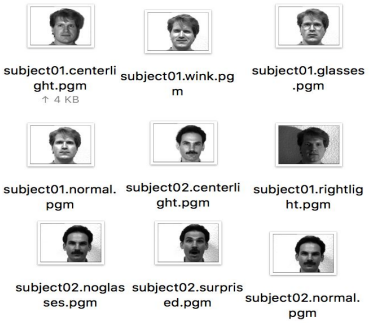
Criminal Facial Detection System

SuoAn Gao, Ye Zhuo

Abstract: With our designed criminal facial detection system, given any **forward-looking** face pictures, we can find whether this picture belongs to the criminal clustering or not. This will help the police department to find a criminal hiding in the city. Wherever places with a camera installed this system can run detection works. Then it could send a message to polices or raise an alarm after finding the criminal.

Description of Database: We used Yale Face Data set for this specific project. This dataset contains 15 individuals of each 11 images.



System Specification: The core function to distinguish human faces is built upon Eigenfaces.[1][2][3][4] It's a specifically modified version of PCA unsupervised learning.[1] However, the limitation is that the face images must be centered and of the same size. Thus, before the Eigenface algorithm, we need to grab only human's face and make the face centered right in the middle[2]. And we need to make sure they all have same size.[1] To do so, we need to introduce three Python libraries in the following section, they are Numpy, OpenCV, and face_recognition.

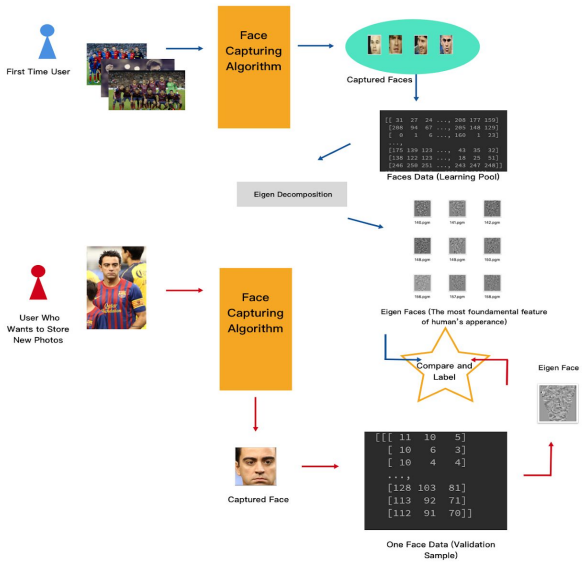


Figure 0. System Configuration

Technical Software Related: This project is built upon Python v.3.6.2 with 3 most important libraries: Numpy, Face_recognition, and OpenCV. Where Numpy is the most widely used linear algebra computational library, In our project it helps us to build all of our algorithms. OpenCV, the most widely used python library in computer vision and image processing. And the last, but not least, is a Git-hub open source library called face_recognition. It takes several steps to set up. It basically fetches out all of the faces in a specific image and slices out the specific image matrix where contains a human face.

Clone the code from github:

```
git clone https://github.com/davisking/dlib.git
```

Build the main dlib library (optional if you just want to use Python):

```
cd dlib
mkdir build; cd build; cmake .. -DDLIB_USE_CUDA=0 -DUSE_AVX_INSTRUCTIONS=1; cmake --build .
```

Build and install the Python extensions:

```
cd ..
python3 setup.py install --yes USE_AVX_INSTRUCTIONS --no DLIB_USE_CUDA
```

Figure 1. How to setup environment before install face_recognition library

There are two specific methods in this face capture library, we used the HoG-based model instead of CNN since it's faster and the accuracy just slightly lower.

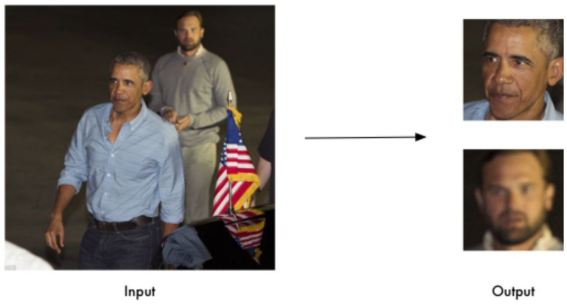


Figure 2. An example of how we use it to capture faces

EigenFace Algorithm: After we put all square image matrices(use opencv.imread to convert from M images) into linear matrices and store all of the linear matrices into a large matrix, we called it 'learning pool', we

need to modify the learning pool before we proceed further[1].

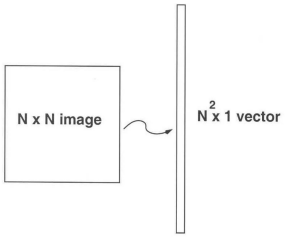


Figure 3. Compress image to linear vector

We are going to find average faces from our dataset.

