Applied Mathematics

Project : Face Recognition using PCA

Group Members : Yu Liu, Di Meng

Submitted to : Desire Sidibe

Dated : 26/12/2016

Centre Universitaire Condorcet - Master Computer Vision

Contents

1 Objective

The main objective of this project is carry out basic face recognition using principle component analysis (PCA). In order to fully implement this project, topics of affine transformation, backward mapping, matrix operation and PCA covered in the applied math and image processing classes are required.

2 Introduction

The basic idea of any face recognition system is to extract a set of discriminative features from the face images. Considering having a 2D face image inputted to the system, we aim to extract features from the image and compare them with those of a set of training images in the database; the output of such system is the face image in the database that best matches the input face. However, for 2D face images, each pixel can be considered a feature (equvalent to a variable). Even small images with size of 100×100 would have 10000 pixels to consider. Our goal here is to perform face recognition analysis with reduced number of variables. In other words, we use PCA analysis to simplify the recognition problem by reducing dimensionality.

Prior to recognition, a normalization step is crucial to unify the scale, orientation and location of features from a face. Both tasks (normalization and recognition) are addressed in this project and discussed in the report.

For this project, we choose to use MatLAB, mainly due to its ease of handling matrices operations.

3 Methodology

The methodology can be classified into two major parts: normalization and face recognition. In our program, normalization is to be considered crucial prerequisite for face recognition; all face images (both the training and test set) employed by the recognition algorithm are assumed to have first been normalized.

3.1 Normalization

Our goal of normalization is to map five features of a face to each of their predetermined locations confined by a fixed 64×64 window. Features include left and right eye, nose, and lip (left-end and right-end point). The predetermined locations are provided.

The average locations of the features over all faces are defined by $\bar{F}$ represented by a 5×3 matrix. The first column is composed of the average x position of each of the five features, while the second column is composed of the average of y positions. The position data is organized in advance and loaded to MatLAB from a pre-made excel file. Hence, x and y position data is to be read from the excel file, where each row in the spreadsheet consists of a single person's feature coordinates. The third column of the matrix consists of five ones for

ease of handling matrix operation. Initially, we assign the feature locations of the first face data to $\bar{F}$.

The affine transformation is a 3×2 matrix as follows, where a_{11}, a_{12} and b_1 correspond to the best rotation and translation transformations that align x positions of all five features to the respective, predetermined locations. Likewise, a_{21}, a_{22} and b_2 transform features' y positions to predetermined locations.

$$\begin{bmatrix} 13 & 20 \\ 50 & 20 \\ 34 & 34 \\ 16 & 50 \\ 48 & 50 \end{bmatrix} = \begin{bmatrix} \mathrm{x}_{leftEye} & \mathrm{y}_{leftEye} & 1 \\ \mathrm{x}_{rightEye} & \mathrm{y}_{rightEye} & 1 \\ \mathrm{x}_{nose} & \mathrm{y}_{nose} & 1 \\ \mathrm{x}_{leftMouth} & \mathrm{y}_{leftMouth} & 1 \\ \mathrm{x}_{rightMouth} & \mathrm{y}_{righMouth} & 1 \end{bmatrix} \times \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ b_1 & b_2 \end{bmatrix}$$

Applying the given predetermined locations (also put into matrix form as shown above on the left side of the equation), we obtain the above matrix operation and can easily solve for the affine transformation parameters by using the MatLAB pseudo command x = $\bar{F}$ \b, where b represents predetermined location matrix, and x represents affine transformation.

Once the affine transformation is obtained, we apply it to $\bar{F}$ and update old $\bar{F}$ by this newly obtained $\bar{F}$. We then compute the best affine transformation of each face image (F_i) mapping to $\bar{F}$ (similar procedure as mentioned above, where we represent a face image as a 5×3 matrix and use the x = $F_i \setminus \bar{F}$ command to find transformations). We then again apply each obtained transformation to the corresponding face matrix F_i. Once all face matrices are transformed, we update $\bar{F}$ by averaging all face matrices (the result would be a 5×2 matrix). We check if another iteration is needed, by accessing the difference between the old and newly obtained $\bar{F}$.

