Picture Crawling and Recognition Report

包家豪 515030910264

1. Background

Acemap is an academic large database and visual academic map system. It can provide some useful information of famous authors by the visual interface, such as affiliations, fields, impact factors. In this project, my purpose is to search and select suitable photos of the authors on the Internet.

The simple method is to download the photos on the authors' homepage. However, it's found that some authors did not put their photos on the homepage. Therefore, we decide to search and collect photos through google, then select the suitable photo. It's easy to find a lot of photos of the authors on the Internet. However, they may be group photos of full-size photos that are not suitable to be put on the personal homepage. In addition, the same keyword may find photos of several people, in which case we need to determine which photo is the right person. These are two major problems that I need to solve.

There is some related work. The acemap group can provide the basic information of the authors, including names, affiliations and impact factors. In addition, they have already searched some photos on the authors' homepage and these photos will be used as ground truth.

Figure 1. We want to search photos of author through google with some keywords and then select the suitable one. The right photo is a group photo and not suitable to be put on the homepage, while the left is what we want.

2. Experiment methods

2.1 Picture crawling

At first, I used the crawler tool to download the photos of the authors and create a database. The python module is called google-images-download. People can set the keyword, number and type of photos and download the photos they need through google

image using this module. I chose one thousand famous authors in computer science and downloaded fifteen photos for every author. Actually, I obtained a 5-G database with 15000 photos in total.

```
from google_images_download import google_images_download   #importing the library

response = google_images_download.googleimagesdownload()   #class instantiation

arguments = {"keywords":"Polar bears,baloons,Beaches","limit":20,"print_urls":True}   #creating list of arguments
paths = response.download(arguments)   #passing the arguments to the function
print(paths)   #printing absolute paths of the downloaded images
```

Figure 2. The python module: google-images-download

2.2 Face recognition

In this part, my purpose is to find how many faces in one download photo. If the photo has no face in it, it must be useless and should be removes. However, if the photo has several faces, it may be a group photo which also can't meet our requirement. In fact, only the photo has one face in it is what we need since the photo is used to be put on the authors' homepage. I used the python module called face-recognition. The module can output the number of faces in the input photos. Using this module, I traversed the entire database and removed all the useless and unsuitable photos.

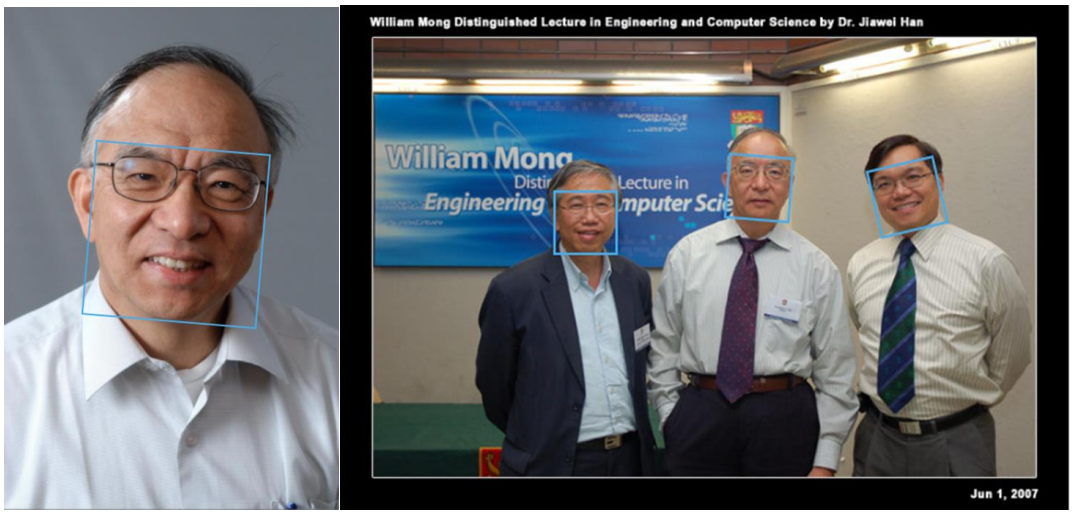

Figure 3. Through detecting the faces in the photo, we can easily judge if the photo is a group photo.

```
import face_recognition
image = face_recognition.load_image_file("your_file.jpg")
face_locations = face_recognition.face_locations(image)
```

Figure 4. The python module: face-recognition. The face-location will return the number of faces in the detected photos. If the face-location doesn't equal one, then remove the photo.

