



T	train_false_positive_rate	train_false_negative_rate	test_false_positive_rate	test_false_negative_rate
1	28.0	10.0	49.0	55.0
2	28.0	10.0	49.0	55.0
3	24.0	1.0	49.0	56.0
4	22.0	1.0	46.0	51.0
5	19.0	0.0	37.0	44.0
6	24.0	16.0	24.0	39.0
7	24.0	19.0	21.0	37.0
8	23.0	21.0	22.0	34.0
9	24.0	21.0	22.0	33.0
10	24.0	21.0	22.0	33.0

(1) The accuracy of train data is maximized on T = 5, from which the accuracy started to decreased.

(2) The growth rate of test data became slow as T increased after T = 6, which means that doing more iterations would become less effective on improving accuracy.

(3) The more iterations we went through (specifically from 1 to 10), the better this model perform on test data.

Part 4: Detect face (15%)

code

```
def detect(dataPath, clf):

    #info from txt
    image_name = ""
    face_cnt = 0
    faces_loc = []
    # variables used in reading line
    line_state = 1
    line_cnt = 0

    file = open(dataPath,"r")
    for line in file:
      line = line.replace("\n", "")
      #---------------line_state = 1 if it's a new image---------------
      if (line_state == 1 ):
        #initialize
        line_cnt = 0
        faces_loc.clear()
        #read image name and face_cnt
        fields = line.split(" ")
        image_name=fields[0]
        face_cnt=(int)(fields[1])
        line_state = 2
      #---------------Otherwise, line_state = 2------------------------
      elif (line_state == 2):
        # read the x,y,w,h info
        fields = line.split(" ")
        faces_loc.append([fields[0],fields[1],fields[2],fields[3]])
        line_cnt+=1
        #-------start printing rectangles on faces------------
        if (line_cnt == face_cnt):
          image = cv.imread('data/detect/' + image_name)
          image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
          image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
          #-------iterate through all faces------------
          for face in faces_loc:
            x=(int)(face[0])
            y=(int)(face[1])
            w=(int)(face[2])
            h=(int)(face[3])
            face_image = image_gray[y:y+h,x:x+w]
            resized_image = cv.resize(face_image, (19, 19), interpolation=cv.INTER_AREA)
            #inhance the image
            clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
            inhanced_image = clahe.apply(resized_image)
            if ( clf.classify(resized_image) or clf.classify(inhanced_image) ):
              cv.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 6)
            else:
              cv.rectangle(image, (x, y), (x+w, y+h), (255, 0,0), 6)
          #----------------------------------------------
          plt.imshow(image)
          plt.axis('off')
          plt.show()
          line_state =1
    file.close()
```

(1) After parsing the .txt file, read the location (x, y), width, and height of each face.

(2) Upon all face information is read, use cv.imread() to read image.

(3) Keep the original image and create a grayscale one (→image_gray).

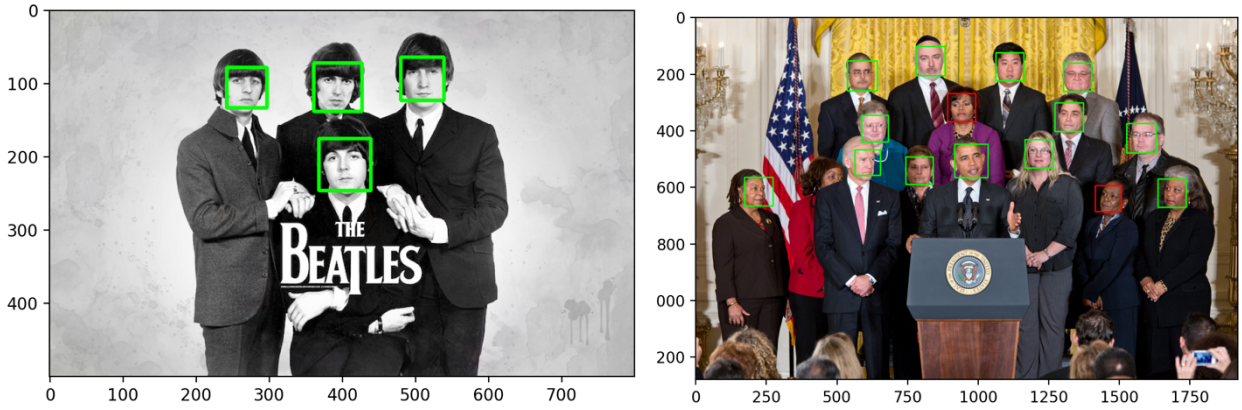
(4) Cut off faces from grayscale images (→face_image).

(5) Resized the face_images to 19 x 19 (→resized_image).

