FACE RECOGNITION USING PCA

Abstract

In this project, you will create a face recognition system. The program reduces each face image to a vector, then uses principal component analysis (PCA) to find the space of faces. This space is spanned by just a few vectors, which means each face can be defined by just a set of coefficients weighting these vectors. Our job is to write the Matlab functions that perform PCA, projection into face space, verifying a user based on a face, finding a face match given a set of user face information.

.

VAISHNAV Mohit and BATERIWALA Malav

Under the Supervision of
DÉSIRÉ Sidibé

Table of Contents

Table of Figures

Abstract:

In this project, you will create a face recognition system. The program reduces each face image to a vector, then uses principal component analysis (PCA) to find the space of faces. This space is spanned by just a few vectors, which means each face can be defined by just a set of coefficients weighting these vectors. Our job is to write the MATLAB functions that perform PCA, projection into face space, verifying a user based on a face, finding a face match given a set of user face information.

Principal Component Analysis

PCA is a technique by which we reduce the dimensionality of data points. For example, consider the space of all *20-by-30*-pixel grayscale images. This is a 600-dimensional space because 600 data values are required to represent the intensities of the 600 pixels. But suppose we only consider images that are valid faces. Such images may lie in a small subspace of the set of all images. If this subspace is a hyperplane spanned by *k* vectors, then we can represent any point that lies on this plane with only a set of k coefficients.

To use PCA to do pattern recognition we first take a representative sample of data points and find the subspace with the desired dimensionality that best fits the sample data. This step needs to be done only once. Once it has been done, you can determine if a new point is a valid data point by projecting it onto the subspace and measuring the distance between the point and its projection.

To use PCA for face recognition we must represent each face image as a vector of pixel values. To generate this vector the face image must be cropped and scaled, and its intensity must be normalized (each vector must have unit length). The cropping can either be done by hand or by searching images for faces. This second option is only possible once the face recognition algorithm has already been trained with some input faces.

Reading In Images

You are given a collection of images of faces from image database. These images are all dimension 320 x 240. From these we compute the “best” eigen faces.

The MATLAB command to read an image is:

```
image = double(imread(imagefile));
```

Building A Matrix Of Images

To learn eigen vectors, the collection of faces must be unraveled into a matrix. To unravel an image of any size, you can do the following:

```
[nrows, ncolumns] = size(image);
image = image(:);
```

The first line above is used only to retain the size of the original image. We will need it to *fold* an unravelled image back into a rectangular image. The second line converts the nrows x ncolumns image into a single nrows*ncolums x 1 vector. In our "training" data every image is the same size (64 x 64) and no rescaling of images is necessary. To read in an entire collection of images, you can do the following:

```
filenames = textread('FILE_WITH_LIST_OF_IMAGE_FILE_NAMES','%s');
nimages = length(filenames);
for i = 1:nimages
image{i} = double(imread(filenames{i}));
end
```

Since all images are the same size, you can get the size of the image from image{1}.

To compose matrix from a collection of k images, the following MATLAB script can be employed (you can also do it your own way):

```
Y = [];
for i = 1:k
Y = [Y image{i}(:)];
end
```

Computing Eigen Faces

Eigen faces can be computed from Y, which is an (nrows*ncolumns) x nimages matrix. There are two ways to compute eigen faces; one is using eigen analysis and the other is by singular value decomposition.

Eigen Analysis

First compute the correlation matrix: corrmatrix = Y * Y'; How large is the correlation matrix? You can compute eigen vectors and eigen values of the correlation matrix as:

```
[eigvecs, eigvals] = eig(corrmatrix);
```

The columns of the eigenvecs matrix (in MATLAB notation the i^{th} column is given by eigvecs(:,i)) are the eigen vectors. The diagonal entries of the eigvals matrix are the eigen values. You can confirm that corrmatrix = eigvecs * eigvals * eigvecs'. To learn more about eigen analysis and the eig() command, type "help eig" into MATLAB.

The intuition for what Eigen vectors and Eigen values really mean is roughly as follows: The eigen vectors are a number of "orthogonal" bases that can be used to compose every image in the collection. What is meant by orthogonal"? Compute the dot product between any two eigen vectors: eigvecs(:,i)*eigvecs(:,j)' (note the apostrophe representing the transpose). If i≠j, then the dot product will be 0. In other words, we cannot explain any part of the i^{th} eigen vector by the j^{th} eigen vector. As a consequence, the parts of any image that are explained by the i^{th} eigen vector cannot be explained by any other eigen vector.

