

# Static and Real Time Face Recognition Base on PCA and SVM

Yan MA

December 16, 2017

Instructor: Prof.Adam Krzyak

Student ID: 40024856

## **Abstract**

In this report, I start from static face recognition and explain how to construct classifier base on PCA and SVM. Using this method, I get 83.5% correct recognition rate. After that, I continuing for the 'Real-Time' face recognition, I use the same classifier along with face detection and face tracking algorithm. Using this method, I get correct recognition rate is 90%. Finally, I have a discussion for the robustness of the classifier.

From section 2 to section 5, I give a detail explanation for static face recognition. Section 2 is an introduction about the ORL face database. Section 3 discusses feature extractor, PCA, including how it works, how to accelerate PCA and how to select reasonable parameters in PCA. Constructing classifier is discussed in section 4, including how to select kernel function, parameters selecting and multi-classifier constructing for support vector machine.

Section 6 is for real time face recognition. Basing on the works above, I establish my own database and use Matlab toolbox for face detection and object tracking to obtain my 'test data', then do the same as previous works, feed the 'test data' into trained classifier and get the result.

Section 7, I discuss the robustness of the classifier and give an example that when I mix my own database and ORL face database together, my face recognition rate down to 75%. Therefore, I get a conclusion that this system is unstable under changes of the enviroment.

This report is base on [Faruqe and Hasan \(2009\)](#) and [Guo et al. \(2000\)](#).

**Keywords:** PCA, SVM, Static and Real-Time Face Recognition

## Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Cambridge ORL Face Database</b>	<b>6</b>
<b>3</b>	<b>Feature Extraction</b>	<b>7</b>
3.1	Principal Component Analysis (PCA) . . . . .	7
3.2	Fast PCA . . . . .	8
3.3	Face Feature Extraction . . . . .	8
3.4	Construct Eigenfaces . . . . .	9
<b>4</b>	<b>Construct Classifier</b>	<b>11</b>
4.1	Data Preprocessing . . . . .	11
4.2	Which Kernel Function to Use? . . . . .	12
4.3	How to decide the parameters? . . . . .	13
4.4	How to construct a multi-classifier? . . . . .	15
<b>5</b>	<b>Experimental Result and Code Running Demo</b>	<b>16</b>
<b>6</b>	<b>Real-time Face Recognition</b>	<b>17</b>
6.1	Establish Face Database . . . . .	18
6.2	Face Detection . . . . .	19
6.3	Face Tracking . . . . .	19
6.4	Face Recognition . . . . .	20
6.4.1	Feature Extraction . . . . .	20
6.4.2	Parameter Selection for Classifier . . . . .	21
6.5	Experimental Result and Code Running Demo . . . . .	22
<b>7</b>	<b>Testing and Robustness Analysis</b>	<b>26</b>
7.1	Testing Example 1: Glasses v.s Without Glasses . . . . .	27
7.2	Testing Example 2: Another Face Testing . . . . .	28

7.3	Testing Example 3: Hat v.s Without Hat . . . . .	29
7.4	Testing Example 4: Combine ORL face database and My Face Database . . . . .	29
<b>8</b>	<b>Conclusion and Future Work</b>	<b>31</b>
<b>9</b>	<b>Appendix</b>	<b>33</b>

## List of Figures

1	Explained Variance Ratio For ORL Face Database . . . . .	9
2	20 Eigenfaces . . . . .	10
3	Face Recognition Rates with Increasing Dimension of Features for Linear, Polynomial and RBF-SVM . . . . .	13
4	Cross Validation for Choosing Parameters $C$ and $\gamma$ . . . . .	14
5	Bottom Up Tree For Constructing Multi-Classifer . . . . .	15
6	Adding Code folder into Workspace . . . . .	16
7	Experimental Result base on SVM and PCA Classifier . . . . .	17
8	My Database Example . . . . .	18
9	Cascade Classifier from Matlab Official Reference . . . . .	19
10	Explained Variance Ratio For My Friends Face Database . . . . .	21
11	Cross Validation For My Friends Face Database . . . . .	22
12	Adding File Path . . . . .	22
13	Adding your Face into Database . . . . .	23
14	Adding your Face into Database 2 . . . . .	24
15	Training the Database and Get the Eigenfaces . . . . .	25
16	Result . . . . .	26
17	Testing Example 1 . . . . .	27
18	Testing Example 2 . . . . .	28
19	Testing Example 3 . . . . .	29
20	Using Real Time Screen Cut Face Tests Classifier Bases on Mixed Face Database . . . . .	30
21	Using ORL Faces Tests Classifier Bases on Mixed Face Database .	31
22	PCA Gives a Bad Projection Example . . . . .	32
23	Format for LibSVM . . . . .	33
24	Grid.py Path Configuration . . . . .	33
25	Running Cross Validation . . . . .	34
26	Cross Validation Result . . . . .	34

## 1 Introduction

Face recognition widely used in public security, security authentication system, credit card verification, file management and interactive system, etc. And it is a current hot research topic in the field of pattern recognition and artificial intelligence. This technique is based on computer vision and recognize faces from static or dynamic images. A general statement of the face recognition problem (in computer vision) can be formulated as follows: Given still or video images of a scene, identify or verify one or more persons in the scene using a stored database of faces. In this project, I use Principle Component Analysis(PCA) and Support Vector Machine(SVM) to implement static and real time face recognition.

## 2 Cambridge ORL Face Database

I use Cambridge ORL(Our Database of Faces) as face database. First, a brief introduction to ORL:

- a. ORL Database has 400 images of people's face. 40 people and 10 images for each, the size of the images are 112 pixels height and 92 pixels width.
- b. The lighting direction and the background for each person are similar.
- c. There are a few changes of expression and posture between images for a person.
- d. Not all of the people have images of these changes, some people have more posture changes, some people have more facial expressions, some of them wear glasses. But these changes are modest.

we can download the database on

<http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>

### 3 Feature Extraction

Feature extraction refers to find new feature subset based on the original feature sets. Linear transformation is a popular method because of its simple using and analysis. Principal Component Analysis (PCA) is one of linear transformation. The main idea is to represent the sample data in high dimensional space on low dimensional space by linear transformation, which represents the original data as well as possible.

#### 3.1 Principal Component Analysis (PCA)

Let's have a brief review for PCA dimension decreasing method. The general statement is, suppose that we have  $n$  samples  $x_1, x_2, \dots, x_n$  in  $d$  dimension space. How to represent them well on  $k$  dimension space using PCA, where  $k \ll d$ . Here is the general steps:

- a. Calculate the Scatter Matrix  $S$ , where  $S = \sum_{i=1}^n (x_i - m)(x_i - m)^T$  and  $m$  is the average value of  $n$  samples.
- b. Select lower dimension  $k$ .
- c. Find the  $k$  biggest eigenvalues of scatter matrix  $S$ .
- d. Find the eigenvectors base on the  $k$  eigenvalues.
- e. Calculate the samples' projection in the  $k$  dimension space

This is a very simple method. However, consider the scatter matrix  $S$  which is a  $d \times d$  squared matrix. If samples have very high dimensions which means they have very high values of  $d$ . If we use the steps above, the calculation part will be very costly. But fortunately we are not helpless. There is a very good 'PCA Acceleration Technique' that can be used to calculate eigenvalues and eigenvectors of scatter matrix  $S$ . I denote this method as fast PCA.

### 3.2 Fast PCA

The fast PCA is based on the following theoretical basis:

Suppose that  $Z_{n \times d} = (x_i - m)$ , therefore, the scatter matrix  $S = (Z^T Z)_{d \times d}$ , but instead calculate scatter matrix  $S = (Z^T Z)_{d \times d}$ , we consider matrix  $R = (Z Z^T)_{n \times n}$ . Generally  $n \ll d$ .