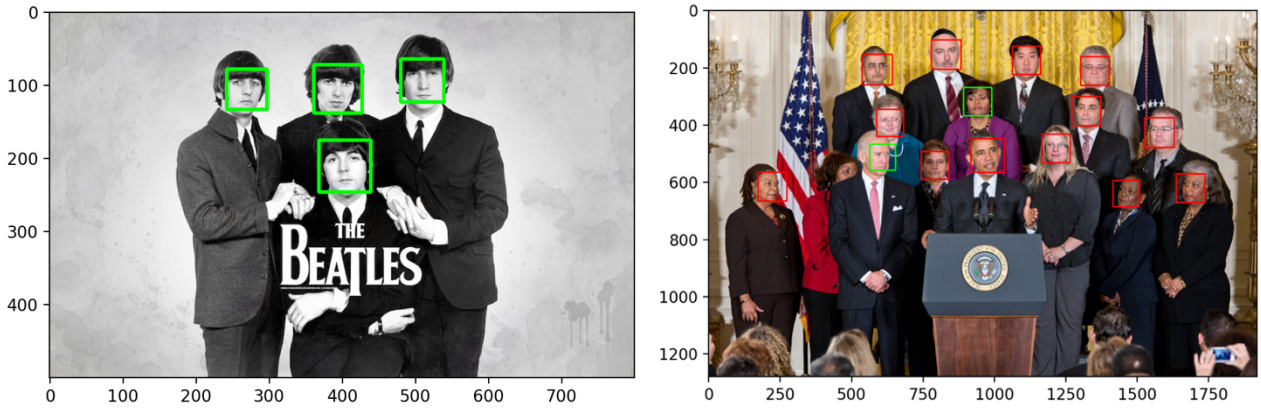
(6) Increase the contrast of resized_imageas (→inhanced_images).

(7) Green rectangles appear if either resized_images or inhanced_images are classified as faces; otherwise there will be red rectangles.

result (T=1)



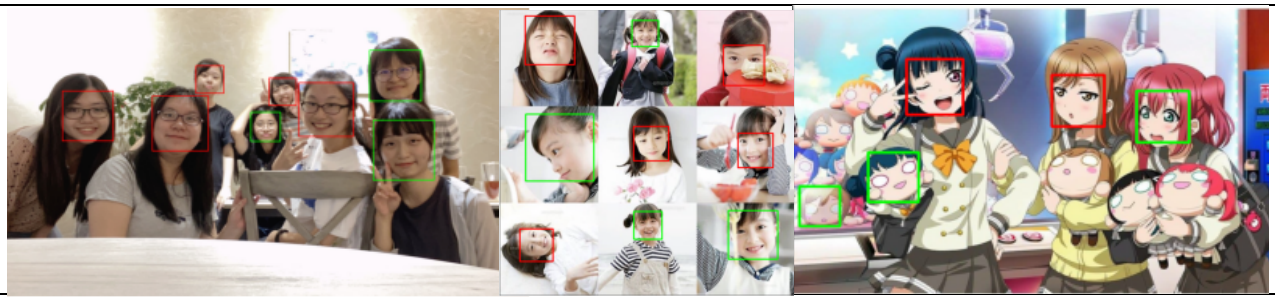
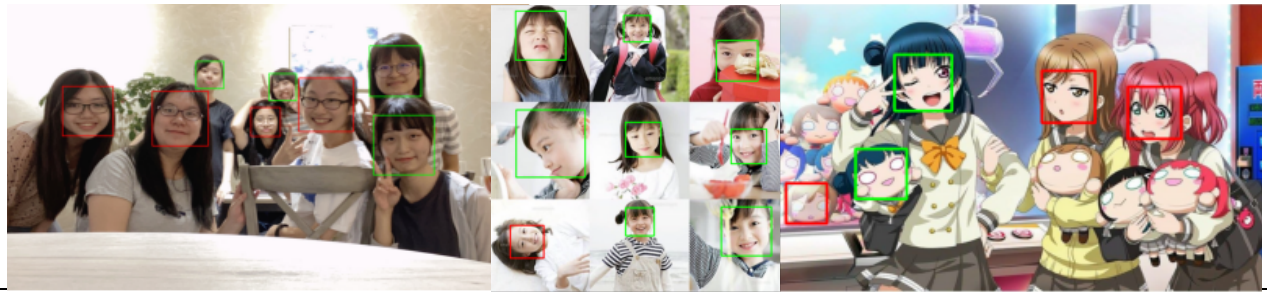
result (T=10)



(1) I don’t know why the right picture failed to classify properly, but I guess the reason is that the model had become more sensitive as T grew (overfitting).

(2) Or maybe the images processed after cv.cvtColor() still show different pattern from grayscale pictures.

Part 5: Test classifier on your own images (5%)

explanaion
(1) Select 3 images and write their information to 'data/detect/MyOwnImages.txt'. (2) The first picture didn't perform well. I guess it's because that the image size is quite larger than the training data, or people with glasses are harder to be identified. (3) The model performed better on identifying little 橋本環奈. I think it might because of the smaller picture size as well as her clear facial features and fair skin.
result (T=1)

result (T=10)


Part 6: Implement another classifier

I only do modifications on threshold as part 2 show.

Part 7: Problems you meet and how you solve them

(1) Failed to import cv2 on VScode > use" import cv2, from cv2 import cv2 as cv" instead.

(2) Failed to call a null classifier bestClf = WeakClassifier() > passed none to the parameter, that is, WeakClassifier(…, feature = none, …)

(3) To store the accuracy of each iteration, the utils.py is also modified as mentioned in Part 3.

(4) I did data augmentation with flip() and rotate(), but nothing is improved.

(5) Locating face from scratch is too difficult, as a result I used face detector found on GitHub to automatically located the faces first, then modified them in details.

```
!pip install git+git://github.com/PnS2019/pnslib.git
from pnslib import utils as utils2
import cv2
import matplotlib.pyplot as plt

face_cascade = cv2.CascadeClassifier(utils2.get_haarcascade_path('haarcascade_frontalface_default.xml'))
image = cv2.imread('data/detect/IMG_01.jpg', cv2.IMREAD_UNCHANGED)

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(image_gray)

for (x, y, w, h) in self_defined_faces:
  cv.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 10)
  print(x,y,w,h)

plt.figure()
plt.imshow(image)
plt.show()
```