If we composed every image in the collection as a linear combination of our eigen vectors: image = $a_1$eigvecs(:,1) + $a_2$eigvecs(:,2) + $a_3$eigvecs(:,3)..., then a_i indicates the contribution of the ith eigen vector to the image. The ith eigenvalue gives us the RMS value of ai. It tells us how much the ith eigen vector contributes to the images, typically. The lower the ith eigen value, the less the ith eigen vector contributes to the composition of the images. If the ith eigen value is zero, the ith eigen vector does not contribute to the composition of the images at all!

You will find it instructive to plot the Eigen values: plot(diag(eigvals));. MATLAB's eig() function returns eigen vectors in order of increasing importance. The first eigen vector is the least important. The last one is the most important. The plot of eigen values will also show this -- the last eigen value is the largest. Note also that if you have k images in Y, no more than k Eigen values are non-zero. That is because you will not need more than k Eigen vectors to explain k images; the importance of the remaining Eigen vectors is zero.

If you have gone through the above exercise, you will quickly realize that it takes a long time to perform eigen analysis: eig() of a 4096 x 4096 correlation matrix takes a lot of time. If our images were larger (more pixels), it would take much much longer. One of the reasons is that eig() operates on a large correlation matrix. Another it that it computes all eigen values and eigen vectors, whereas we know that if we have k images in Y, only k of the Eigen vectors (and Eigen values) are relevant -- the others have zero importance. The faster way to achieve the same end is via singular value decomposition:

You can use any of these techniques to compute the Eigen faces. The first Eigen face will be used to detect faces in the group photograph.

The eigen face will be in the form of a nrows*ncolumns x 1 vector. To convert it to an image, you must fold it into a rectangle of the right size. MATLAB will do it for you with:

```
eigenfaceimage = reshape(eigenfacevector, nrows, ncolumns);
```

nrows and ncolumns are the values obtained when you read the image. eigenfacevector is the eigen vector obtained from the eigen analysis (or SVD).

Scanning An Image For A Pattern

Let I be a P x Q image. Let E be an N x M Eigen face. The following MATLAB loop will compute the normalized dot product of every N x M patch of I as follows:

```
E = E/norm(E(:)); for i = 1:(P-N)
for j = 1:(Q-M)
patch = I(i:i+N-1,j:j+M-1);
m(i,j) = E(:)'*patch(:) / norm(patch(:));
```

```
end
end
```

m(i,j) is the normalized dot product, which represents the match between the eigen face E and the N x M patch of the image with its upper left corner at the (i,j)-th pixel. Note that we are normalizing both the eigen face and the patch in the above code. This will make sure that the results are invariant to both the size and lighting of the images.

Some Processing Tricks For Better Results

Converting Color to Greyscale:

The test image we are given is a color image. This must be converted to grey scale. To do this, first read the image in as a color image. You can do this imply using imread(), just as you would read a greyscale image. imread() will give you three images, stored together in a three-dimensional array (the elements of which must be accessed using three indicies). So, for example, image(i,j,1) will give you the (i,j)-th pixel of the red image, image(i,j,2) will give you the corresponding green pixel and image(i,j,3) the blue pixel. To convert it to a greyscale image, the red, green and blue images must be added. This can be done by:

```
greyscaleimage = squeeze(mean(colorimage,3));
```

Histogram Equalization

Histogram equalization can be performed in MATLAB using histeq(image). However this must be done before we convert it to doubles. This means that when we read in the training data, the routine for reading in image files must itself change to:

```
image = double(histeq(imread(imagefilename)));
```

Normalization of image:

The main objective of our project is to recognize faces from the collected set of face data. We have collected the five pictures of every student and the idea here is to extract few features from the faces with the goal of reducing the number of variables used to represent the faces. In the next step, we divide the faces into two categories of train and test images. The train images should be selected as one facing the camera and the other two of side face. The locations of five facial features are extracted from every image: left eye, right eye, tip of nose, left mouth corner and right mouth corner as presented below.

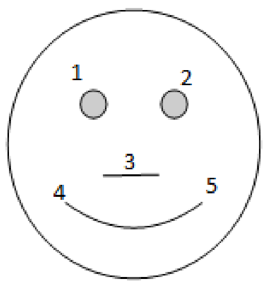

Figure 1: Facial Features used

But, the problem here is, an image has high dimensionality space (each image is a point in a space of dimension d = MN, M and N being image size) as each pixel is considered as a variable of an image. So, we can reduce the dimensionality by using PCA to simplify recognition problem, which can be considered as the core concept. The given set of the localized data is as follows:

x	*y*
13	20
50	20
34	34
16	50
48	50

Training

Training the face detector requires the following steps (summary):

1. Calculate the mean of the input face images
2. Subtract the mean from the input images to obtain the mean-shifted images
3. Calculate the eigenvectors and eigenvalues of the mean-shifted images
4. Order the eigenvectors by their corresponding eigenvalues, in decreasing order
5. Retain only the eigenvectors with the largest eigenvalues (the *principal components*)
6. Project the mean-shifted images into the eigenspace using the retained eigenvectors

Step by step:

1. There are M images in the training set.

2. There are K most significant Eigenfaces using which we can satisfactorily approximate a face. Needless to say K < M.

3. All images are $N \times N$ matrices, which can be represented as $N^2 \times 1$dimensional vectors. The same logic would apply to images that are not of equal length and breadths. To take an example: An image of size 112 x 112 can be represented as a vector of dimension 12544 or simply as a point in a 12544-dimensional space.

