Schorlar Photo Mining

Lv Ruiliang
515030910208

Abstract

The AceMap system is a novel academic system to analyze the big scholarly data and visualize the results through a map approach. while it is still under construction. For example, in the author profile pages, there are no scholar photos. Mining scholar photos from the Internet could be a challenging work, considering the large scale of image data and the fact that there is no deterministic correspondence of name and photo on the Internet. With the help of search engine, Crawler and recently face recognition techniques, this task could be tackled efficiently with a sounding accuracy.

1. Background

In recent years, the AceMap system [1] has made meaningful achievements in presenting academic data. AceMap integrates several algorithms in the field of network analysis and data mining, and then displays the information in a clear and intuitive way, aiming to help the researchers facilitate their work. By far, AceMap has implemented the following functions: dynamic citation network display, paper clustering, academic genealogy, author and conference homepage, etc.

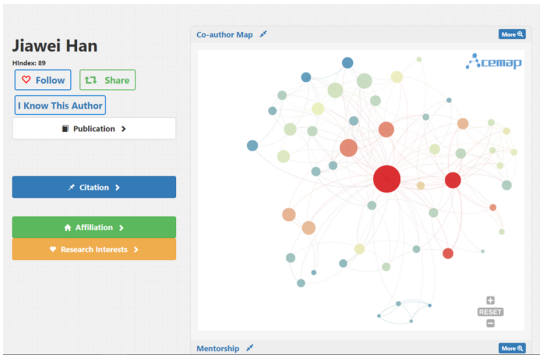


Figure 1: Previous author profile on AceMap

However, there is still much work to complete the system. Take the author profile page as an example. For each scholar in computer science related areas, there is a corresponding author profile that shows basic information about the author, including research interests, publication, citation and affiliation, even the novel co-author map. Nevertheless, there is no photo of the scholar on the system yet, as shown in Figure 1. As we know, photo is probably the most representative profile about a person. Thus it is worthwhile to incorporate the scholar photos to the system. To the best of my knowledge, this is the first project to mine scholar photos from the Internet.

1.1. Introduction

The objective of this task is quite straightforward: Given a list of CS top authors with their basic information, we need to seek for the most representative photo for each scholar, as shown in Figure 2. The basic information for each scholar include its name, an unique id in Acemap system (8-digit hexadecimal) and its affiliation. The given list consists of 230,000 scholars in a csv file. The output format should indicate the mapping between scholar id and corresponding photo. (Since there are exists lots of name conflicts, it is unambiguous to use unique scholar id.)

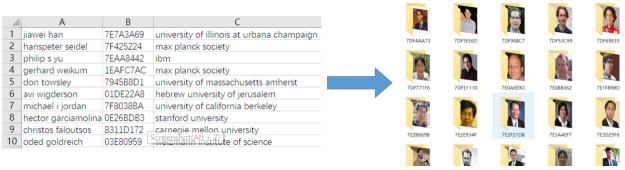


Figure 2: Introduction of the task

1.2. Challenges

Although the task seems quite intuitive, there exist several challenges.

- **Large scale of data**: There is more than 200,000 scholars in computer science related areas.
- **Lack of ground-truth labels**: There is no existing dataset for training, making it unsuitable to use supervised learning approach.
- **Name conflict**: Scholars may share the same name with famous stars or other scholars.

1.3. Related Work

Web image retrieval With the fast development of Web technology, Web image searching has become a popular application, which returns relevant images given certain query. In the early days, Web images were manually labeled with text based on which they could be indexed and retrieved [2]. However, with the burst of large-scale image data, manually labeling becomes impractical. Several image retrieval techniques like content-based image retrieval (CBIR) were introduced to overcome the difficulty ([3], [4]). However, it is difficult and inefficient to extract semantic patterns from images. Current practical commercial image search engine exploit surrounding text aside images and corresponding statistic data [5]. Those commercial image search engine like Google does provide high quality results, but the accuracy still needs further technical processing for our task.

Face recognition Face recognition technologies has been under intensive research, especially when recent deep learning technology emerges ([6], [7], [8]). The general structure of face recognition is illustrated in Figure 3. Within this pipeline, face detection is to count the number of faces and locate all the faces in an image. The most difficult and tricky part is feature extraction, where original RGB / Grayscale images are projected in certain dimensions and mapped to vectors. Those dimensions generated a space where the similarity between faces could be measured by their geometry distances. There are already several work accomplish this mapping efficiently ([9], [10]), with an excellent performance on widely acknowledged dataset. Different from face detection, face recognition further typically uses the extracted features to verify whether two faces are the same person or not.