Suppose that  $n$  dimensional column vector  $v$  is the eigenvector of matrix  $R$ , then we have:

$$(Z Z^T) v = \lambda v$$

. On both side we do left-handed multiply  $Z^T$  and we get:

$$Z^T (Z Z^T) v = Z^T \lambda v \Rightarrow (Z^T Z) (Z^T v) = \lambda Z^T v$$

we find that  $(Z^T v)$  is the eigenvalue of the scatter matrix  $S$ . According to the mathematical deduction above, we find that we can calculate the eigenvalue of a small matrix  $R = (Z Z^T)_{n \times n}$ ,  $v$  and then do a left-handed multiply  $Z^T$ , where  $(Z^T v)$  is the eigenvector of  $S$ . For Matlab implementation, please check file 'fast-PCA.m'.

### 3.3 Face Feature Extraction

For feature extraction from the ORL face database, firstly, I have to construct our sample matrix, function "ReadFaces()" help us read the face image. For each image, I put it in a row in the sample matrix. Therefore, for a  $112 \times 92$  matrix, we store it in the sample matrix as a 10304 dimension row. Then, we store the sample matrix as "FaceMat.mat" file under "/Mat" in Matlab workspace.

Secondly, using PCA to decrease the dimension, but how to select  $k$ ? I use "Ex-



plained Variance Ratio”. This is defined as following function.

$$EVR = \frac{\lambda_1 + \lambda_2 + \dots + \lambda_k}{\lambda_1 + \lambda_2 + \dots + \lambda_k + \lambda_{k+1} + \dots + \lambda_n}$$

where  $\lambda_i$  is the  $i$ th biggest eigenvalue in the scatter matrix. In this project, I get the  $EVR$  values with different  $k$  shows in Figure1. We find that  $k = 20$  is a good choice as  $\lambda_i$ , where  $i > 20$  is not too large and would not influence the  $EVR$  value too much. In addition, when  $k = 20$ , the first 20  $\lambda$ 's can represent almost 75% of the overall features. Therefore, we choose  $k = 20$ .

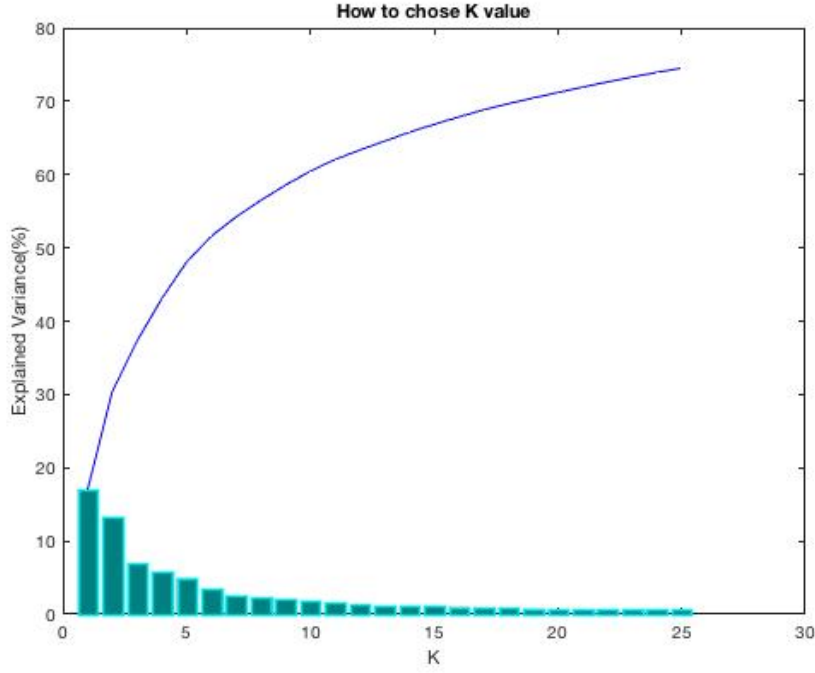


Figure 1: Explained Variance Ratio For ORL Face Database

### 3.4 Construct Eigenfaces

Base on the fastPCA function above we can decrease images dimension from 10304 to 20. The eigenvector matrix  $V$  is one of the output of "fastPCA" func-

tion, which is  $10304 \times 20$  matrix. For each column is the eigenvector (10304 dimension) of scatter matrix  $S$ . In face recognition, we consider them as eigenfaces. I put each column in a  $112 \times 92$  matrix, same as the original image, and I get 20 eigenfaces as showed in Figure2.

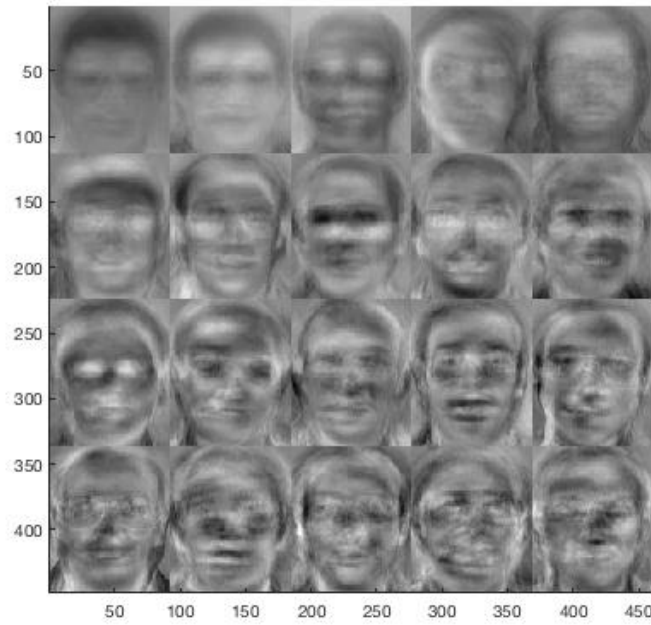


Figure 2: 20 Eigenfaces

Eigenfaces analysis: We can consider each eigenfaces as the basis vector in 20 dimension space, and each person has a different position in this space. But why the position is different? The answer is that they are different in 20 principle component.

For example, the background of each eigenfaces don't have too much different, which means that they are not important in the face recognition. Another example, the 4th image in the first row, left side of the face pixel is different compared with the right side, therefore, this eigenface represent the lighting influence probably.

For another example, on the 3rd row, the 1st eigenface, we find that the eye socket part is highlight compared with its skin, therefore, it may represents the depth of the eye sockets.

## 4 Construct Classifier

In this section, I am going to construct the classifier base on Support Vector Machine (SVM). As we all know that SVM is a binary classifier, and we have to choose a good kernel function to solve non-linear classification problem. Therefore, we can construct our classifier by solving the following questions step by step.

- a. Data preprocessing
- b. Which kernel function is good to use?
- c. How to decide the parameters?
- d. How to construct a multi-classifier?

### 4.1 Data Preprocessing

Before I go into classifier construction part, I need to do some data pre-processions. The steps are shown as follows:

First step. Splitting data into training data (5 pictures for each person) and test data (5 pictures for each). Therefore, we have 200 samples for training and 200 for testing.

Second step. Read the pictures into sample matrix, for each image we save it as a  $1 \times 10304$  row . Therefore the size of the sample matrix is  $400 \times 10304$ , I save it into a matrix called "FaceContainer", for more detail, please check file function "readFaces.m".

Third step. Using PCA decrease the dimension. In this project, I select the new dimension  $k = 20$ , and get 20 eigenfaces, which I have a detail discussion in section 3.3 and section 3.4.

Fourth step. Scaling the data, here are 2 main reasons why have to scaling the data.

- 1 To avoid the features which have a big numeric ranges overwhelming features have small numeric ranges. We have a disccusion in assignment1.
- 2 For SVM, we usually need to select kernel, and in most cases we need calculate inner products, if we have a big numeric range features and do inner product for them, it is very costly and may exceed memory.

In this report, we use "max&min scaling" method. Suppose that  $min_A$  and  $max_A$  is feature  $A$ 's min value and max value respectively.  $v \in A$ . The mapping defined as follows:

$$v' = \left[ \frac{v - min_A}{max_A - min_A} (new\_max_A - new\_min_A) \right] + new\_min_A$$

$v' \in [new\_min_A, new\_max_A]$ , for Matlab implementation part, please check file function "scaling.m" for more detail.