Algorithm for Finding Eigenfaces:

- Obtain M training images I_1, I_2 … I_M, it is very important that the images are centered.

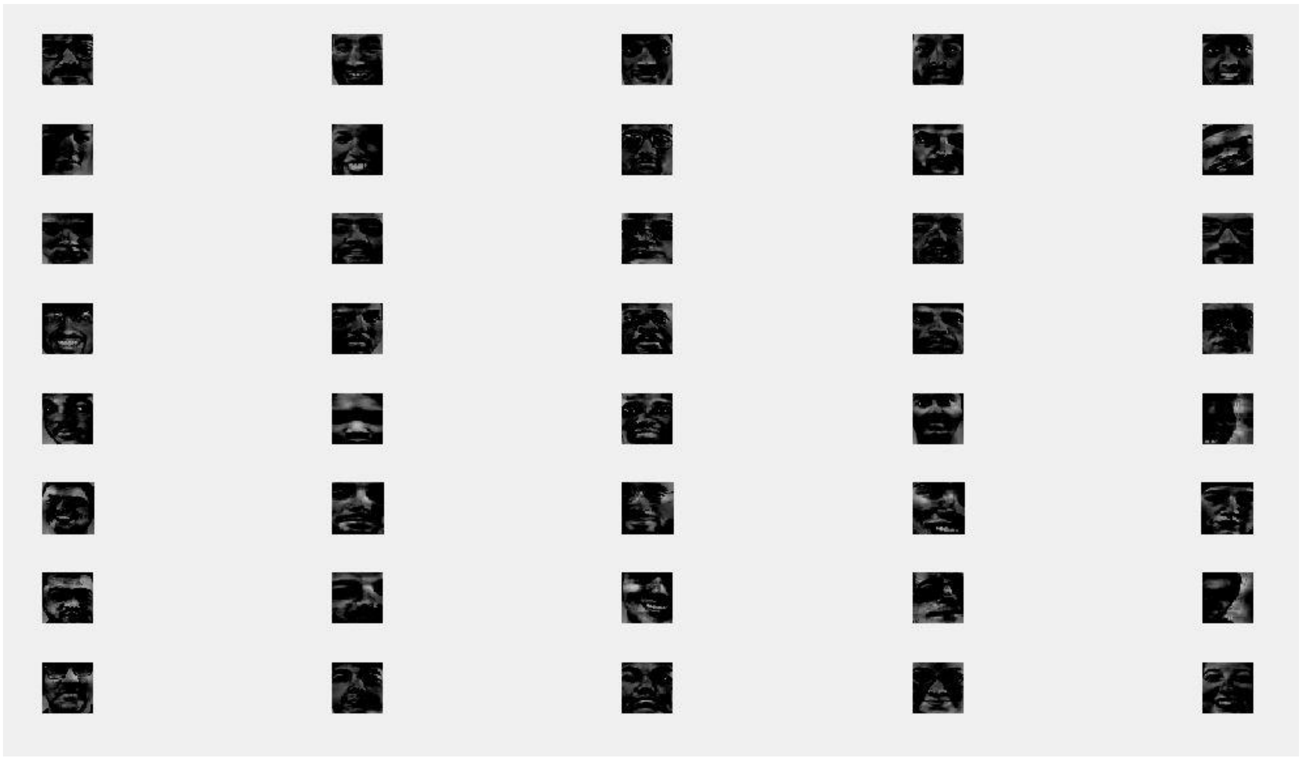
Figure 2: Images of class of 40 Students

- Represent each image I_i as a vector Γ_i as discussed above.

$$I_i = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}_{N \times N} \xrightarrow{\text{concatenation}} \begin{bmatrix} a_{11} \\ \vdots \\ a_{1N} \\ \vdots \\ a_{2N} \\ \vdots \\ a_{NN} \end{bmatrix}_{N^2 \times 1} = \Gamma_i$$

- Find the average face vector Ψ.