In practice, it is observed that $\bar{F}$ quickly converges within 4 iterations. Thus we simply set 10 iterations as a terminal condition. Affine transformation for each face found in the last iteration is stored to transform the corresponding face image. Ideally after transformation, the five features of all faces would be mapped closely to their respective predetermined locations confined in a 64×64 window. To ensure the transformed image is sized 64×64, we use MatLAB's tforminv command and define output as 64×64. By using backward mapping, we find the pixel in the input 320×320 image that correspond to each pixel of the $64 \times$ 64 output image. If the found pixel lies outside the logical range for either x or y position (beyond 320.5 or below 0.4) its corresponding output pixel will be set zero; otherwise, we round the found pixel and copy its intensity value in the output.

It is worth mentioning that, the images loaded in MatLAB do not necessarily follow the same order as how position data is organized in the excel file (due to the fact that MatLAB is case-sensitive when loading images, while list ordering in excel is not). In order to correctly apply affine transformation on the right face, a match table is created. The match table searches the excel column of names (names of the faces) aiming to find a match to the filename of the loaded image; if a match is found, match table stores the row number of the position data corresponding to that image. With the row number, each face image can directly access its corresponding position data, thus also the its affine transformation computed by this position data.

The transformed face images, all sized 64×64, are manually separated into the test_images and train_images folders. 96 images (filename ending 1,2, or 3) are classified as training images while the other 55 images (filename ending 4 or 5) are test images.

3.2 Face Recognition

Each training image is reshaped into a one-dimensional vector with length 4096. With a total of 96 training images, the training data becomes a 4096×96 matrix (referred as D).

Two methods are tested and discussed. In **method 1**, the matrix D is first centered by subtracting the mean of features (in this case, a single pixel can be considered as a feature). The covariance is obtained by computing $\frac{1}{95}DD^T$. PCA is then applied on the covariance matrix (4096×4096) using the MatLAB eig() command. The resulted eigenvectors in columns are the principle components.

The feature vector of the training set is obtained by projecting D into the PCA space (only keeping k principle components corresponding to the k largest eigenvalues). The value k is obtained by summing the eigenvalues (in descending order) up to when 85 percent of the sum of all eigenvalues is reached. This steps concludes data training. We project all training images in the PCA space as database.

For face recognition, we input a test image and compute its feature vector. Firstly, the test image is vectorized (4096×1) and then centered by subtracting the mean features of training set. This centered test vector is then projected to the PCA space by multiplying the reduced-dimension principle components of the training set, giving us the test image feature vector.

The squared Euclidean distance of the test image feature vector and all those of every training image is determined by computing $(Xtest - Xtrain_N)^T(Xtest - Xtrain_N)$ (N refers to the N^{th} training image). The training image with the smallest squared Euclidean distance from the test is considered best match.

Method 1 involves heavy computation when constructing the training database (it takes approximately 2 minutes for an I7 processor to run). A preferred, simplified technique is introduced (referred as **method 2**), where we first compute $\frac{1}{95}D^TD$ as a lower dimension covariance matrix (96×96). We again use the eig() command to obtain the eigenvectors (principle components) of the low-dimension covariance, and then multiply it by D to map the eigenvectors into 4096 space (the resulted eigenvector has dimension of 4096×96). Face recognition in **method 2** follows the same procedure as shown in **method 1**.

4 Descriptions of Experiments

4.1 Normalization

To assess the performance of normalization, we individually inspect the five feature' positions for each transformed face. Because the coordinate data of every face feature may be biased as they are provided by different individuals, we consider the normalization step successful as long as the transformed features fall close enough to the predetermined locations.

4.2 Face Recognition

To more efficiently assess best match accuracy, we prepare a separate MatLAB script that loads all 55 test images as a single 4096×55 matrix (the final script with the user interface aims to allow user to input one test image at a time, which is not efficient for assessing the

overall accuracy). We apply the same procedure as described in section 3.2 to obtain the test images feature vectors (in this case it is a feature "matrix").

