

# Face Recognition using Eigenfaces

Jiahui Shao, ID: 210934641

School of EECS, Queen Mary University of London, UK

ec211077@qmul.ac.uk

## Table of Content

1. Getting Started .....	2
2. Complete the lab2.m file.....	2
2.1 Read in the training and test images .....	2
2.2 Construct the mean image and the covariance matrix .....	2
2.3 Compute the Eigenface of the training set.....	2
2.4 Display the Mean Image .....	2
2.5 Display the first 20 Eigenfaces .....	3
2.6 Project both training images and testing images onto the first 20 Eigenfaces.....	4
2.7 Compute the distance from the project test images to the projected training images.....	4
2.8 Display the top 6 best matched training images for each test image.....	4
2.9 Compute the recognition rate using 20 Eigenfaces.....	5
2.10 Investigate the effect of using different number of Eigenfaces for recognition .....	6
2.11 Investigate the effect of K in K-Nearest Neighbour (KNN) classifier. ....	7

## List of Figures

1. Eigenface .....	2
2. Mean Image .....	2
3. The first 20 Eigenfaces .....	3
4. The top 6 best matches training images.....	4
5. Recognition_rate_20.....	5
6. averageRR table (size: 1*40).....	6
7. Recognition rate (Eigenface: 1-40).....	6
8. Recognition rate (k: 1-40).....	7

## 1. Getting Started

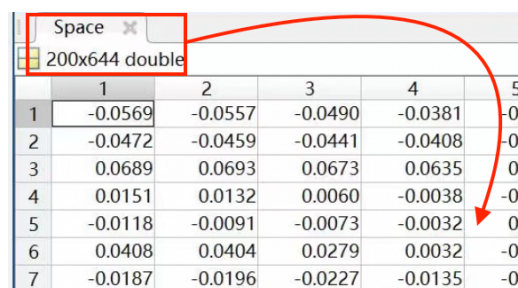
## 2. Complete the lab2.m file

### 2.1 Read in the training and test images

### 2.2 Construct the mean image and the covariance matrix

### 2.3 Compute the Eigenface of the training set

The parameter represented by Eigenface is *Space*, which has the shape 200\*644, as shown in Fig.1 below:



	1	2	3	4	5
1	-0.0569	-0.0557	-0.0490	-0.0381	-0.0381
2	-0.0472	-0.0459	-0.0441	-0.0408	-0.0408
3	0.0689	0.0693	0.0673	0.0635	0.0635
4	0.0151	0.0132	0.0060	-0.0038	-0.0038
5	-0.0118	-0.0091	-0.0073	-0.0032	-0.0032
6	0.0408	0.0404	0.0279	0.0032	-0.0032
7	-0.0187	-0.0196	-0.0227	-0.0135	-0.0135

Fig. 1. Eigenface

Eigenface works by transforming a face from one pixel space to another and then calculating similarity in the new space. The covariance matrix of all the facial photos in the training set is decomposed into eigenvectors to achieve this. "Eigenface" is the name given to these eigenvectors.

### 2.4 Display the Mean Image

The mean image is shown in Fig.2 below:

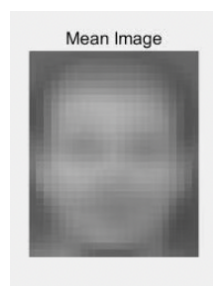


Fig. 2. Mean Image

Calculating the mean image is a very important step in face recognition. The solution is as follows:

Each training image should be stretched into a lengthy column and then combined to form a huge matrix  $W$ . If each image is  $D \times D$  in size, then each face sample's dimension after stretching into a column is  $Y = D \times D$ . If there are 30 images, the sample matrix  $W$  has a dimension of  $30 \times Y$ . To get a "mean image," which is a  $1 \times Y$  vector, add 30 images all in the relevant dimensions and average them. Finally, display the mean image.

## 2.5 Display the first 20 Eigenfaces

The relevant codes are as follows:

```
%-----  
% The first step is normalization. Make sure Shape is in 0,255 range  
Space_normalize = normalize_0_255(Space);  
% The second step is to Display of the 20 first eigenfaces  
figure;  
x=4;  
y=5;  
for i=1:20 % Display the first 20 eigenfaces in turn  
    Image = uint8 (zeros(28, 23)); % Initialise the shape of eigenface  
    for k = 0:643  
        Image( mod (k,28)+1, floor(k/28)+1 ) = Space_normalize(i,k+1); % Reshape  
    end  
    subplot (x,y,i); % Set the relative position of each subplot  
    imshow (Image);  
    title(i);  
end  
%-----
```

The solution:

The first step in displaying the Eigenface is to normalise the *Space* parameter so that the values are in the range 0-255, as they have to be converted to a pixel range in order to display them properly. Here I have defined a *normalize\_0\_255.m* function with the following code:

```
%-----  
function result = normalize_0_255(A)  
ymin=0;  
ymax=255;  
xmin = min(min(A));  
xmax = max(max(A));  
result = round((ymax-ymin)*(A-xmin)/(xmax-xmin) + ymin);  
end  
%-----
```

The second step involves using *for* loop and *subplot* functions to output the 20 eigenfaces images in turn. One thing to note is that the initial shape is 1\*644, so we need to reshape the shape to 28\*23 first. Selecting the top 20 eigenfaces means selecting the top 20 eigenvectors with larger eigenvalues.