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i$$

- Subtract the mean face from each face vector Γ_i to get a set of vectors Φ_i. The purpose of subtracting the mean image from each image vector is to be left with only the distinguishing features from each face and “removing” in a way information that is common.

$$\Phi_i = \Gamma_i - \Psi$$

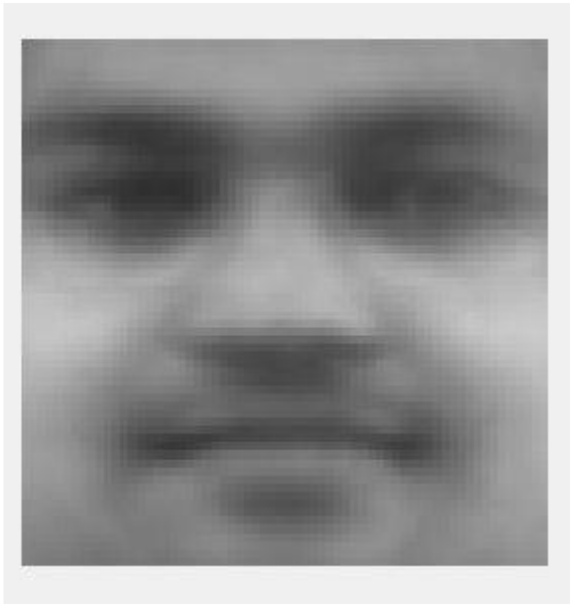

Figure 3: Average image of the training set

- Find the Covariance matrix C:

$$C = AA^T, \text{ where } A = [\Phi_1, \Phi_2 \ldots \Phi_M]$$

Covariance matrix has simply been made by putting one modified image vector obtained in one column each.

- Also C is a $N^2 \times N^2$ matrix and A is a $N^2 \times M$ matrix.
- We now need to calculate the Eigenvectors u_i of C, However note that C is a $N^2 \times N^2$ matrix and it would return N^2 Eigenvectors each being N^2 dimensional. For an image this number is HUGE. The computations required would easily make your system run out of memory. How do we get around this problem?
- Instead of the Matrix AA^T consider the matrix A^TA. Remember A is a $N^2 \times M$ matrix, thus A^TA is a $M \times M$ matrix. If we find the Eigenvectors of this matrix, it would return M Eigenvectors, each of Dimension $M \times 1$, let's call these Eigenvectors v_i.
- Now from some properties of matrices, it follows that: $u_i = Av_i$. We have found out v_i earlier. This implies that using v_i we can calculate the M largest Eigenvectors of AA^T. Remember that $M \ll N^2$ as M is simply the number of training images.
- Find the best M Eigenvectors of $C = AA^T$ by using the relation discussed above. That is: $u_i = Av_i$. Also keep in mind that $\|u_i\| = 1$.
- Select the best K Eigenvectors, the **selection of these Eigenvectors is done heuristically**.