Now, we finish the data preprocessing part, then I am going to construct our SVM based classifier.

## 4.2 Which Kernel Function to Use?

SVM classifier is linear classifier, I add Kernel Function for non-linear classification problem. In this project, I select "Radial Basis Function (RBF)":

$$K(x, y) = \exp\left(-\gamma\|x - y\|^2\right)$$

The reasons are as follows:

- 1 As a Kernel Function, RBF is able to deal non-linear classification problem.
- 2 For Polynomial-SVM, we have to consider memory overflow because of dot production.
- 3 For Sigmoid-SVM, it performs the similar way compared with RBF-SVM, but we have to decide 2 parameters' value.
- 4 Linear-SVM is a particular case of RBF-SVM
- 5 RBF-SVM performs better than Linear-SVM and Polynomial-SVM, as shown in Figure3

Figure 3 is from [Faruque and Hasan \(2009\)](#)

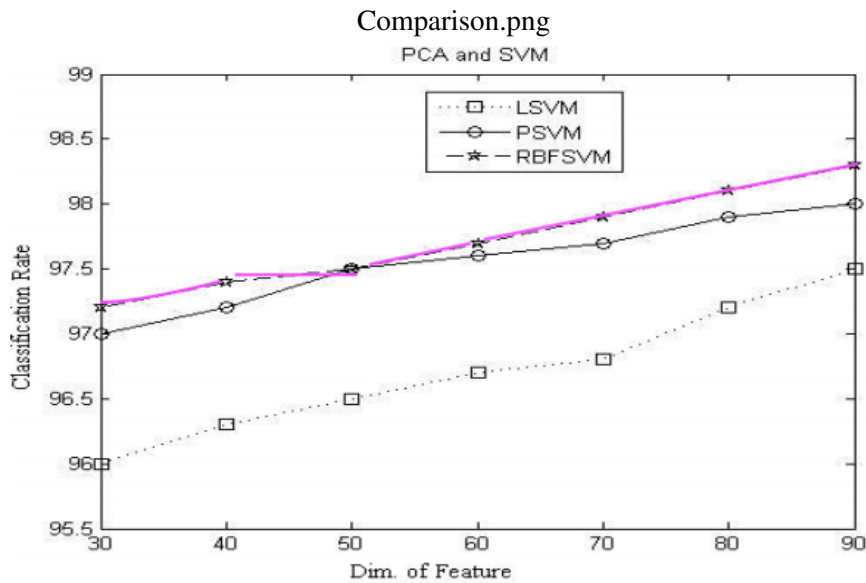


Figure 3: Face Recognition Rates with Increasing Dimension of Features for Linear, Polynomial and RBF-SVM

#### 4.3 How to decide the parameters?

I choose RBF as kernel function, therefore I have 2 parameters to decide.  $\gamma$  in kernel function and  $C$  in "Soft Margin SVM". But how to choose better parameters

to get a good recognition rate?

This problem is a optimization problem, the variables are  $C$  and  $\gamma$ , and the objective function is the recognition rate for testing data. However, it is really hard to find function between objective function and the variables. Therefore, I don't use optimization algorithm to solve this question.

Luckily, the SVM library LibSVM provides a very practical parameter selection method based on cross validation and grid searching. However, the searching toolbox is a script file of Python and have a specified format. Therefore, make sure Python is installed. For result visualization, gnuplot.exe is needed, but it is easy to find on the internet. For more details of configuration and using, please check appendix.

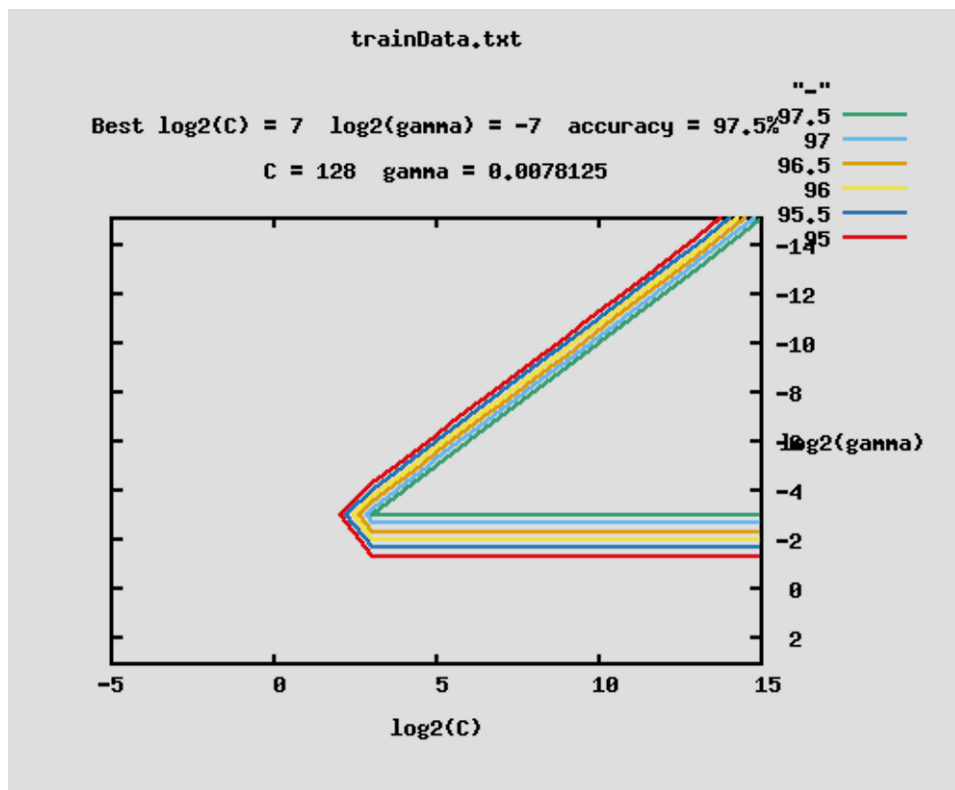


Figure 4: Cross Validation for Choosing Parameters  $C$  and  $\gamma$

According to the Figure4, when  $C = 128$  and  $\gamma = 0.0078$ , the correct recognition rate is 97.5%.

#### 4.4 How to construct a multi-classifier?

Bottom-up Decision Tree, this scheme was proposed in [Guo et al. \(2000\)](#) and [Faruqe and Hasan \(2009\)](#).

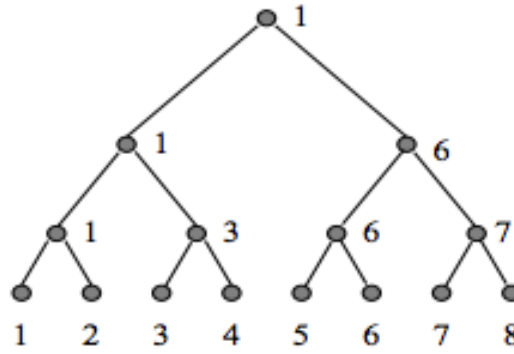


Figure 5: Bottom Up Tree For Constructing Multi-Classfier

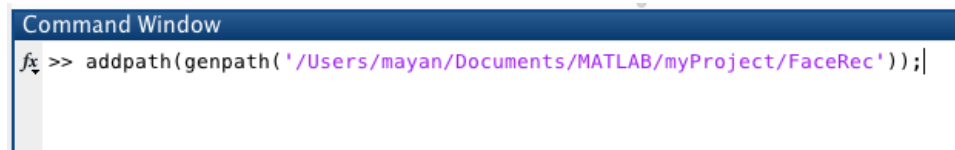
Suppose there are eight classes in the data set, the decision tree is shown in Fig. 5, where the numbers 1-8 encode the classes. Note that the numbers encoding the classes are arbitrary without any means of ordering. By comparison between each pair, one class number is chosen representing the winner of the current two classes. The selected classes (from the lowest level of the binary tree) will come to the upper level for another round of tests. Finally, the unique class will appear on the top of the tree.