One column of the test images' feature matrix at a time, we compute the squared Euclidean distance between the target test's feature and all those of the training images, store the squared distances as a vector of length equivalent to the size of training database (since we measure distances of a single test to all training set). We then take the minimum of the distance vector and use its index to find the corresponding image from the training data - the best match. Because test data and training data originate from image files, we check the filenames (excluding the last numeric character) to confirm if a correct match is found.

We apply the above experiment to assess recognition accuracy on both scenarios (two methods of computing for covariance matrix as mentioned in section 3.2). Moreover, aside from finding the best match, We also check if the correct face is among the first F matches by setting F from 1 (best match) to 15.

5 Results and Analysis

We find that most faces are properly transformed without significant deformation, and features from most faces fall close enough to the predetermined locations. Initially, features of some transformed faces (8 out of 151 faces) are observed to be falling onto incorrect locations, which would hinder recognition. We investigate these source images and their respective coordinate data; it turns out some features (or even all five features from some images) do not match their coordinate data. Such issue is resolved after correcting their feature coordinate data.

For recognition, when **method 1** is used and principle component is obtained as 26, the best match accuracy is 60 percent (33 correct matches out of 55). Correct matching among the first F faces and the respective recognition accuracy are depicted in the plot below indicated by the blue bubbles.

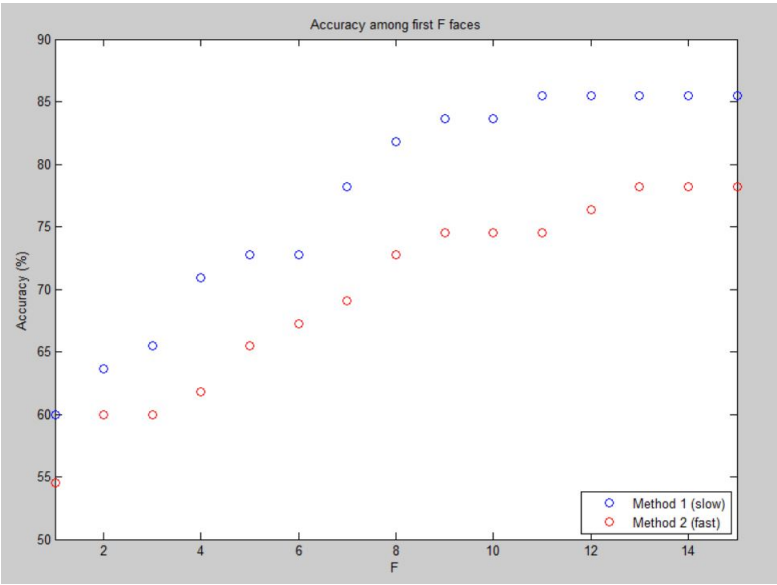

Fig. 1. Accuracy at Different F

It can be observed that the accuracy rises when F increases, and then begins to saturate at around 85 percent when F is large (F is greater than 10). This is somehow expected: an image not recognizable in the first few F faces implies that it does not provide unique feature vector used for differentiation. In such cases, the found matched faces can be considered random and thus will not positively affect accuracy when F increases.

The same assessment is done for **method 2**. The best match accuracy is 55 percent (30 correct matches out of 55). Correct matching among the first F faces and the respective recognition accuracy are illustrated in the same plot above as indicated by the red bubbles. The same "saturation" phenomenon, even though not as prominent as that of **method 1**, can still be observed when F reaches 12.

Principle components (eigenfaces from **method 1**) up to the specified k value (k = 26) is depicted as below.

Fig. 2. Eigenface (method 1)

Principle components (eigenfaces from **method 2**) up to the specified k value (k = 26) is depicted as below.