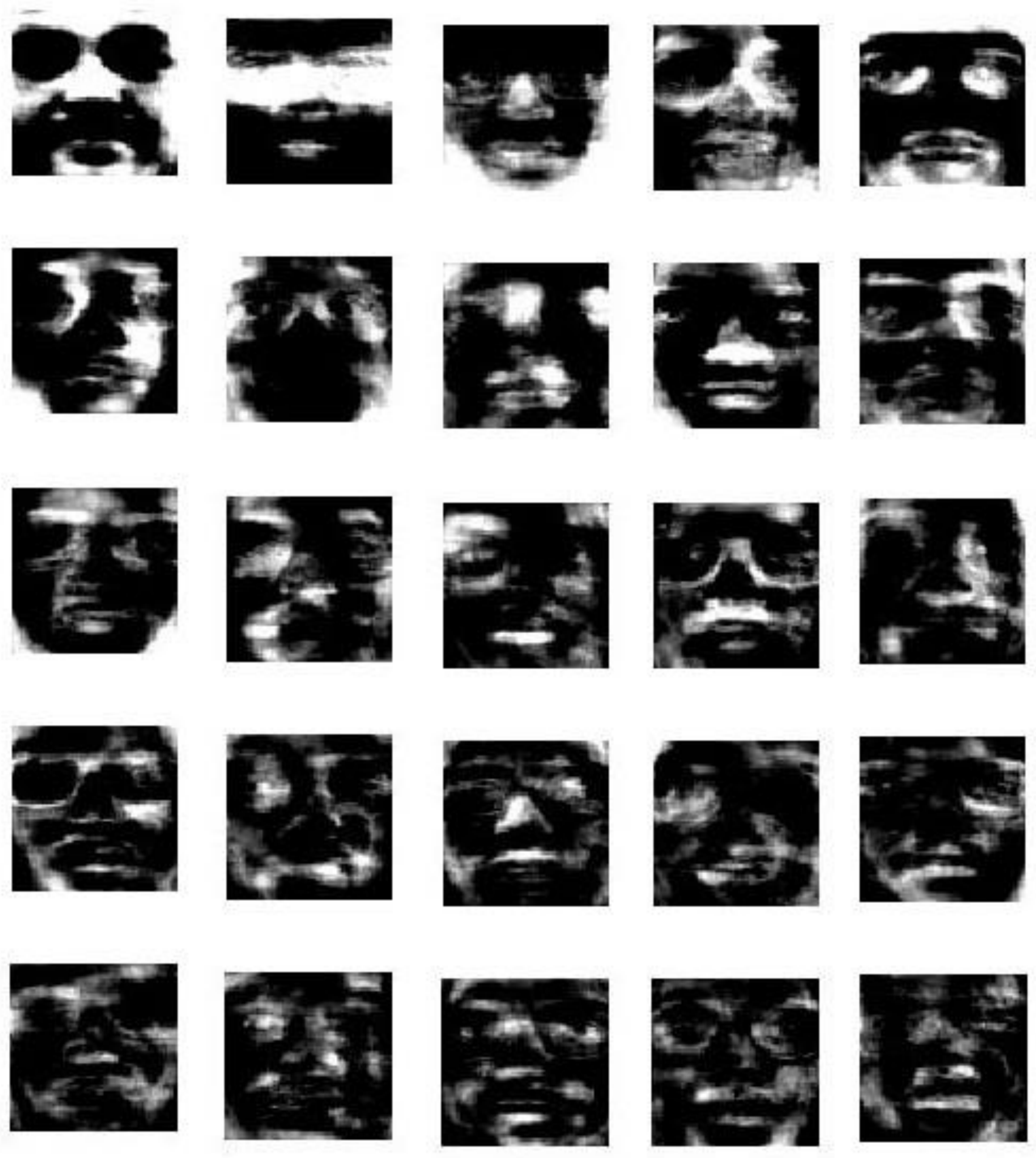

Figure 4: Few eigenfaces

Finding Weights:

The Eigenvectors found at the end of the previous section, u_i when converted to a matrix in a process that is reverse to that in STEP 2, have a face like appearance. **Since** these are Eigenvectors and have a face like appearance, they are called Eigenfaces. Sometimes, they are also called as **Ghost Images** because of their weird appearance.

Now each face in the training set (minus the mean), Φ_i can be represented as a linear combination of these Eigenvectors u_i:

$$\Phi_i = \sum_{j=1}^{K} w_j u_j$$ m, where u_j 's are Eigenfaces.

These weights can be calculated as :

$$w_j = u_j^T \Phi_i.$$

Each normalized training image is represented in this basis as a vector.

$$\Omega_i = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

where i = 1,2… M. This means we must calculate such a vector corresponding to every image in the training set and store them as templates.