Denote the number of classes as  $c$ , the SVMs learn  $C_n^2$  discrimination functions in the training stage, and carry out comparisons of  $c - 1$  times. Therefore, the training time complexity is  $O(n^2)$  and the comparison time complexity is  $O(n)$ . For more detail, please check file function 'multiSVMClassify.m' and 'multiSVMTrain.m'

under 'SVM' folder in the workspace.

## 5 Experimental Result and Code Running Demo

First, add the code folder into the Matlab workspace, here my workspace is '/Users/mayan/Documents/MATLAB/myProject/FaceRec'. As shown in Figure 6.

A screenshot of the MATLAB Command Window. The title bar is blue and says "Command Window". The command prompt is "fx >>". The command entered is "addpath(genpath('/Users/mayan/Documents/MATLAB/myProject/FaceRec'));" in a purple font. The cursor is at the end of the command.

```
fx >> addpath(genpath('/Users/mayan/Documents/MATLAB/myProject/FaceRec'));
```

Figure 6: Adding Code folder into Workspace

Second, we choose parameters  $C = 128$  and  $\gamma = 0.0078$  to train the classifier, and we got correct rate is 83.5%. Then, I choose person number 1 as test sample and feed it in the classifier, we got a correct answer.



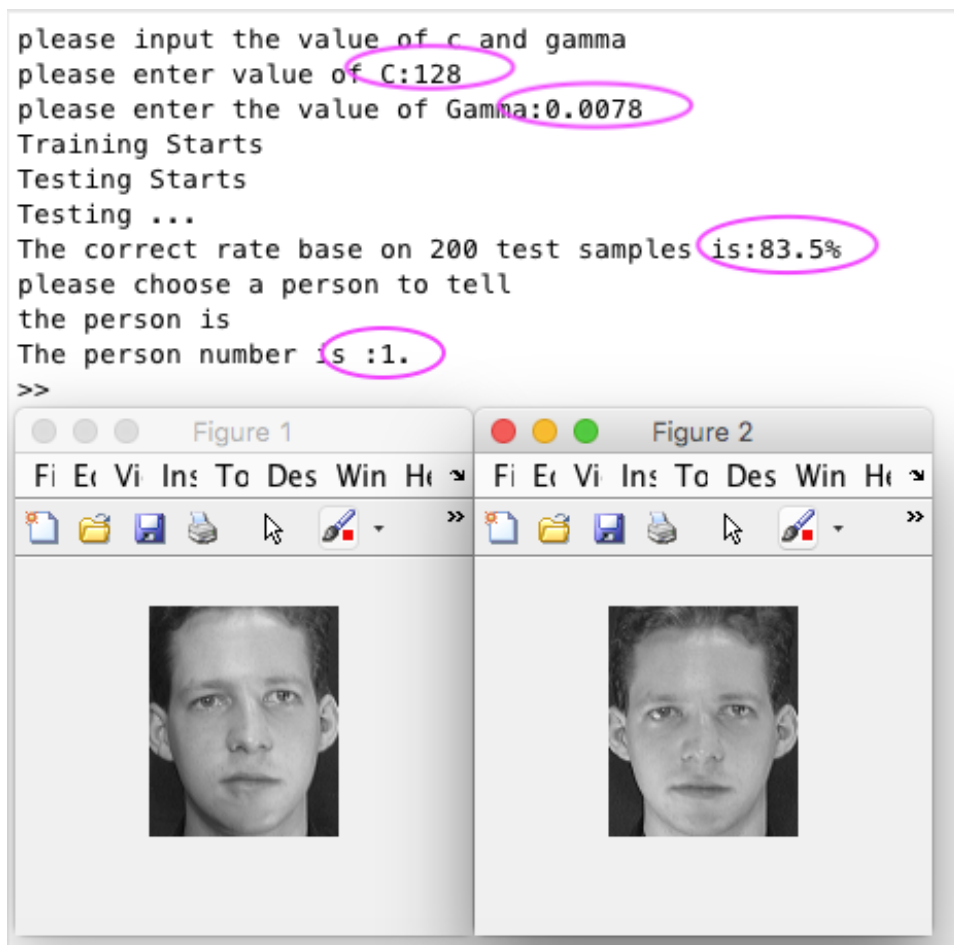


Figure 7: Experimental Result base on SVM and PCA Classifier

## 6 Real-time Face Recognition

In this section, I continued to the implementation of the Real-Time Face Recognition. In the previous sections, I give the detail of the static face recognition. However, real-time face recognition is more complexity than static face recognition and beyond the reference paper. Therefore, I won't go into very detail of the algorithm for face detection and face tracking, I will use Matlab toolbox for both algorithm.

## 6.1 Establish Face Database

First, I establish the face database of some of my friends, 7 people in total and 10 pictures for each. The size of each image is same with the ORL face database, 92 width and 112 height (pixels). I took the picture in Concordia EV Building lab. The background is a whiteboard.



Figure 8: My Database Example

Some Explanation:

- a. Camera: I used 'macvideo', the video format is 'YCbCr422\_1280x1024'.
- b. Face Detection: I use Matlab toolbox function 'vision.CascadeObjectDetector();'.
- c. Face Tracking: I use toolbox function 'vision.PointTracker()'.
- d. ScreenCut: Screencutting for the detected area, which size is  $112 \times 92$ . Please check 'cutPic.m'.

For more detail, please check function file 'training.m', the output will be a image file generate under 'Data/ORL/' in workspace, and the format of the image is '.pgm' which is same with the ORL database.

## 6.2 Face Detection

For this final project, I use 'vision.CascadeObjectDetector' System object in vision toolbox of Matlab 2016a to detect peoples face using Viola-Jones algorithm [Viola and Jones \(2001\)](#).

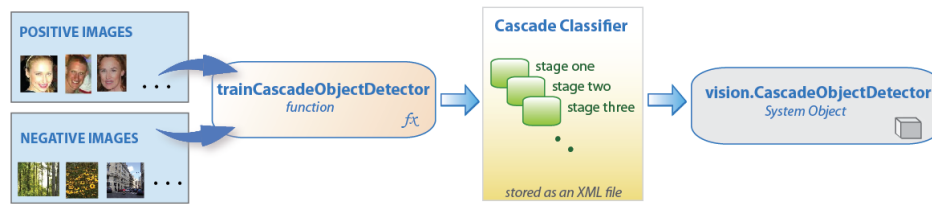


Figure 9: Cascade Classifier from Matlab Official Reference

Viola-Jones algorithm, to put it simply, is an examining box for a sliding window in an image to try to match different dark/light regions so it can identify a face. The size of the window varies on different scales for different faces, however, the ratio of the window remains unchanged.

Matlab used cascade classifier (as shown in figure 9) to quickly determine the regions at which human faces are detected. Each of the cascade classifier consists of stages, and each stage contains a decision stump. According to Matlab, they designed those stages to quickly rule out the regions that does not have a face in earlier stages(namely, reject negative samples), and thus save time for analyzing potential region with face in deeper stages.

## 6.3 Face Tracking

I am using Kanade-Lucas-Tomasi (KLT) algorithm [Mstafa and Elleithy \(2016\)](#) to do face tracking. In the matlab video processing toolbox, it provide a function 'vision.PointTracker' to tracking points using KLT algorithm. This algorithm is

basically based on feature point tracking on the first face, and keeps on tracking it until there is no feature point available.

For the first feature points set on tracking, we used eigenvalue algorithm to find corner points. Basically this is ShiTomasi corner detection algorithm [Shi et al. \(1994\)](#) which detects the corner. It directly computes the value of eigenvalues to determine whether it is a point of interest or not.

After detecting those points, we will be able to track each of the points we found from Shi-Tomasi corner detection. For each consecutive frame we will try to match the points from the step above. There might be points missing, and if it is the case we will rule them out. As long as there are at least 2 points exist in the videoframe, we will be able to continue tracking the face by finding out the affine transformation of those points.