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i \qquad \text{(equation. 1)}$$

Here Ψ is the average face in our data set. It looks like:

Figure 4. Average face in the dataset

Then, we subtract the average face from our learning pool(****notice*** the learning pool is a matrix, data set is a collection of images) and have the average-excluded matrix. We denote this matrix as A.

where $A = [\Phi_1\ \Phi_2 \cdots \Phi_M]$ $(N^2 \text{x} M \text{ matrix})$

Next step, in a regular PCA algorithm, will need to find the covariance matrix of A,

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T \quad (N^2\text{x}N^2 \text{ matrix})$$

However, it's not practical while dealing with images since the dimension of the matrix is too big. Not even need to mention how many computational resources are needed for doing Eigen decomposition for such a large matrix.

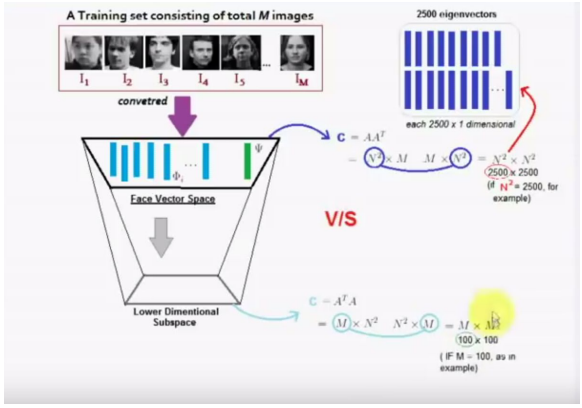


Figure 5. Comparison of finding covariance matrix and finding dimensional reduced covariance

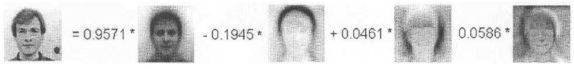
Thus, we instead of finding AA^T, $(N^2\text{x}N^2 \text{ matrix})$ and apply eigen decomposition at such large matrix, we could first compute $A^T A$, a M x M matrix, and find its K-largest eigenvectors, v_i, a M x K matrix, then map it back to a K x

N^2 matrix, which is the approximate eigenvector of or original learning pool.

These eigenvectors are called eigen faces.

Figure 6. Some eigenfaces we generated

We know that each every image could be approximately represented and separated out from each other by adding the average face and weighted eigenfaces together[1]. For example, in this graph below, the person on the left



could be represented by the combination of these weighted faces.

Face recognition: We decompose all the image in the dataset to get wight matrix according to the decompose of faces. When a new image coming in, we follow the same process as before by first remove the average face, and then decompose it by using the eigenfaces we founded before. In other words, it will find a combination of these eigenfaces, and we got a weight for this face. By comparing this new weight and the weight matrix, we can find which weight in the weight matrix has the minimum Euclidean distance to the new weight. Then these two weights have high probability coming from the same person.

Applications: With our criminal facial detection system, given any picture, we can find whether this picture belongs to the criminal clustering or not. This will help policies to find a criminal hiding in the city.

Using the same process, we can get the special criminal face's weight according to eigenface subspace. With real-time images from cameras in the cities, we can crop every face in every image and compare weight from crop image between the special criminal face's. If the weight error falls into a specified range which can be recognized as correct, then the person is a suspect in criminal. Wherever places with a camera installed this system can run detection works, then send a message to polices or raise an alarm after finding the criminal who

is absconding. Not only for criminal facial detection, but also for finding lost old man or pets. Using a camera to help will save a lot of time and money, and would be an efficient and effective way to find the person or pets you need.

Conclusion: In conclusion, in this project, we successfully build a criminal facial detection system which can recognize people's face from a picture, crop it, and compare it with the dataset to find this picture belong to criminal or not. Our algorithm is using a variant of PCA unsupervised learning. The advantage of our detection system is it can fast do a comparison and identify the special people from a real-time image. However, the limitation is that the face images must be looking forward and centered. But it is doable because the real-time camera can alway get a centered image if the suspect person enters the camera range and we tune the camera angle.

Reference:

[1]M. Turk and A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991, hard copy.

[2]Wang, Hongjun, Jiani Hu, and Weihong Deng. "Face Feature Extraction: A Complete Review." IEEE Access 6 (2018): 6001-6039.

[3]Paul, Liton Chandra, and Abdulla Al Sumam. "Face recognition using principal component analysis method." International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) 1.9 (2012): pp-135.

[4]Wright, John, et al. "Robust face recognition via sparse representation." IEEE transactions on pattern analysis and machine intelligence 31.2 (2009): 210-227.