The output is shown in Fig.3 below:



Fig. 3. The First 20 Eigenfaces

Some understandings of eigenface:

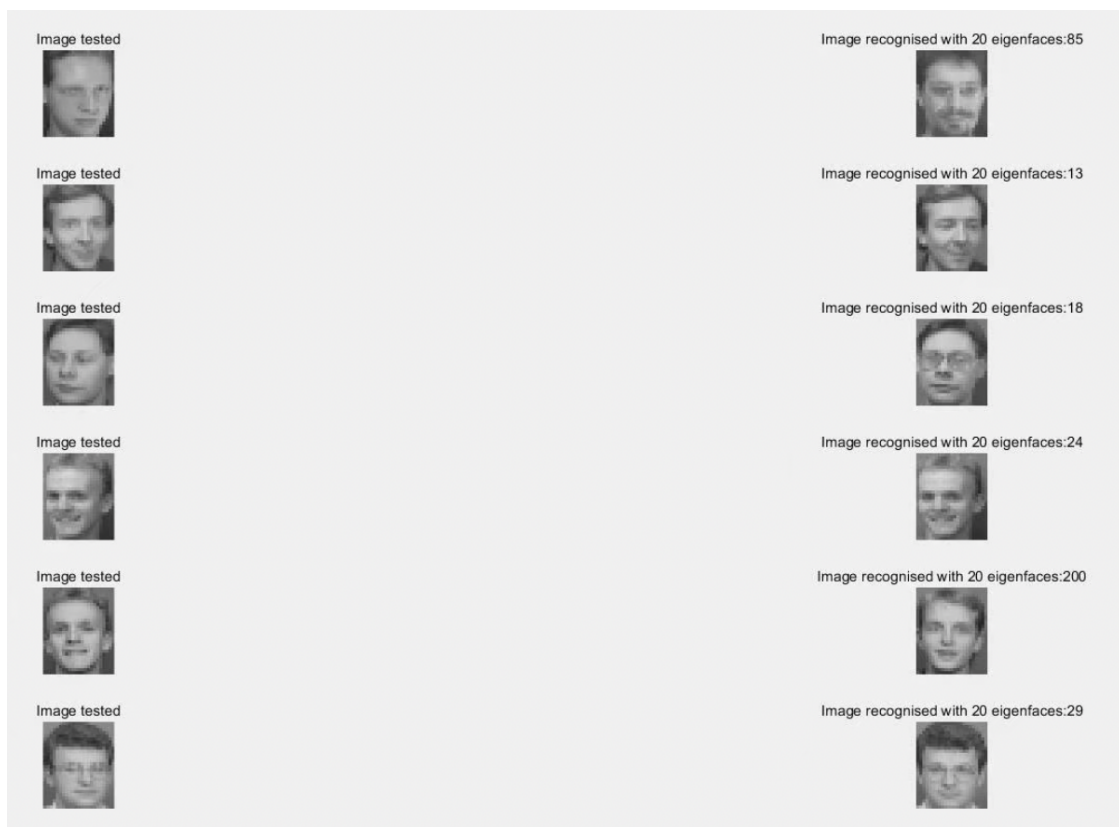
By looking at the Fig.3 above, the eigenface resembles a ghost. Eigenface has a wide range of applications, and its most immediate application is face recognition. In this demand, eigenface has an advantage in terms of efficiency compared to other means, as they are very fast to compute and can process many faces in a short time. However, one problem with the use of eigenfaces in practice is that they can cause a significant drop in recognition rates when subjected to different lighting conditions and imaging angles. Therefore, it is preferable to use eigenfaces for recognition based on uniform lighting conditions and frontal face images.

## 2.6 Project both training images and testing images onto the first 20 Eigenfaces

## 2.7 Compute the distance from the project test images to the projected training images

## 2.8 Display the top 6 best matched training images for each test image

The output is shown in Fig.4 below:



**Fig. 4.** The top 6 best matched training images

The explanation:

By looking at Fig.4 above, we can see that the recognition results are relatively good based on 20 eigenfaces. For example, the pair of faces in the second row have different expressions; the pair of faces in the third row, one with and the other without glasses; and the pair of faces in the last row have different angles. But with these noises, the corresponding faces are correctly recognised.