For more details, please check Matlab official reference:

<https://www.mathworks.com/help/vision/ref/vision.pointtracker-system-object.html>

## **6.4 Face Recognition**

Up to now, I have already established my own face database. And the main idea of real time face recognition is detecting and tracking face, screencutting the picture, then put the picture into the trained classifier, which is based on PCA+SVM algorithm.

### **6.4.1 Feature Extraction**

I use the same strategy to find the eigenfaces, PCA. Therefore, I have to determine the  $k$  which is the dimension we want to decrease to. As section 3.3, we use 'Explained Variance Ratio' to decide  $k$ , the result shown as Figure 10

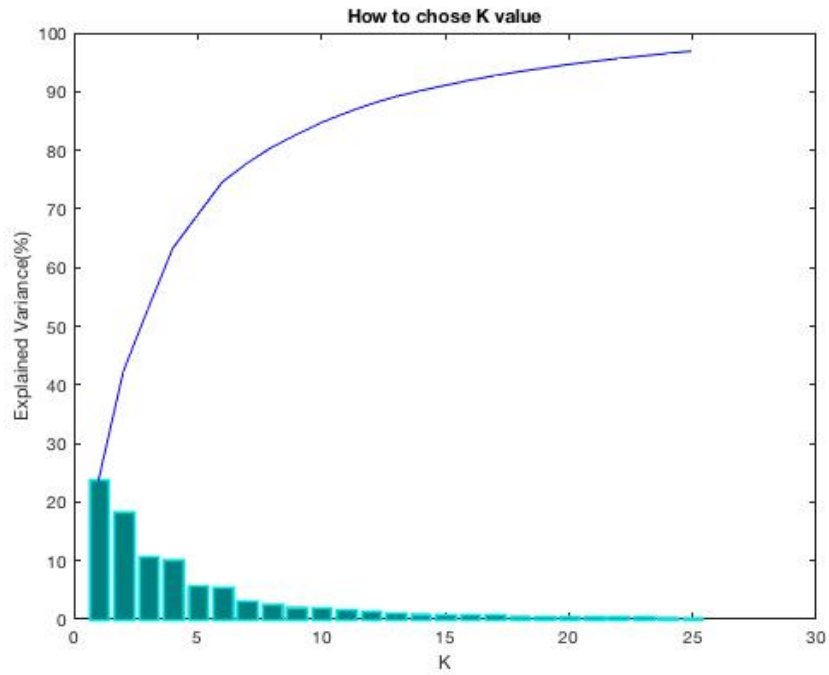


Figure 10: Explained Variance Ratio For My Friends Face Database

According to Figure 10, I still choose  $k = 20$ , where 20 eigenvalues can represent 95% of a image.

#### 6.4.2 Parameter Selection for Classifier

As I did in section 4.3, I use LibSVM library to do cross-validation for us. The result is shown in Figure 11,

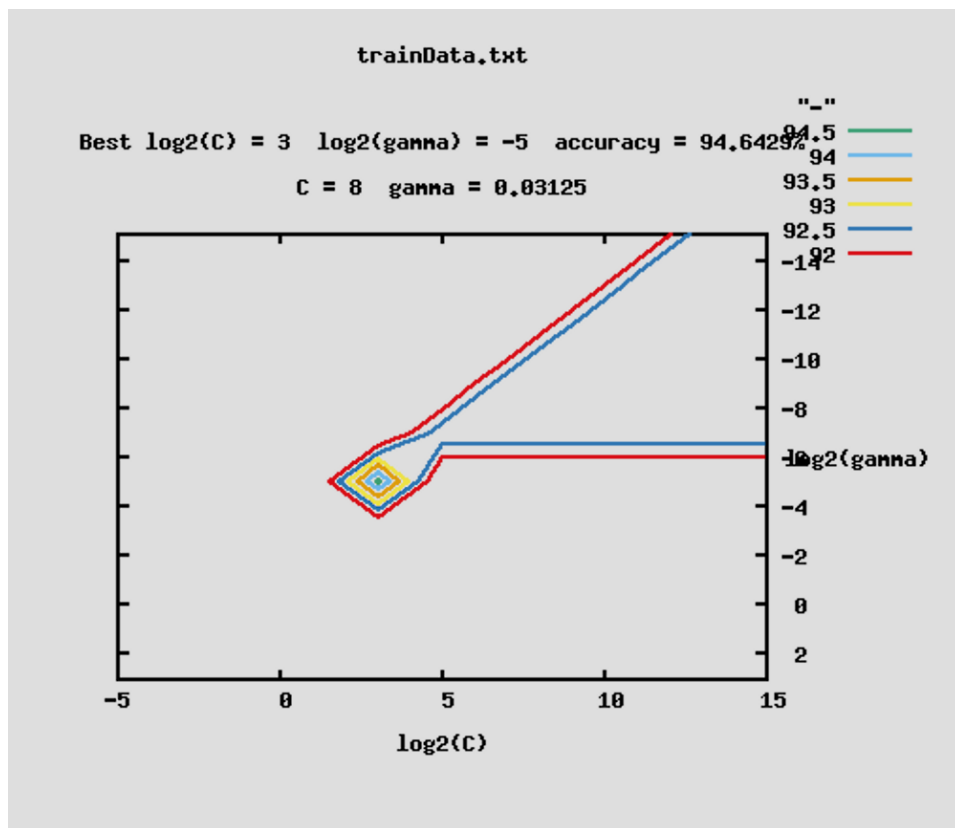


Figure 11: Cross Validation For My Friends Face Database

As shown in Figure 11, we choose  $C = 8$ ,  $\gamma = 0.0315$  and we got the recognition accuracy for training data is 94.5%

## 6.5 Experimental Result and Code Running Demo

First, add path into the work space:

```
Command Window
fx >> addpath(genpath('/Users/mayan/Documents/MATLAB/A1/FaceRec'));
```

Figure 12: Adding File Path

Second, putting your face into data base, we use the function 'training.m'.

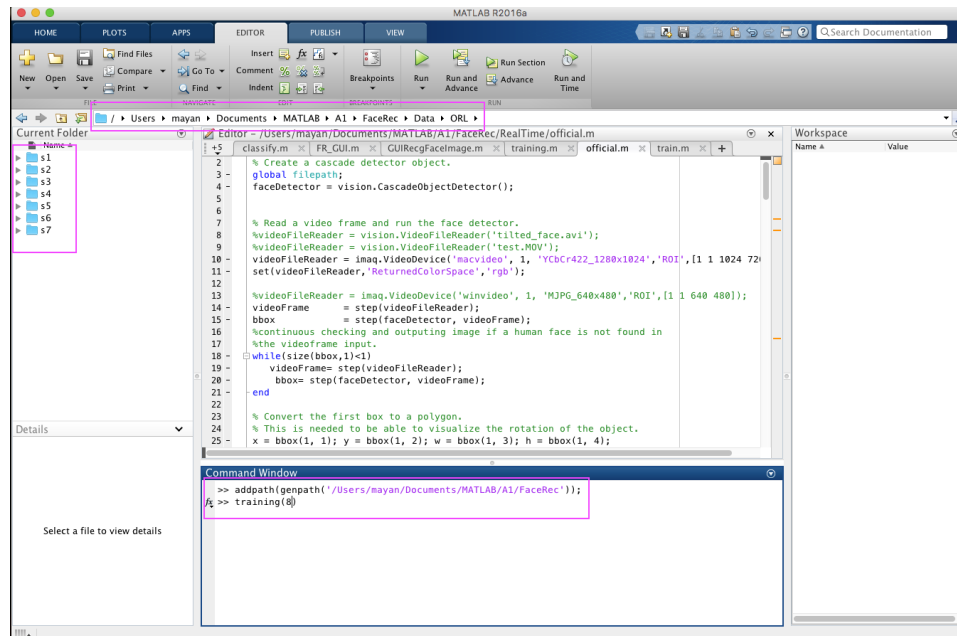


Figure 13: Adding your Face into Database

As you can see in the highlight area. I have 7 face database under '/Users/mayan/Documents/MATLAB/A1/FaceRec/Data/URL'. Therefore, at the command area we type 'training(8)' to add the 8th face database.