2.3 Clustering

Now we want to pick the suitable photo for each author from the database. Considering that photos downloaded with the same keyword may contain several people, I first cluster photos and choose the suitable one in the category that contains the most photos. This part includes three steps.

2.3.1 Feature points

First, detect the feature points of faces. Using the python module called dlib, I select 64 feature points in one face.

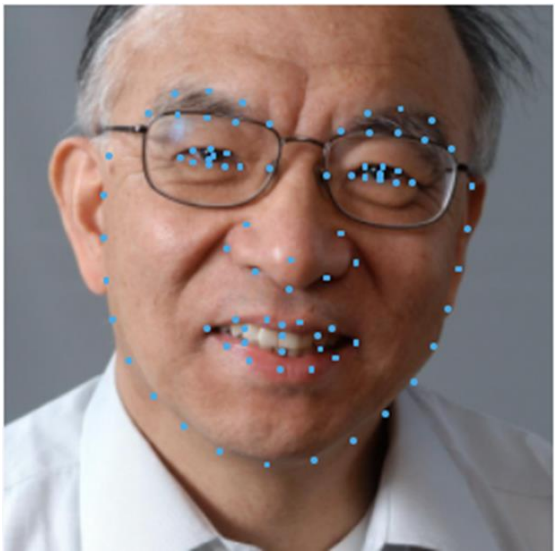

Figure 5. Feature points

2.3.2 Vector space

Second, map the feature points to a vector space. Using the pre-train model, I map the feature points to 128 dimensional vector space. the figure below shows the result of one vector in the space.

```
F:\Python\my_dlib_codes\face_recognition>python my_face_recogniton.py
Processing file: F:\Python\my_dlib_codes\face_recognition\faces\jobs.jpg
Number of faces detected: 1
face 0; left 184; top 64; right 339; bottom 219
-0.179784
0.15487
0.10509
-0.0973604
-0.19153
0.000418252
-0.0357536
-0.0206766
0.129741
-0.0628359
....
```

Figure 6. Vector space

2.3.3 Distance

Third, calculate the distance between two faces in the vector space. If the distance is

smaller than 0.6, then consider the two faces belong to the same person. I use Euclidean distance and the formula is shown below.

$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Figure 7. Euclidean distance

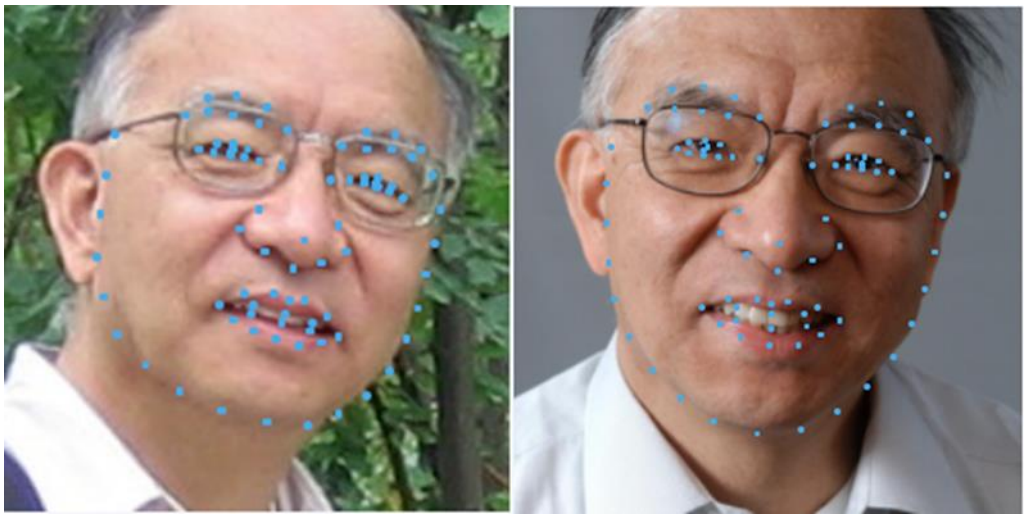

Figure 8. The distance between the two faces is 0.48, so the model will classify them together.

3. Result and analysis

Through this project, I figure out a feasible method of searching and selecting photos for authors from google. Using the basic information of authors provided by the acemap group, I obtained one thousand authors' photos at last.