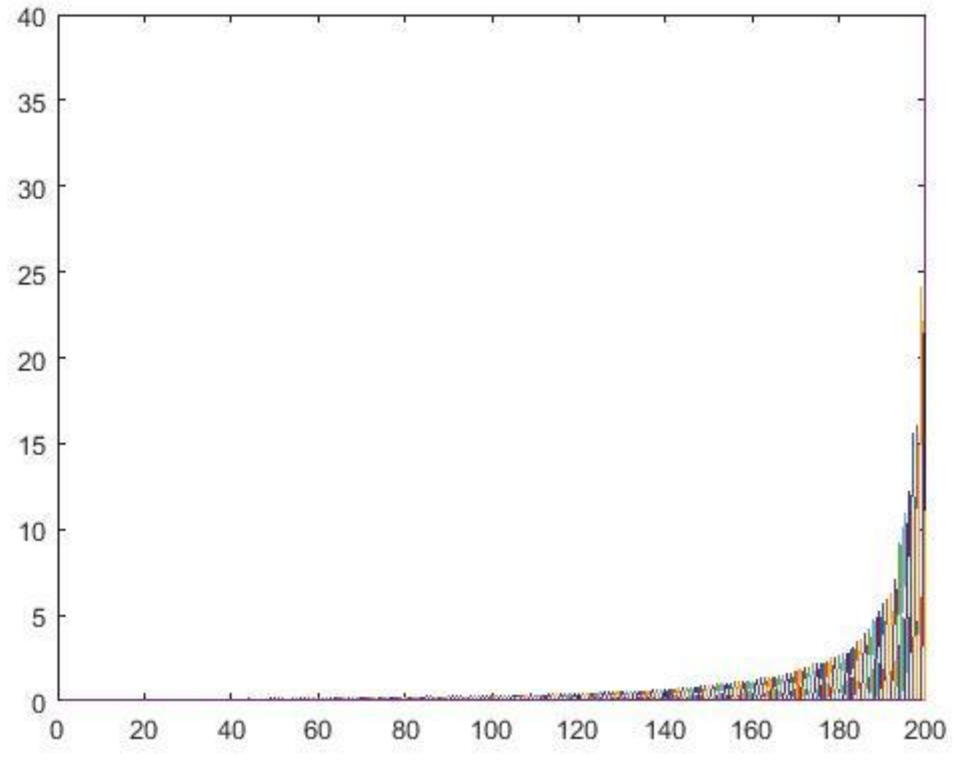

Figure 5: Eigenvalues of set of training images

Recognition Task:

Now consider we have found out the Eigenfaces for the training images, their associated weights after selecting a set of most relevant Eigenfaces and have stored these vectors corresponding to each training image.

If an unknown probe face Γ is to be recognized, then:

- We normalize the incoming probe Γ as $\Phi = \Gamma - \Psi$.
- We then project this normalized probe onto the Eigenspace (the collection of Eigenvectors/faces) and find out the weights.

$$w_i = u_i^T \Phi.$$

- The normalized probe Φ can then simply be represented as:

$$\Omega = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix}$$

After the feature vector (weight vector) for the probe has been found out, we simply need to classify it. For the classification task we could simply use some distance measures or use some classifier like Support Vector Machines (something that I would cover in an upcoming post). In case we use distance measures, classification is done as:

Find $e_r = min \left\|\Omega - \Omega_i\right\|$. This means we take the weight vector of the probe we have just found out and find its distance with the weight vectors associated with each of the training image.

And if $e_r < \Theta$, where Θ is a threshold chosen heuristically, then we can say that the probe image is recognized as the image with which it gives the lowest score.

If however $e_r > \Theta$ then the probe does not belong to the database. I will come to the point on how the threshold should be chosen.

For distance measures the most commonly used measure is the Euclidean Distance.

Distance Measures:

Euclidean Distance: The Euclidean Distance is probably the most widely used distance metric. It is a special case of a general class of norms and is given as:

$$\left\|x - y\right\|_e = \sqrt{\left|x_i - y_i\right|^2}$$

Conclusion:

We have applied various techniques to improve the algorithms like histogram equalization for better image understanding, converting color image to the gray scale image, detecting the face and converting the face to a smaller matrix size of 64 x 64 to fast process the algorithms. Using these techniques have drastically helped us improve the performance of the algorithm. We can achieve the accuracy up to the level of 72.5% in the first case match.

Bibliography:

- http://jmcspot.com/Eigenface/
- http://www.vision.jhu.edu/teaching/vision08/Handouts/case_study_pca1.pdf
- https://onionesquereality.wordpress.com/2009/02/11/face-recognition-using-eigenfaces-and-distance-classifiers-a-tutorial/
- https://blog.cordiner.net/2010/12/02/eigenfaces-face-recognition-MATLAB/
- https://courses.cs.washington.edu/courses/cse455/10wi/projects/p2/index.html
- http://www.cs.cmu.edu/~pmuthuku/mlsp_page/assignments/assignment2_hints.html
- https://stats.stackexchange.com/questions/130721/what-norm-of-the-reconstruction-error-is-minimized-by-the-low-rank-approximation
- https://fr.mathworks.com/MATLABcentral/fileexchange/17032-pca-based-face-recognition-system
- https://fr.mathworks.com/MATLABcentral/fileexchange/48479-face-recognition-using-eigenfaces
- http://openbio.sourceforge.net/resources/eigenfaces/eigenfaces-html/facesOptions.html
- https://www.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf
- http://blog.manfredas.com/eigenfaces-tutorial/