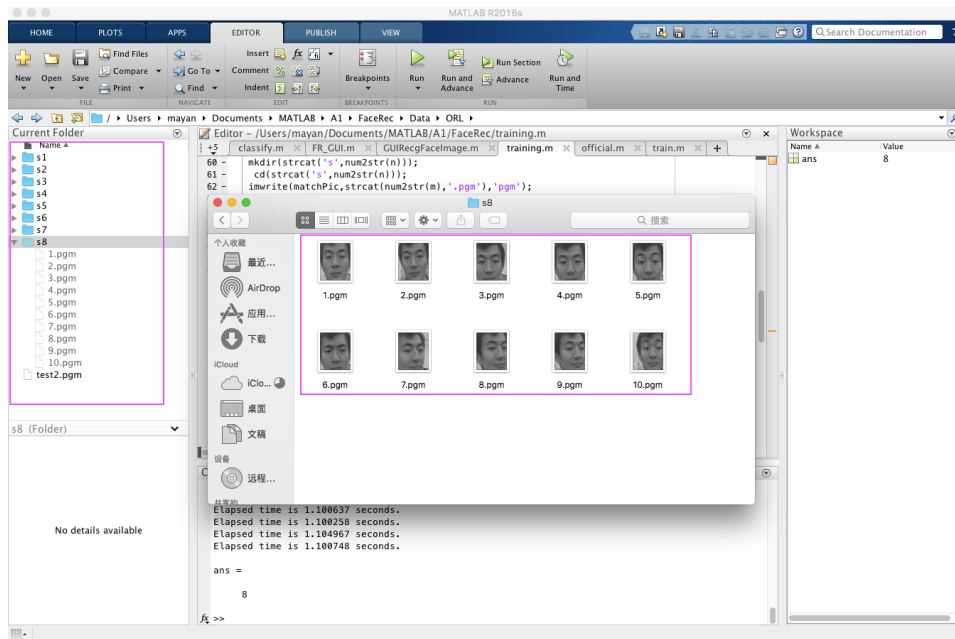


Figure 14: Adding your Face into Database 2

Third, training the database using the parameters I got from the cross validation.

Where  $C = 8$  and  $\gamma = 0.0315$ .



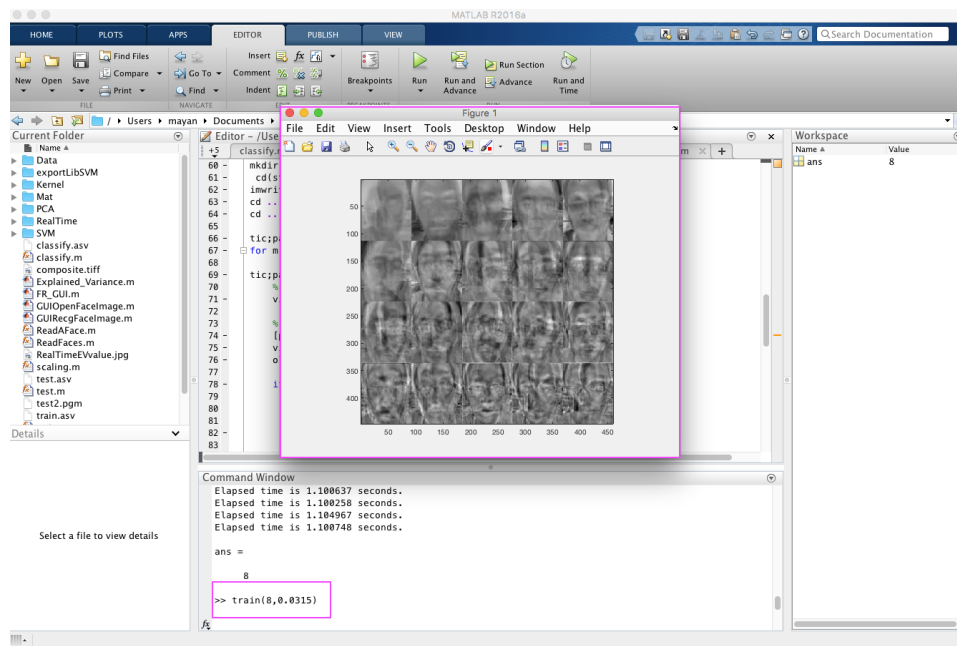


Figure 15: Training the Database and Get the Eigenfaces

In this step you can also use function 'test.m' in command area. The result for 40 samples, the correct recognition rate is 90%

Fourth. Real-Time Face Recognition. I wrap the function in 'official.m'. You can type 'official' in command and result shown as follows:

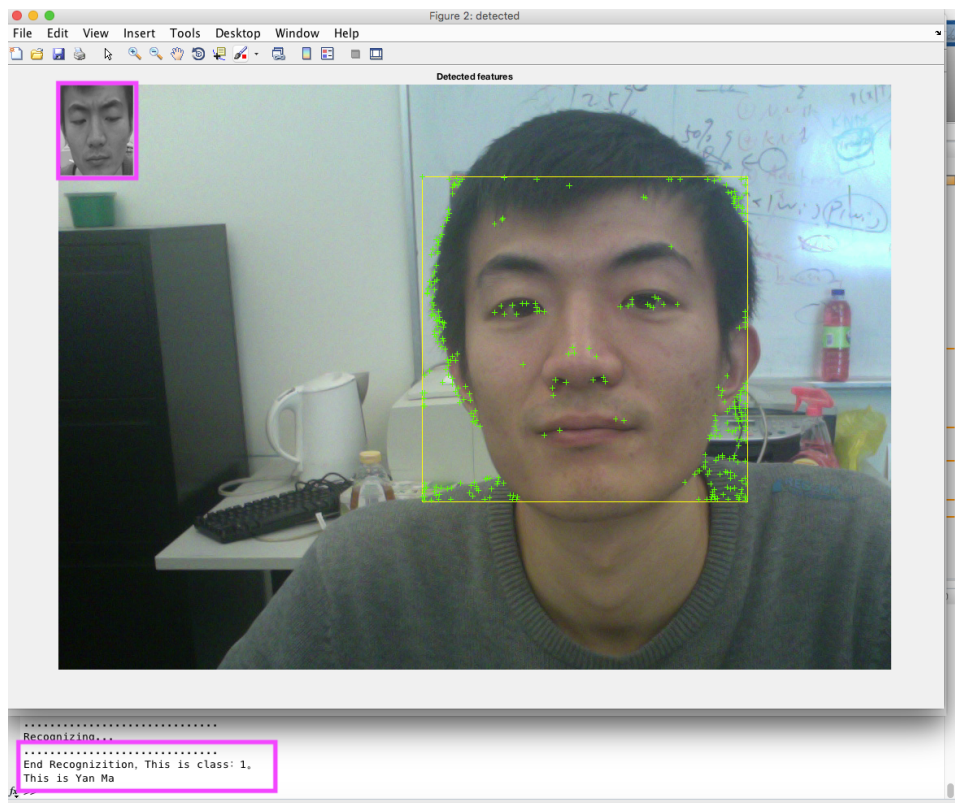


Figure 16: Result

As shown in Figure 16, machine recognized my face correctly and display my name. But because of the background, the tracking point is not good enough.

## 7 Testing and Robustness Analysis

In this section, I will test the classifier and adding some noises to check how it works.