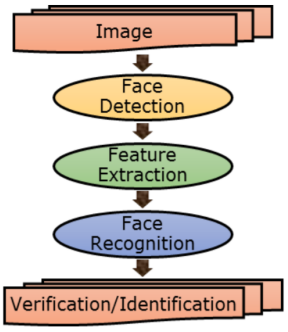


Figure 3: General structure of face recognition [11]

2. Approach

With the help of search engine, crawler and face recognition technique, my designed approach could be divided into three steps: The first step is to build a photo library that contains a set of photos for each scholar in the scholar list. The second step is photo cleaning, which is to analyze whether a photo is valid and remove invalid photos. The last step is to select the best representative photo for each scholar and build the final database.

2.1. Building Photo Library

As mentioned before, the objective for this step is to download a set of photos for each scholar. There are several technologies that helps to complete the objective, such as search engine, Python crawler and remote server.

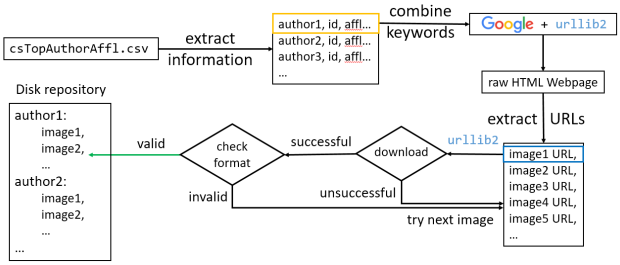


Figure 4: Overview pipeline to build photo library

The overview framework of this step is shown in Figure 4, which is quite intuitive and easy for understanding. After extracting scholar information, keywords are combined and sent to Google image search engine as search terms. For each query, I use the Python `urllib2` library to download the raw Web page. Then image URLs are extracted by analyzing the HTML source code using regular expression techniques. For each image, I continue to use the `urllib2` module to download images from the Web. Afterwards, images are checked whether they are in desired format (*eg.* ".jpg", ".png" etc). Due to various unexpected events like network broke down or time out, the download could fail or the image format is invalid. For these cases, the program divert to the next image and try downloading until the downloaded number meets the requirements (typically 10 20 images for each scholar).

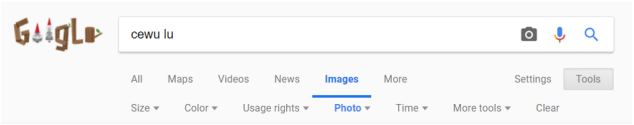


Figure 5: Setting the type tag to"photo"

There are some implementation details for this step that needs to be mentioned. About the choice of search engine, popular search engines include Google, Baidu, Bing etc. Google could provide results with higher quality, but using Google in mainland needs VPN or Shadowsocks settings and could be quite unstable. Using Baidu or Bing has

lower latency, however, the results are obviously not as accurate as Google and contain a lot of noise. My solution is to deploy my program directly on a remote foreign server, which could exploit Google at a much lower delay. Indeed it enjoys a speed 10 times faster than downloading via my local PC. One trick to improve the search quality is to set the type of image to "photo", as shown in Figure 5. This trick could greatly filter those uncorrelated noise out. To tackle the name conflict problem, I combine name and affiliation as search term, typically 10 images searched by name and 5 images searched by name + affiliation.

```
    except UnicodeEncodeError as e:
        download_status = 'fail'
        download_message = "UnicodeEncodeError on an image...trying next one..." + " Error: " + str(e)

    except TimeoutError as e:
        download_status = 'fail'
        download_message = "TimeoutError on an image...trying next one..." + " Error: " + str(e)

except HTTPError as e:
    download_status = 'fail'
    download_message = "HTTPError on an image...trying next one..." + " Error: " + str(e)

except URLError as e:
    download_status = 'fail'
    download_message = "URLError on an image...trying next one..." + " Error: " + str(e)

except ssl.CertificateError as e:
    download_status = 'fail'
    download_message = "CertificateError on an image...trying next one..." + " Error: " + str(e)

except IOError as e:
    download_status = 'fail'
    download_message = "IOError on an image...trying next one..." + " Error: " + str(e)

except IncompleteRead as e:
    download_status = 'fail'
    download_message = "IncompleteRead...trying next one..." + " Error: " + str(e)
```