MATLAB Code

```
clc
close all
clear

%Delete Previously saved variables
delete('Vector.mat');
delete('Names.mat');
delete('averageimage.mat');

%Read Images from database path and save them
DatabasePath = './faces_features/';
normalize_path = './normalization/';
average_path = './average/';
Length = 320;
Width = 240;

%Read and save coordinate from the files:
%save the coordinate in in binary and txt format with the name
%coordinate.mat and coordinate.txt and dimension as number of images x 10
imagefiles = dir(strcat(DatabasePath,'*.jpg'));
nfiles = length(imagefiles);
readcoordinates(DatabasePath);
```

```
%Affine transformation of the images and normalization:

load coordinate.mat;
Normalization(DatabasePath, normalize_path, coordinate,
size(coordinate,1),Length, Width);
```

```
%average of all the image:(Preparing Ghost Image)
imagefiles = dir(strcat(normalize_path,'*.jpg'));
%nfiles = length(imagefiles);   % Number of files found
S=[];
Name = [];
m=0;
No_Training = 5; %Number of Training Image for each face
averageimage = zeros(64,64);
```

```
for index = 1:No_Training:nfiles

    for i = 0:No_Training-1
        img_idx = index + i;
        if img_idx <= nfiles
            [currentfilename, currentimage] = readimages(normalize_path, 64,
64, img_idx); %reading image at every level
            temp_currentfilename = split(currentfilename,string(sum(i+1)));
            currentfilename1 = string(temp_currentfilename(1));
            currentimage = histeq(currentimage);
            averageimage = averageimage + currentimage;
%             m = mean(mean(averageimage));
            temp = reshape(currentimage, 1, 64*64);
            S = [S; temp];
            Name = [Name; string(currentfilename1)];
        end
    end

    savefilename = strcat(average_path,currentfilename,'.jpg');
    imwrite(averageimage,savefilename,'jpeg');

end
averageimage = averageimage/nfiles;
averageimage = reshape(averageimage, 1, 64*64);
S = S-averageimage;
save 'Vector.mat' S;
save 'Names.mat' Name;
save 'averageimage.mat' averageimage;
```

Published with MATLAB® R2017a

```
%Input Parameters:
%DatabasePath:        Folder Path from where images have to be read.
%l,b:                 Length and Width of image
%im_number:           Read the desired image with a particular index no.

%Output Parameters:
%Currentfilename:     Returns the name of the file
%currentimage:        Returns the image of the specified index number.
```

```
function [currentfilename, currentimage] =
readimages(DatabasePath,l,b,im_number)

imagefiles = dir(strcat(DatabasePath,'*.jpg'));
nfiles = length(imagefiles);    % Number of files found

S = [];

for index = 1:nfiles

    %averageimage = zeros(l,b);
    if (index==im_number)
        index
        % read images from directory
        currentfilename = imagefiles(index).name;
        currentfilename2 = strcat(DatabasePath,currentfilename);
        currentimage = im2double(imread(currentfilename2));
        if ndims(currentimage)>2 %To check the gray level image
            currentimage = rgb2gray(im2double(imread(currentfilename2)));
        end
        [l1, b1] = size(currentimage);
        if l1~=l || b1~=b
            currentimage = imresize(currentimage, [l,b]);
        end
%             temp = reshape(currentimage, 1, l*b);
%             averageimage = averageimage + .2*currentimage;
%             m = mean(temp);
%             S = [S; temp-m];
    end

    %save file to disk
%     savefilename = strcat(savepath,currentfilename);
%     imwrite((averageimage),savefilename,'jpeg');
end

end
```

Published with MATLAB® R2017a

```
%DatabasePath:      Specify the destination from where the database of the
%                   coordinate has to be made.
```

```
%Returns:
%Coordinate:        Coordinate files in .mat format
%nfiles:            Number of files read
```

```
function [coordinate, nfiles] = readcoordinates(DatabasePath)
```

```
%Read and save coordinate from the files:
%save the coordinate in in binary and txt format with the name
%coordinate.mat and coordinate.txt and dimension as number of images x 10

imagefiles = dir(strcat(DatabasePath,'*.txt'));
nfiles = length(imagefiles);   % Number of files found
coordinate = zeros(nfiles,10);
for index = 1:5:nfiles %as there are 5 test images in each case
%     averageimage = zeros(l,b);

    for i = 0:4
        img_idx = index + i;
        if img_idx <= nfiles
            % read files from directory
            %fprintf('Hello')
            currentfilename = imagefiles(img_idx).name;
            currentfilename = strcat(DatabasePath,currentfilename);
            coordinate(img_idx,1:10) = textread(currentfilename, '%f',10);
        end
    end
end

%save file to disk
save('coordinate.txt','coordinate','-ascii');
type('coordinate.txt');
%in binary format
save 'coordinate.mat' coordinate;
```

```
end
```

Published with MATLAB® R2017a

```
%DataBasePath:      From where the input image has to be read
%NormalizePath:     Specify the Destination where you have to save the
%                   normalized image
%Coordinate:        Coordinate Matrix obtained from ReadCoordinate Files.
Load the
%                   file before calling this Normalization File.
```

```
%nfiles:             Number of files in the destination path for loop
iteration and
%                    reading all the files.
%Length and Width:   Size of the input image
%Returns:            save the normalized images in the specified folder.
```

```
function Normalization(DatabasePath, normalize_path, coordinate, nfiles,
Length, Width)

FixedPoints = [13,20; 50,20; 34,34; 16,50; 48,50];

for i = 1:nfiles
    A = FindTransformation(coordinate(i,:), FixedPoints);
    A_tform = maketform('affine',A.T);
    [currentfilename, currentimage] = readimages(DatabasePath, Length, Width,
i); %reading image at every level
    Im_Updated = imtransform(currentimage,A_tform,'XData',[1 64],'YData',[1
64]);
    %Im_Updated = imwarp(currentimage,imref2d(size(currentimage),[1, 64],[1,
64]), A);
    savefilename = strcat(normalize_path,currentfilename);
    imwrite(Im_Updated,savefilename,'jpeg');
end

end
```