## 7.1 Testing Example 1: Glasses v.s Without Glasses

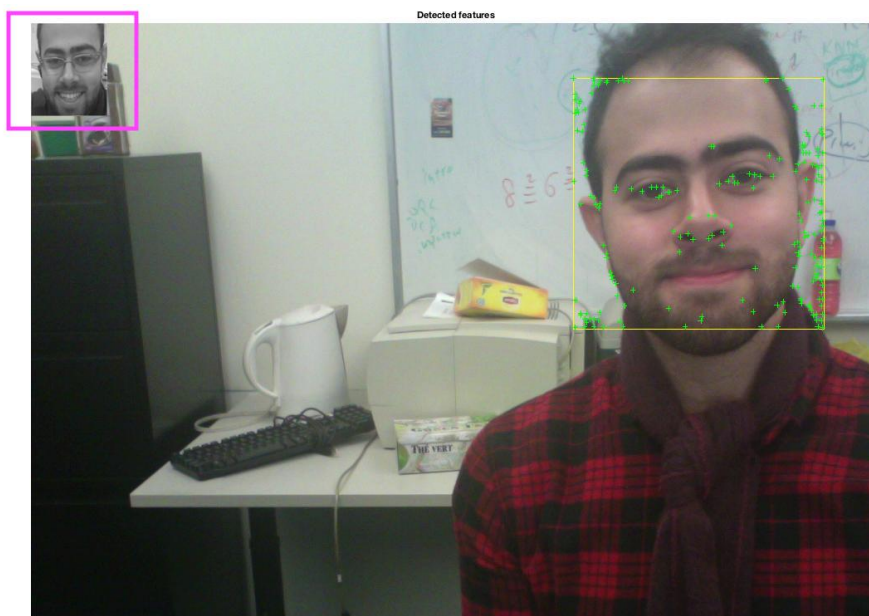


Figure 17: Testing Example 1

The first testing example is my friends, where he wears glasses in his face database, but he tests without glasses. The result is shown in Figure 17. As the result shows, the machine still can recognize the face of him, and the tracking points fit good for the face outline.

## 7.2 Testing Example 2: Another Face Testing

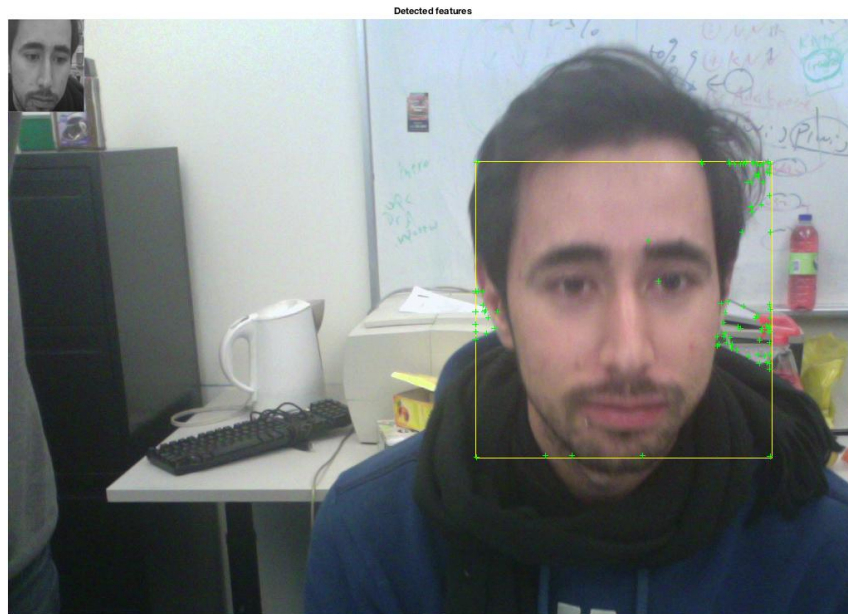


Figure 18: Testing Example 2

In this example. I test my another friends. The tracking points are influenced by the background of the video frame, but the face still be detected, but when the student move, the quadrangle doesn't tracking well.

### 7.3 Testing Example 3: Hat v.s Without Hat

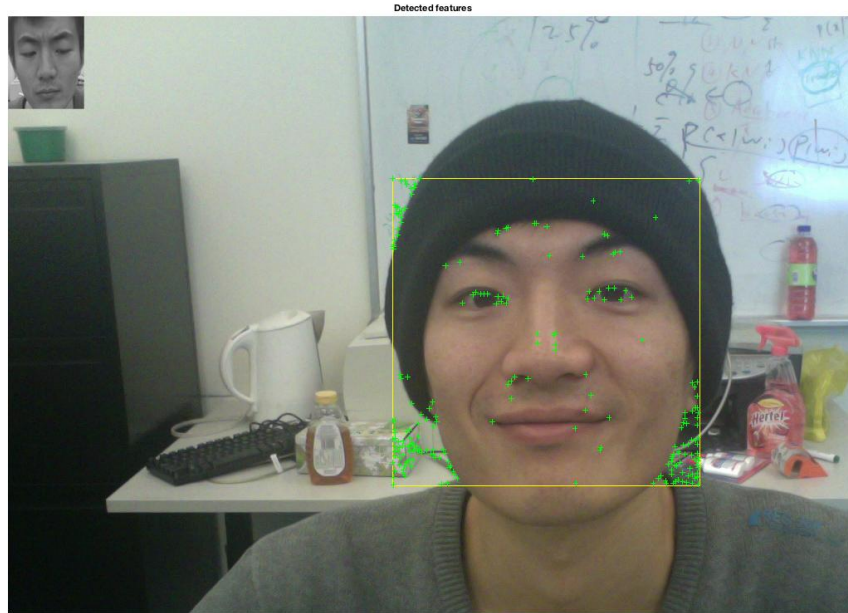


Figure 19: Testing Example 3

In this example I add hat as a noise, where I am not wear hat in my face database, but I test with a hat. The result is shown in Figure 19. As the result shows, the machine still can recognize the my face, and the tracking points fit good for the face outline.

### 7.4 Testing Example 4: Combine ORL face database and My Face Database

In this example, I put my own 8 faces database into ORL and remove the first 8 ORL faces. The correct recognition rate is down to 75%. And the face recognition becomes unstable.

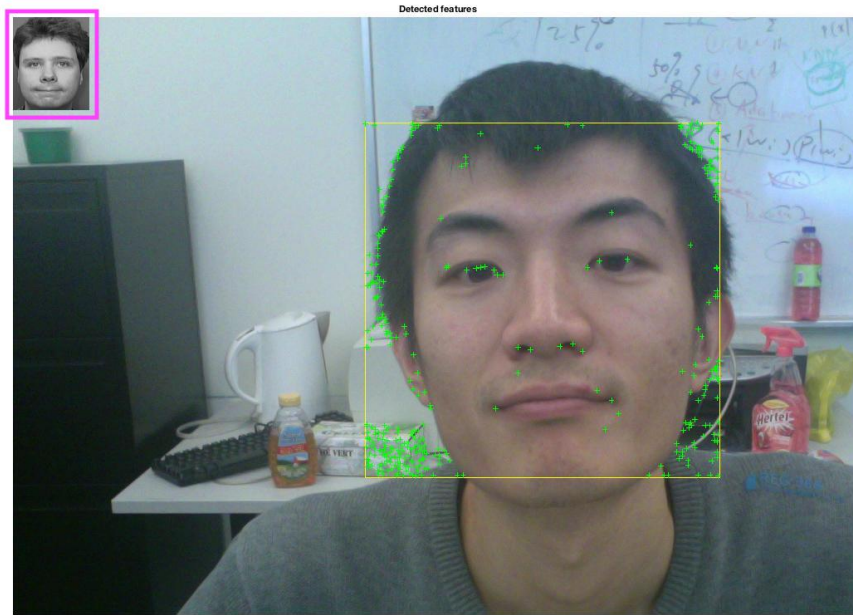


Figure 20: Using Real Time Screen Cut Face Tests Classifier Bases on Mixed Face Database

As we can see from the Figure 20, the testing give a wrong recognition. I think it is because I have 392 ORL face database and 8 my own database, but only the size of image is same, but background and lighting direction e.t.c is different, therefore, compared with ORL face databases, my own face database can consider as noisy. To test my hypothesis, I test 10 examples faces from ORL database, all the examples get the correct result.