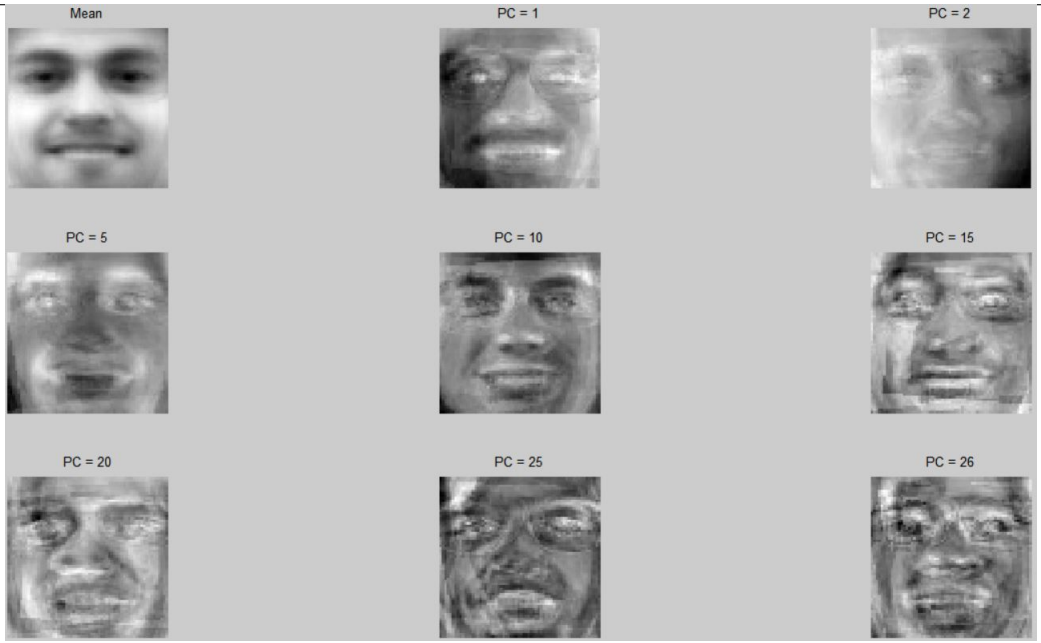

Fig. 3. Eigenface (method 2)

As we observe, the corresponding eigenfaces from each method are similar but not identical, hence causing slight differences in the resulted accuracy. We conclude that the faster approach (method 2) only approximates actual eigenfaces. At the cost of slight accuracy drop, method 2 provides an alternative recognition solution at significantly lighter computation (method 1 requires total of approximately 110 seconds to construct the training database, while method 2 requires 0.9 second).

When individually inspecting incorrect matches, we notice that most test images that are incorrectly identified are mostly side-view faces. The closest matches that these test faces receive are usually other side-view faces from the database.

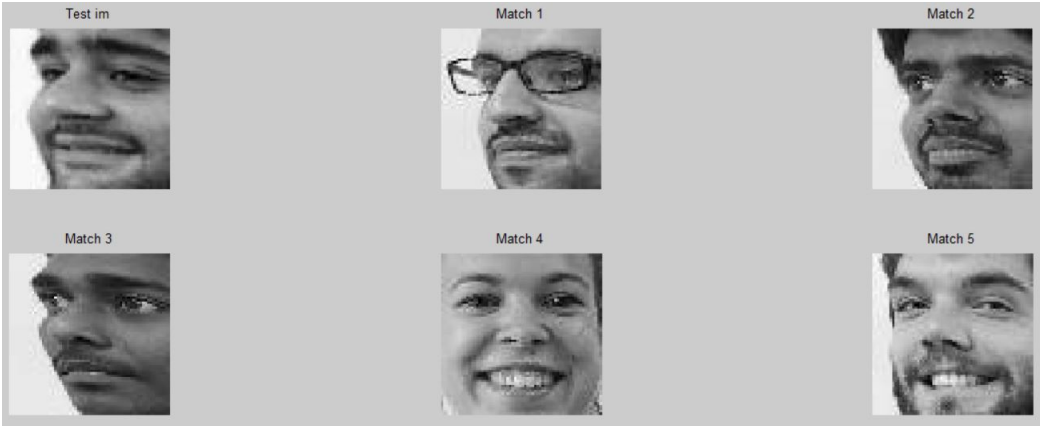

Fig. 3. Incorrect Match Side View (F = 5)