Published with MATLAB® R2017a

```
%Returns the Affine Transform of the given set of coordinate to the
%specified FixedPoint set of coordinate.
%M is of the size 1x10 which is converted in (x,y) shape again.
```

```
function A = FindTransformation(M,FixedPoints)
FloatingPoint = reshape(M,[2,5]);
A = fitgeotrans (FloatingPoint', FixedPoints, 'affine' );
end
```

Published with MATLAB® R2017a

```
clc
clear
%Reading the image:
```

```matlab
load Vector.mat
load Names.mat
load averageimage
A = S';
%Find the covariance matrix of the trainig matrix.
C = (S*S')/size(S,1);

%Eigen value of the covariance matrix.
[V, D] = eig(C);

%Arranging the eigenvalues
L_eig_vec = [];
for i = 1 : size(V,2)
    if( D(i,i)>0 )
        L_eig_vec = [L_eig_vec V(:,i)];
    end
end

%Finding the eigen faces of the image:
Eigenfaces = zeros(size(A,1),size(L_eig_vec,2));
for i = 1:size(L_eig_vec,2)
    Eigenfaces(:,i) = A * L_eig_vec(:,i);
    Eigenfaces(:,i) = Eigenfaces(:,i)/norm(Eigenfaces(:,i));
end
%Projected image according to each eigenface
ProjectedImages = [];
Train_Number = size(Eigenfaces,2);
for i = 1 : Train_Number
    temp = Eigenfaces'*A(:,i); % Projection of centered images into facespace
    ProjectedImages = [ProjectedImages temp];
end

%testing phase:
test_path = './testimage/'; %Location of the test folder
test_normalize_path = './testimage/normalize/';
[coordinate_test, nfiles] = readcoordinates(test_path);
save './testimage/coordinate_test.mat' coordinate_test

Normalization(test_path, test_normalize_path, coordinate_test,
size(coordinate_test,1),320, 240);
count = 0;
o = [];
p = [];
```

```
for k = 1:nfiles
    [currentfilename, currentimage_test] = readimages(test_normalize_path,
64, 64, k); %reading image at every level
    currentimage_test = histeq(currentimage_test);
    temp = reshape(currentimage_test, 1, 64*64);
    S_test = temp-averageimage;
    ProjectedTestImage = Eigenfaces'*S_test';

    %Finding the Eucladian distance from each vector and finding the
    %minimum.
    Euc_dist = [];
    for i = 1 : Train_Number
        q = ProjectedImages(:,i);
        temp = ( norm( ProjectedTestImage - q ) )^2;
        Euc_dist = [Euc_dist temp];
    end

    [Euc_dist_min , Recognized_index] = min(Euc_dist);
    OutputName = Name(Recognized_index);

    Out = split(currentfilename,'5');
    out1 = string(Out(1));

    %Displaying the images not detected correctly
    if out1 == OutputName
        o = [o, out1];
    else
        count = count + 1;
        p = [p, out1];
    end
end
o %correctly Detected Output
p %Undetected Output
```

```
accuracy = (1-(count/nfiles))*100
```

```
o =

  1x29 string array
  Columns 1 through 6

    "Ali"    "Alyafi_"    "Anaghan"    "Asraf Ali_"    "Birhanu"
"Boudissa"

  Columns 7 through 12

    "Doiriel"    "Dudhagara"    "Fahad"    "Flavien"    "Genke"    "Hasan"

  Columns 13 through 18

    "Herrera_"    "KB"    "Malav"    "Mohammed Abdul"    "Ngige"    "Pandey"

  Columns 19 through 24

    "Rakic"    "Reimmer"    "Sonizara"    "Sulaiman_"    "Yawson"
"antoine"

  Columns 25 through 29

    "shah"    "toshpulatov"    "tushar"    "valencia"    "zotova"


p =

  1×11 string array

  Columns 1 through 6

    "Kozynets"    "Vaishnav"    "benali"    "berada"    "brianna"    "gulnur"

  Columns 7 through 11

    "khan"    "lav"    "ochoa"    "rockenbach"    "roger"


accuracy =

   72.5000
```

Published with MATLAB® R2017a