By comparison with photos provided by the acemap group, I roughly estimated the accuracy of this method. The result is shown in the table below.

Table 1. Experiment result

keyword	total	correct	wrong
name	50	44	6
name + affiliation	50	47	3

As we can see, when we use name as keyword, the accuracy is about 88%, while when we use name and affiliation as keyword, the accuracy is increasing and reach 94%. This is because the affiliation can solve the duplicate name problem. Take the author michael i jordan for example. If we using his name as keyword, the downloaded photos will mostly belong to the famous basketball player Jordan. When we search his name and affiliation in google, the result becomes much better. What's more, there are many author have the same name and using name as keyword cannot distinguish them. However, they basically will not be in the same affiliation. So, using affiliation as keyword, we can effectively download the photos we want. In actual operation, I used name as keyword to download 9 photos and used name and affiliation to download 6 photos.

Appendix

Python scripts

downloads.py

```
# -*- coding: UTF-8 -*-
from google_images_download import google_images_download
import csv
import os

reader = csv.reader(open('csTopAuthor.csv'))
response = google_images_download.googleimagesdownload()
for i,row in enumerate(reader):
    if i ==1000:break
    author = row[0]
    ID = row[1]
    arguments = {"keywords":author,"limit":20,"print_urls":True}
    paths = response.download(arguments)
    oldname = './downloads/'+author
    newname = './downloads/'+ID
    os.rename(oldname,newname)
```

face_recognition.py

```
# -*- coding: utf-8 -*-
import face_recognition
import os
from PIL import Image
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

downloads_dir = "./downloads"
author_ID_list = os.listdir(downloads_dir)
err = 0

for author_ID in author_ID_list:
    photolist = os.listdir(downloads_dir+'/'+author_ID)
    print author_ID+'\n'
    for photo in photolist:
        try:
```

```
                image =
face_recognition.load_image_file(downloads_dir+'/'+author_ID+'/'+photo)
                face_locations = face_recognition.face_locations(image)
                print len(face_locations)
                if len(face_locations) != 1:
                    os.remove(downloads_dir+'/'+author_ID+'/'+photo)
            except:
                os.remove(downloads_dir+'/'+author_ID+'/'+photo)
                print 'error'
                err += 1
print err
```

clustering.py

```
# coding: utf-8
import sys
import os
import dlib
import glob
import cv2

current_path = '.'
model_path = current_path + '/model/'
shape_predictor_model = model_path + '/shape_predictor_5_face_landmarks.dat'
face_rec_model = model_path + '/dlib_face_recognition_resnet_model_v1.dat'

namelist = os.listdir(current_path+'/downloads')
for name in namelist:
    face_folder = current_path+'/downloads' + '/'+ name
    output_folder = current_path+ '/output'+'/'+ name
    if not os.path.isdir(output_folder):
        os.makedirs(output_folder)

    detector = dlib.get_frontal_face_detector()
    shape_detector = dlib.shape_predictor(shape_predictor_model)
    face_recognizer = dlib.face_recognition_model_v1(face_rec_model)

    descriptors = []
    images = []

    facelist = os.listdir(face_folder)

    for f in facelist:
```

```
        print('Processing file：{}'.format(f))
        img = cv2.imread(face_folder+'/'+f)
        if img is None:continue

        img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        dets = detector(img2, 1)
        print("Number of faces detected: {}".format(len(dets)))

        for index, face in enumerate(dets):
            shape = shape_detector(img2, face)
            face_descriptor = face_recognizer.compute_face_descriptor(img2, shape)

            descriptors.append(face_descriptor)
            images.append((img2, shape))

labels = dlib.chinese_whispers_clustering(descriptors, 0.5)
print("labels: {}".format(labels))
num_classes = len(set(labels))
print("Number of clusters: {}".format(num_classes))

face_dict = {}
for i in range(num_classes):
    face_dict[i] = []
for i in range(len(labels)):
    face_dict[labels[i]].append(images[i])

for key in face_dict.keys():
    file_dir = os.path.join(output_folder, str(key))
    if not os.path.isdir(file_dir):
        os.makedirs(file_dir)

    for index, (image, shape) in enumerate(face_dict[key]):
        file_path = os.path.join(file_dir, 'face_' + str(index))
        print file_path
        dlib.save_image(image, file_path+'.jpg')
```