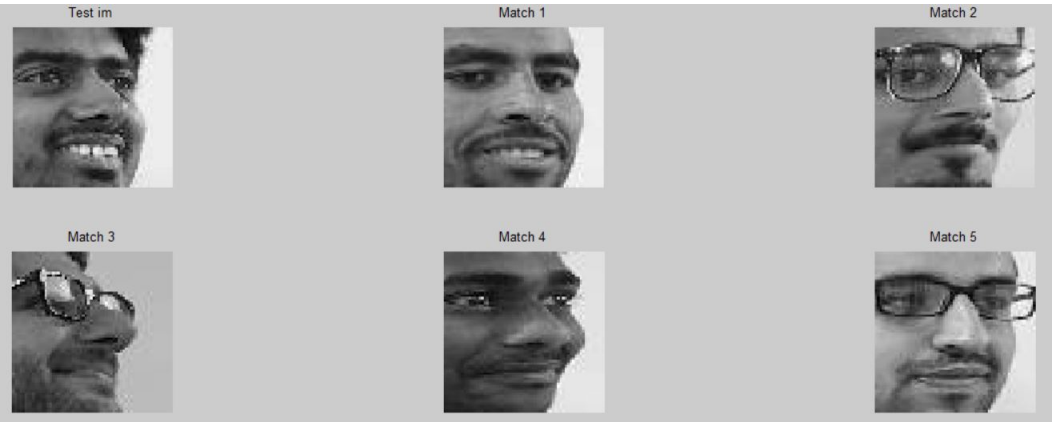

Fig. 4. Incorrect Match Side View (F = 5)

Exceptions do exist, for instance, front-view faces of Nayeem and Ivan are never correctly identified even setting F up to 5.

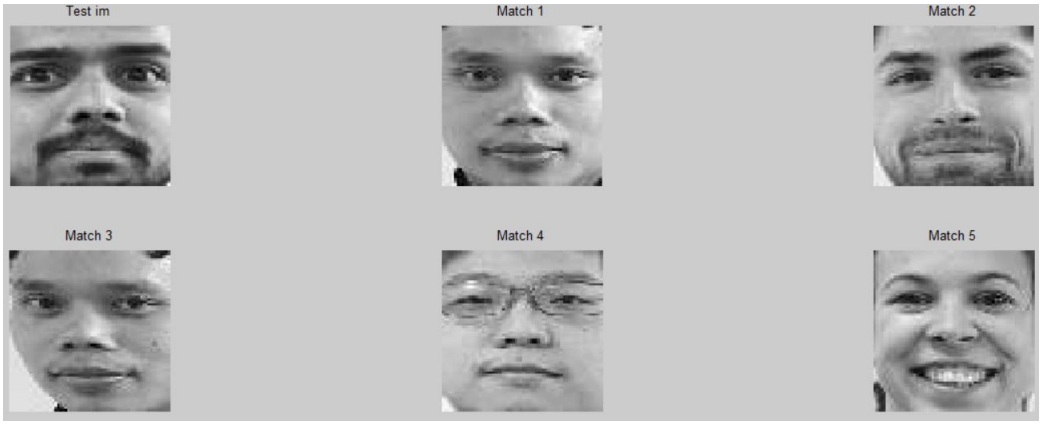

Fig. 5. Incorrect Front-View Match (Nayeem)