Figure 6: Handling various exceptions

Meanwhile, deploying the program on remote server results in more strict requirements of the robustness of the code. For example, the program needs to handle various kinds of exceptions automatically, as shown in Figure 6. Every exception could cause break down of the running program. Adding those exceptions indeed costs a lot of time to test and configure.

```
class TimeoutError(Exception):
    pass

def handler(signum, frame):
    raise TimeoutError()
```

(a) Define TimeoutError and handler

```
# set alarm to avoid timeout
signal.signal(signal.SIGALRM, handler)
signal.alarm(30)
try:
```

(b) set signal alarm

Figure 7: Setting timeout exception

Moreover, apart from those system build-in exceptions, there are still some errors needs to be defined by myself. For instance, there is no TimeOutError in Python2 and timeout occurs frequently when downloading image (especially when calling the `read()` function). Setting a timer to restrict the download time could greatly improve the efficiency of the program and avoid deadlock. However, implementing this function could be quite tricky. After trying several solutions, I use the `signal` module to set a timer in the base of OS. As shown in Figure 7, I define a TimeoutError, inherited from the `Exception` class, and a handler function to raise the TimeoutError. In the image downloading block, set a alarm of 30 seconds using the `signal`.

2.2. Photo Cleaning

The objective of this step is to remove improper images and crop those photos with single-face. As shown in Figure 8, improper images could be group photo or the cover photo of the author's book etc. Face detection is the typical technology used in this step. Specifically, first to count faces in an image using the Python module `face_recognition`, which is based on a C++ library dlib with Python APIs. Afterwards, images with multiple faces or 0 face are removed, left only single-face images. For the single-face photo, crop the image for the convenience of further steps while keeping the original photo.

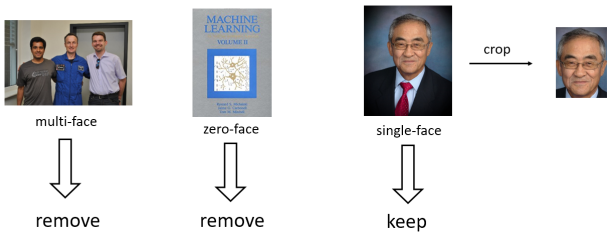


Figure 8: Illustration of photo cleaning

The implementation was done by calling the function `face_recognition.face_locations(image)`, which could list the coordinates of all faces in an image. The function returns a list, with whose length we could know the number of faces in an image.

2.3. Photo Selection

The objective of this step is to select the most representative photo from the remaining photos. The best photo apparently need first be the scholar himself. For some factors like name conflict, the search result may contain photos of different person. We need to first verify the identity of each face, i.e. whether they represent the same person. Then we should find a photo from the largest cluster, which probably the scholar himself.

The techniques used in this step is typically face recognition (distinguished from face detection, as previously mentioned). I still use the `face_recognition` module and use `face_recognition.face_encodings()` to extract features of faces and map them to 128-dimension vectors. Then I calculate the similarity between every pair

of images using the inner product of their corresponding vectors

$$\text{sim}_{ij} = V_i \cdot V_j \tag{1}$$

where V_i and V_j are corresponding mapped vector of two image I_i and I_j. Afterwards, for each photo, calculate a score based on its similarities between other images as

$$s_i = \sum_{j=1}^{N} \text{sim}_{ij} \cdot \text{I}(\text{sim}_{ij} > \tau) \tag{2}$$

where N is the number of valid photos (left after photo cleaning) and τ is a threshold that distinguish whether two faces represent the same person or not, typically set 0.6. $\text{I}(\cdot)$ is the indicator function, which is 1 when the corresponding expression is true and 0 otherwise.

Finally I choose the photo with the highest score as the best photo. The idea behind this design is quite intuitive since the metric prefers the centering photo among the largest cluster.

3. Experimental Results

I implement and run my program under the environment of Ubuntu 16.04.3 LTS and Python 2.7.12. I have downloaded photos of more than 6,000 scholars, up to 100,000 photos, 30+ GB. Data are formatted in id-photo mapping by folder name, as shown in Figure 9. Generally, the results seems quite satisfying, however, further quantitative evaluation is needed. We could consider the photo on the home page of the scholar to be ground truth. I compare my results with photos crawled from the home page of scholars (if exists) and obtain an accuracy of 96%.