## 2.9 Compute the recognition rate using 20 Eigenfaces

The relevant codes are as follows:

```
%-----  
Threshold = 20; % using 20 Eigenfaces  
  
% Calculate the distances from the project test images to the project training  
images  
Distances=zeros(TestSizes(1),TrainSizes(1));  
for i=1:TestSizes(1),  
    for j=1: TrainSizes(1),  
        Sum=0;  
        for k=1: Threshold,  
            Sum=Sum+ ((Locationstrain(j,k)-Locationstest(i,k)).^2);  
        end,  
        Distances(i,j)=Sum;  
    end,  
end,  
  
Values=zeros(TestSizes(1),TrainSizes(1));  
Indices=zeros(TestSizes(1),TrainSizes(1));  
number_of_test_images=zeros(1,40);% Number of test images of one given person  
for i=1:70,  
    number_of_test_images(1,Identity(1,i))= number_of_test_images(1,Iden-  
tity(1,i))+1;  
    [Values(i,:), Indices(i,:)] = sort(Distances(i,:));  
end,  
  
rec_rate = []; % Initialize rec_rate  
for i = 1: length(Imagestest(:,1))  
    % if the indices of train does not match with Identity in test then rate is 0.  
    if ceil(Indices(i,1)/5) == Identity(i)  
        rec_rate(i) = 1;  
    else  
        rec_rate(i) = 0;  
    end  
end  
  
recognition_rate_20 = sum(rec_rate)/70 *100; % the output  
%-----
```

The solution:

First set the value of *Threshold* to 20, meaning that only 20 Eigenfaces are used for the subsequent calculations. Next, the distance from the projected test images to the projected training images is calculated to obtain a parameter: *Distances*. Then the values of each row of *Distances* are sorted to obtain *Indices*. Finally, the *Indices* are compared with the *Identity* in test images. If the indices of train matches with the identity in the test, then rate is 1, if the indices of train does not match with Identity in test then rate is 0. The final calculated recognition rate is **82.8571**.

The output is shown in Fig.5 below:

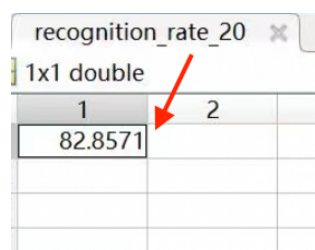


Fig. 5. recognition\_rate\_20

## 2.10 Investigate the effect of using different number of Eigenfaces for recognition

The outputs are shown in Fig.6 and Fig.7 below:

averageRR										
1x40 double										
	1	2	3	4	5	6	7	8	9	10
1	11.4286	40	51.4286	61.4286	70	72.8571	77.1429	84.2857	84.2857	85.714
2										

Fig. 6. averageRR table (size: 1\*40)

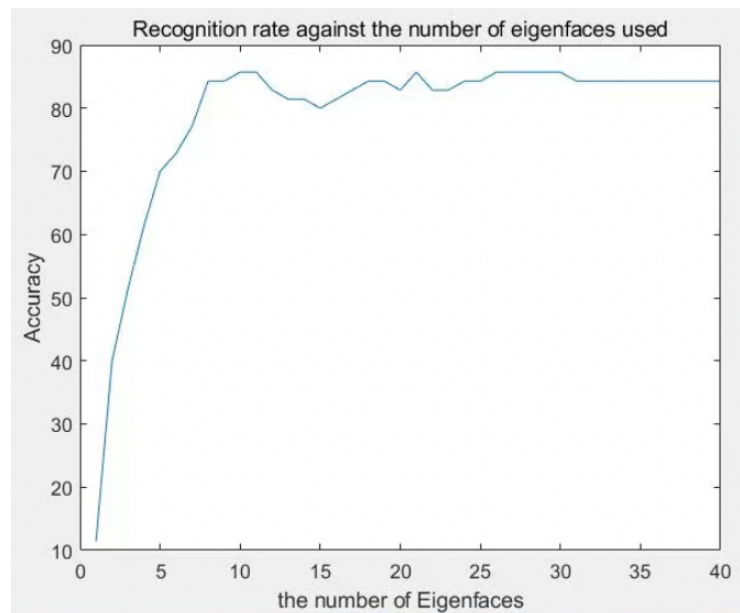


Fig. 7. Recognition rate (Eigenface: 1-40)

The solution:

The solution to this step is roughly the same as the previous step 2.9, except that step 2.9 only calculates the recognition rate for eigenface = 20, whereas this step calculates the recognition rate for eigenface from 1 to 40. Finally, a matrix *averageRR* (Fig.6) with a shape of 1\*40 is calculated and visualised to obtain the corresponding recognition rate figure (Fig.7).