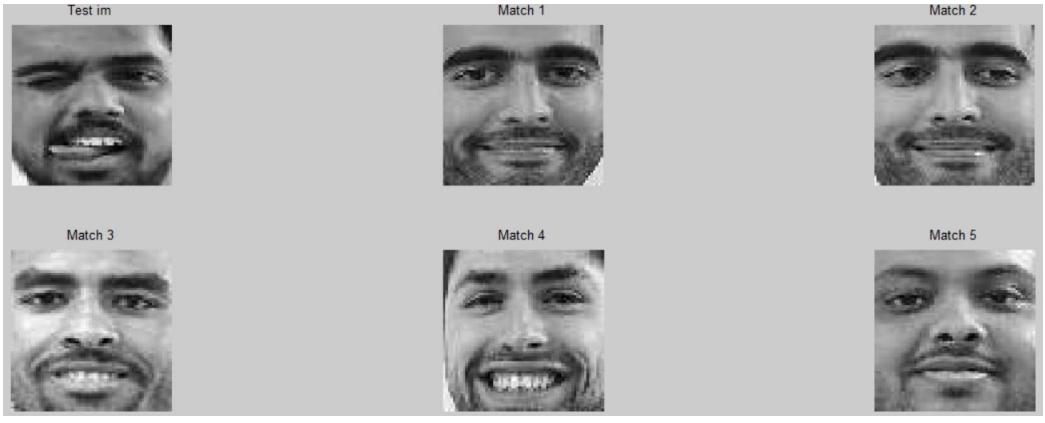

Fig. 6. Incorrect Front-View Match (Nayeem)

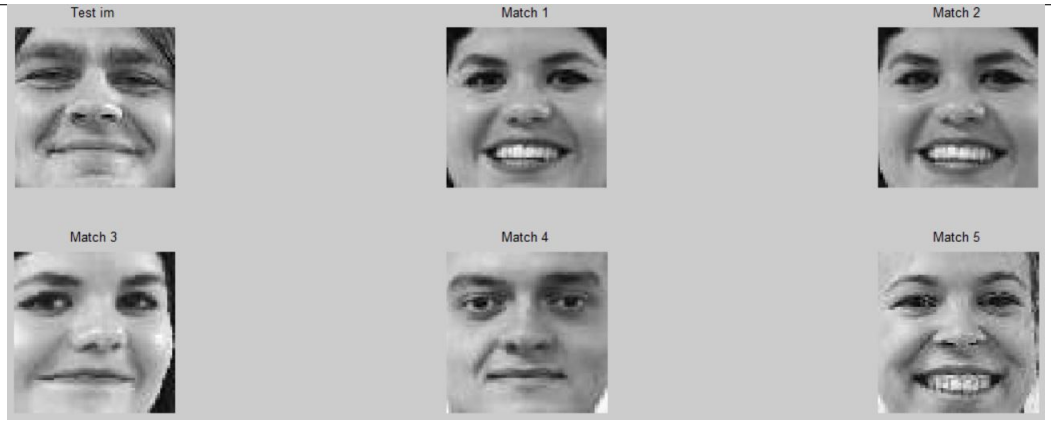

Fig. 7. Incorrect Front-View Match (Ivan)

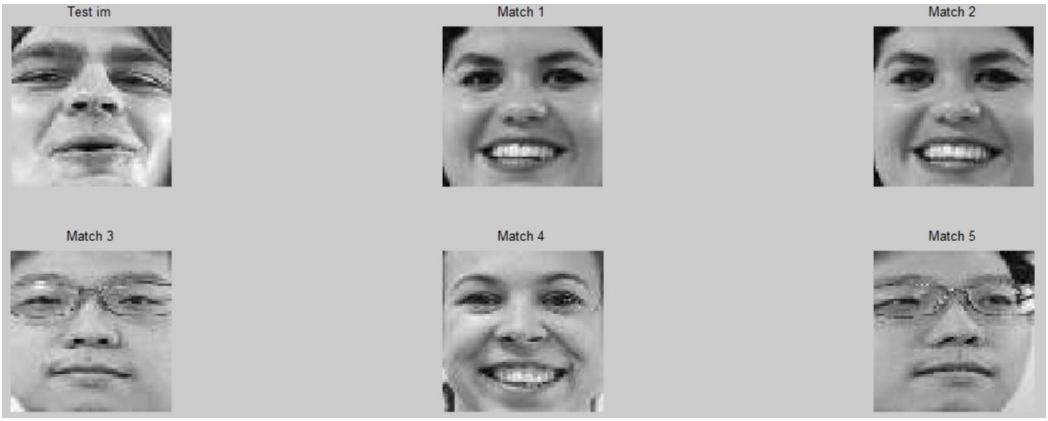

Fig. 8. Incorrect Front-View Match (Ivan)

In contrast, side-view Tajwar, even though the transformation has led to significant deformation, is best match at his front-view face.