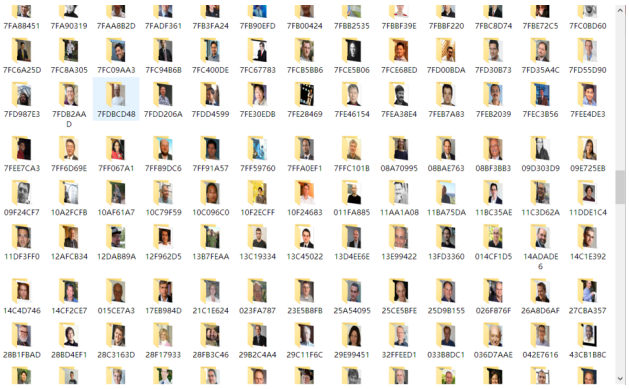
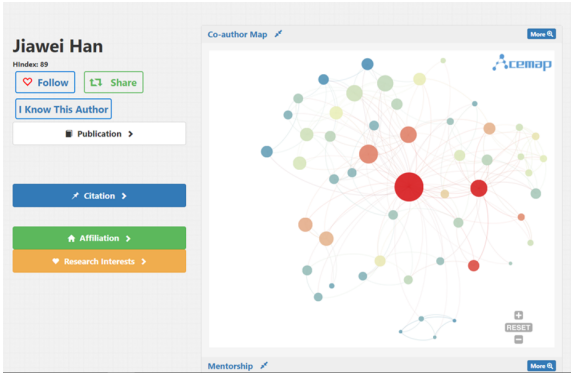
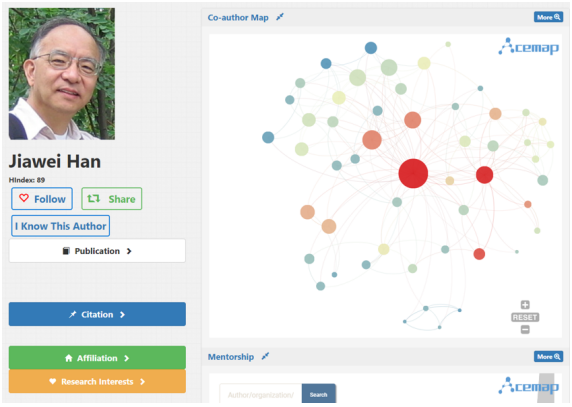


Figure 9: Results

Also, I have submitted more than 1,000 photos to the AceMap system and brought online (http://acemap.sjtu.edu.cn/), as shown in Figure 10.



(a) Before



(b) After

Figure 10: Online demo (http://acemap.sjtu.edu.cn/)

Using score to select photo has some advantage in efficiency than clustering algorithms. Typical clustering algorithms like Chinese Whisper [12] needs first calculate the similarity of every pair, then have several iterations to cluster. Therefore, it is much slower than simply picking best photo by score.

4. Conclusion

In this project, I design and implement all the program by myself. Considering the large scale of data, I tried several ways to improve the efficiency of the code. Considering the lack of dataset and ground-truth labels, I use unsupervised method to select the best photo. For name conflict problem, I search by name combined with corresponding affiliations and greatly reduce the conflict problem. With selection by similarities, I achieve sounding accuracy as well as a fast speed. For experiment, I downloaded more than 100, 000 images and selected photos for more than 6,000 scholars. I also compared my results with photos crawled from the home page of scholars and achieve an accuracy higher than 95%.

References

[1] Zhaowei Tan, Changfeng Liu, Yuning Mao, Yunqi Guo, Jiaming Shen, and Xinbing Wang. Acemap: A novel approach towards displaying relationship among academic literatures. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 437–442, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[2] S-K Chang and Arding Hsu. Image information systems: where do we go from here? *IEEE transactions on Knowledge and Data Engineering*, 4(5):431–442, 1992.

[3] Yong Rui, Thomas S Huang, and Shih-Fu Chang. Image retrieval: Current techniques, promising directions, and open issues. *Journal of visual communication and image representation*, 10(1):39–62, 1999.

[4] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on pattern analysis and machine intelligence*, 22(12):1349–1380, 2000.

[5] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[6] Chaochao Lu and Xiaoou Tang. Surpassing human-level face verification performance on lfw with gaussianface. In *AAAI*, pages 3811–3819, 2015.

[7] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems*, pages 1988–1996, 2014.

[8] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[9] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[10] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.

[11] Er Gurpreet Kaur, Er Harjot Kaur, and Er Manpreet Kaur. A survey on face recognition techniques. *International Journal of Advanced Research in Computer Science*, 8(4), 2017.

[12] Chris Biemann. Chinese whispers: an efficient graph clustering algorithm and its application to natural language processing problems. In *Proceedings of the first workshop on graph based methods for natural language processing*, pages 73–80. Association for Computational Linguistics, 2006.