The explanation:

As can be seen from Fig.7 above, the curve increases sharply in the initial eigenface 1–10 interval, but after 10, it stays in a relatively flat state. This means that, in practice, the number of eigenfaces does not need to be large to achieve a relatively satisfactory recognition rate. In addition, in practice, as the number of eigenfaces increases, the reconstructed faces become clearer. This is because the more eigenvectors are used for face reconstruction, the less information is lost and the clearer the reconstructed face becomes. For this task, 10 Eigenfaces should be enough. If too many Eigenfaces are taken, the recognition rate does not improve significantly and the computational cost increases.

## 2.11 Investigate the effect of K in K-Nearest Neighbour (KNN) classifier.

The relevant codes are as follows:

```
%-----
AVG = zeros(1,40);
tem = ones(5,1);
train_labels = [tem;2*tem;3*tem;.....;38*tem;39*tem;40*tem]; % training set labels
for m=1:40,
    k = m; % K values are taken in order from 1 to 40.
    mdl = fitcknn(Locationstrain,train_labels,'NumNeighbors',k); % KNN classifier
    class = predict(mdl,Locationstest); % predict labels
    rec_rate = [];
    for i = 1: length(Imagestest(:,1)),
        if ceil(class(i,1)) == Identity(i)
            rec_rate(i) = 1; % if match, 1
        else
            rec_rate(i) = 0; % if not match, 0
        end
    end
    recognition_rate_KNN = sum(rec_rate)/70 *100; % calculate recognition rate
    AVG(1,m) = recognition_rate_KNN;
end
figure;
plot(AVG(1,:)); % Display the recognition rate figure
xlabel('the value of k'); ylabel('Accuracy');
title('Recognition rate against the value of k used (KNN classifier)');
%-----
```

The solution:

I would like to calculate the trend in the recognition rate for the case where the value of  $k$  is taken from 1 to 40. First, I need to define a parameter called "*train\_labels*" to hold the labels of the training images. It is known that the labels of the given training images are in groups of 5, so there are 40 groups in total, with labels ranging from 1 to 40. Then the *for* loop and *fitcknn()* functions are called to predict the *class* for different values of  $k$  ( $70 \times 1$ ). Next, the results of *class* are compared with *Identity*. If the indices of train matches with the identity in the test, then rate is 1. If the indices of train does not match with Identity in the test then rate is 0. Finally, output the figure of the recognition rate for different values of  $k$ .

The output is shown in Fig.8 below. It can be clearly seen that the overall trend of the curve is downward, so it is sufficient to take a default value of 1 for  $k$ .

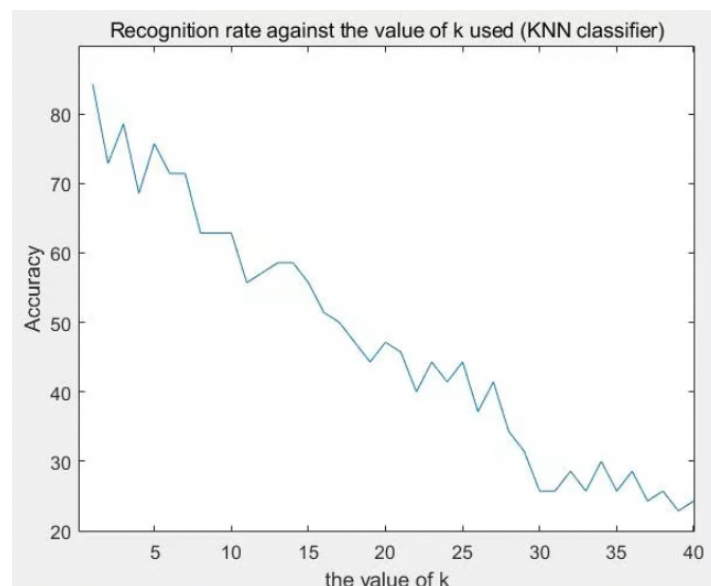


Fig. 8. Recognition rate (k: 1-40)

The explanation:

The basic idea of KNN face recognition is to find the  $k$  faces closest to the face to be tested by converting the image of a face into an eigenvector and then training the dataset by calculating the Euler distance. However, the KNN algorithm itself has some advantages and disadvantages. The advantages are that it is insensitive to outliers and has high computational accuracy, but the disadvantages are that the time complexity and space complexity are relatively high. In addition, the  $k$ -value needs to be defined manually and it takes several experiments to find the optimal value. As can be observed from Fig.8 above, the recognition rate in this task is highest when the  $k$  value is taken as 1, which is about 85%. Then, as the value of  $k$  increases, the recognition rate decreases. At the end, the recognition rate is only about 30% when the  $k$  value is 40. The possible reason for this is that a larger  $k$  value, which corresponds to a point that is too far away, will also influence the classification of unknown objects and will produce an underfitting situation, i.e., not really classifying the unclassified objects. In this step of 2.11, the best recognition is achieved with a  $k$  value of 1.