Fig. 9. Correct Side-View Match (Tajwar)

6 User Interface

A simple user interface is created for real-time, interactive face recognition. It has four main parts.

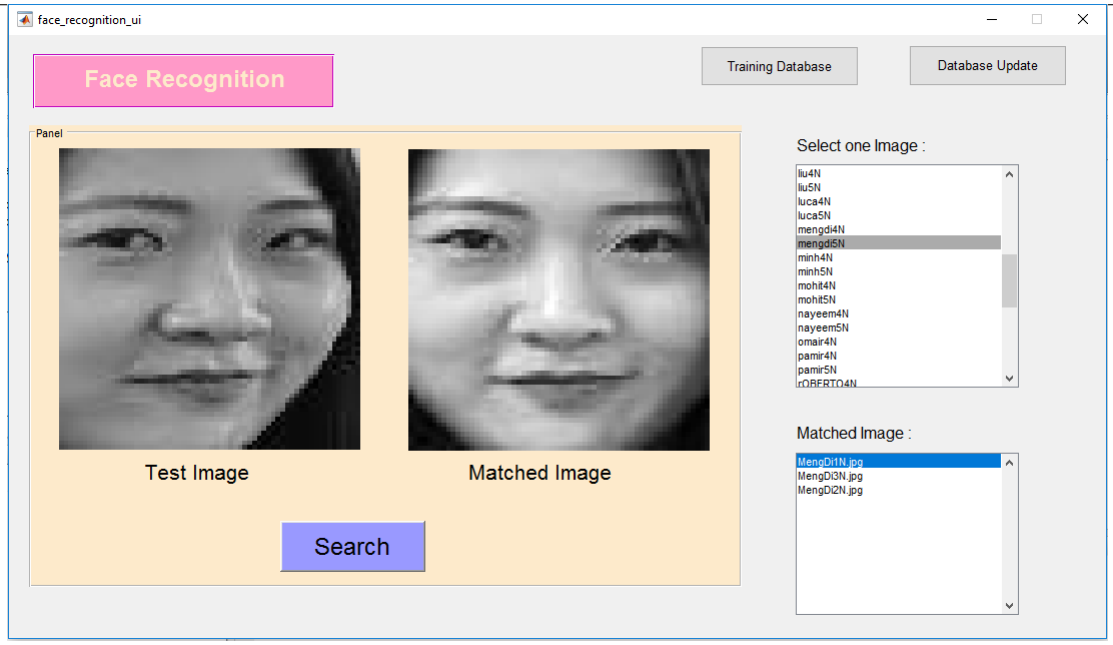

Fig. 10. Simple User Interface

In the top-right of the UI, there are two buttons which are Training Database and Database Update. Firstly, the user has to press the Training Database Button to initiate constructing the training database (by default, method 2 due to its speed). Database Update is used in cases when user has changed the training images in the Training Database folder; this will then update the data to calculate new feature matrix for the training images.

In the right side of the UI, there are two lists. First list is for user to choose the face which they want to test. The next list shows the top three matched faces in rank. User can also click anyone in the matched list to see the recognized face.

The test face will be displayed in the left window once user selects an image in the list of test images. Then the matched face will be shown in the right window as soon as user presses the Search button.

7 Conclusion

From this project, we learn that eigenface provides an easy way to realize face recognition. Its training process follows simple mathematics concept and is easy to code. By selecting a proper principle component, eigenface adequately reduces statistical complexity in face image representation. Moreover, once the database is constructed, face recognition can be achieved in real time. Last but not least, we learn a useful technique (method 2) to address high resolution images.