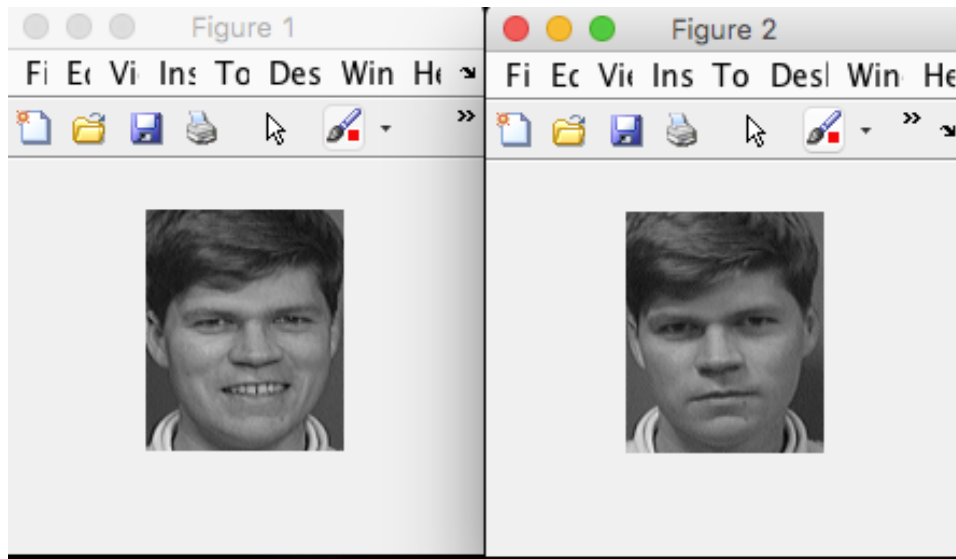


Figure 21: Using ORL Faces Tests Classifier Bases on Mixed Face Database

Therefore, this face recognition system is not robust for the change of environment, we have to collect and establish database at the same environment to make sure this system have a good recognition rate.

## 8 Conclusion and Future Work

In this project, I implement static and real time face recognition based on PCA and SVM. For static face recognition, we have 400 faces database and we use 200 samples as training data and 200 samples as test data, we got 83.5% correct recognition rate. For real time, I establish a 80 faces database, 8 person and 10 for each. I use the same strategy as previous work, 40 samples as training data and 40 samples for testing. We got 90% correct recognition rate.

Then, I give a detail discussion for the robustness of this system in section 7. First, I use my own database test some of my friends faces, with different condition (namely, glasses, hats, expressions...), and I get good recognition results. However,

when I combine my database and ORL together (8 faces in from my database and 392 from ORL), I find that, the classifier still works well on ORL faces but have a bad result for real time recognition results.

For the future work, I still need to find a way to get a better recognition rate. These methods main including 2 parts, the first one is to compare different extractor, like Linear Discriminant Analysis(LDA). In some cases, PCA cannot get a good results, for example

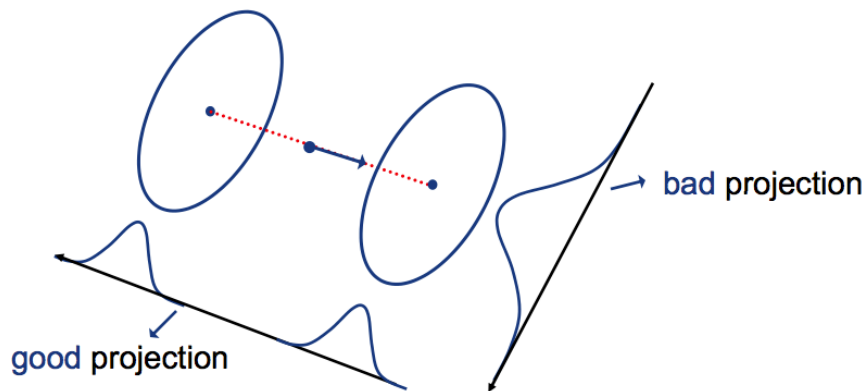


Figure 22: PCA Gives a Bad Projection Example

Here in Figure 22, the result for PCA is a bad projection.

The 2nd one is to compare different classifier. In face recognition, we have a lot of classifier to choose, Neural Network, Convolutional Neural Networks. e.t.c, In the future work, I will do a comparison of these methods and choose a better one which can give a higher correct recognition rate.



## 9 Appendix

In this report, I just use grid.py function for cross-validation, therefore, I just explain how to configure the library and how to use grid.py function to do cross-validation. First, make sure that Python and Gnuplot are installed on your computer.

Function grid.py needs a specific format, shown as follows:

```
<label> <index1>:<value1> <index2>:<value2> ...
```

Figure 23: Format for LibSVM

The file function "export.m" export the training data into LibSVM demanded format. The input is the path for the train data matrix, the output is a 'trainData.txt' file under exportLibSVM path.

Then, configure the path for grid.py function under ' /libsvm-3.22/tools/', change the path for 'svmtrain\_pathname' and 'gnuplot\_pathname', where you have to find 'gnuplot.exe' and 'svmtrain.exe'. Mine is showing as follows:

```
class GridOption:
    def __init__(self, dataset_pathname, options):
        dirname = os.path.dirname( file )
        if sys.platform != 'win32':
            self.svmtrain_pathname = os.path.join(dirname, '/Users/mayan/Desktop/libsvm-3.22/svm-train')
            self.gnuplot_pathname = '/usr/local/Cellar/gnuplot/5.2.2_1/bin/gnuplot'
        else:
            # example for windows
            self.svmtrain_pathname = os.path.join(dirname, r'..\windows\svm-train.exe')
            # svmtrain_pathname = r'c:\Program Files\libsvm\windows\svm-train.exe'
            self.gnuplot_pathname = r'c:\tmp\gnuplot\binary\pgnuplot.exe'
```

Figure 24: Grid.py Path Configuration

Now, we can feed the training data into 'grid.py'. Put the 'trainData.txt' under grid.py path

```

Last login: Thu Dec 14 23:04:28 on ttys000
ocn37:~ mayan$ cd desktop/libsvm-3.22/tools
ocn37:tools mayan$ python grid.py trainData.txt

```

Figure 25: Running Cross Validation

```

tools — -bash — 80x24
[local] 1 3 57.0 (best c=128.0, g=0.0078125, rate=97.5)
[local] 1 -9 91.0 (best c=128.0, g=0.0078125, rate=97.5)
[local] 5 -3 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] -1 -3 91.0 (best c=128.0, g=0.0078125, rate=97.5)
[local] 11 -3 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] -3 -3 91.0 (best c=128.0, g=0.0078125, rate=97.5)
[local] 9 -3 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 3 -3 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 15 -3 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] -5 -3 91.0 (best c=128.0, g=0.0078125, rate=97.5)
[local] 7 -3 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 1 -3 92.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 -7 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 -1 94.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 -13 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 1 81.0 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 -11 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 -5 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 -15 93.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 3 57.0 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 -9 97.5 (best c=128.0, g=0.0078125, rate=97.5)
[local] 13 -3 97.5 (best c=128.0, g=0.0078125, rate=97.5)
128.0 0.0078125 97.5
ocn37:tools mayan$

```

Figure 26: Cross Validation Result

The result is shown in the terminal, for ORL face database, good parameters are  $C = 128$  and  $\gamma = 0.0078$ .

## References

- Faruqe, M. O. and Hasan, M. A. M. (2009). Face recognition using pca and svm. In *Anti-counterfeiting, Security, and Identification in Communication, 2009. ASID 2009. 3rd International Conference on*, pages 97–101. IEEE.
- Guo, G., Li, S. Z., and Chan, K. (2000). Face recognition by support vector machines. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 196–201. IEEE.
- Mstafa, R. J. and Elleithy, K. M. (2016). A video steganography algorithm based on kanade-lucas-tomasi tracking algorithm and error correcting codes. *Multimedia Tools and Applications*, 75(17):10311–10333.
- Shi, J. et al. (1994). Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE.
- Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE.