PCA Face Recognition

Applied Mathematics Project

By :
Mohammad-Rami Koujan
Mohammad Alkhatib

Overview

The objective of this project is to apply PCA (Principal Component Analysis) in Face Recognition application. Face recognition has been one of the famous field in Computer Vision. Face Recognition application will extract a set of a particular pixel in the image and try to find the best match with a database of images. Before we can do PCA, we need to detect a face in the image and make the normalized image from that. In this project, detecting the face is done manually. By manually defined the features location in the images, we can proceed to the next step. Normalization will be based on the affine transformation. After these steps, we can proceed to apply PCA for face recognition.

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

A set of eigenfaces can be generated by performing a mathematical process called principal component analysis (PCA) on a large set of images depicting different human faces.

Goals

In this project, we implemented a face recognition system by using principal component analysis, which is known as PCA. PCA method provide a mathematical way to reduce the dimension of problem.

Since the most elements of a facial image are highly correlated, it is better to extract a set of interesting and discriminative feature of a facial image. Mathematically speaking, we transform the correlated data to independent data. To implement the transform, we employed some linear algebra method such as SVD (Chapter 3). The main idea is to obtain Eigenfaces that every face can be regard as a linear combination of these eigenfaces (Chapter 4). Then the face recognition problem convert to a mathematical problem: what is the linear combination of a face? In other words, it simplify a problem from 2D to 1D.

1 Introduction

1.1 Face Recognition

Face recognition (FR) has always remained a major focus because we identify people through their faces, which we can say it as the primary method. Actually, we can ask something's like this, what is Face Recognition? Identifying a person by his face from previous knowledge. Why do we use Face Recognition? To identify or determine people's identity. Obviously, we can say that there are many different identification technologies like fingerprints, user id and password, retina and iris recognition etc. so why do we use only this recognition when we have others? Because when we use most of the other technologies like the above given ones you should stop and press the fingers or enter the password or show your eye but FR can be done while you are walking in or out i.e. there is no need to stop and show your face. Where do we use? The most useful applications are crowd surveillance, video content indexing, personal identification (ex. driver's license), entrance security, etc.

1.2 Principal Component Analysis

Principal Components Analysis (PCA) is a practical and standard statistical tool in modern data analysis that has found application in different areas such as face recognition, image compression and neuroscience. It has been called one of the most precious results from applied linear algebra. The Principal Component Analysis (PCA) is one of the most successful techniques that have been used in image recognition and compression. PCA can be used for many reasons such as Simplification, data Reduction, modeling, outlier detection, variable selection, classification, prediction and so on. But in here the purpose of PCA is to reduce the large dimensionality of the data space (observed variables) to the smaller intrinsic dimensionality of feature space (independent variables), which is needed to describe the data economically.

1.3 Singular Value Decomposition

The Singular Value Decomposition (SVD) is a widely used technique to decompose a matrix into several component matrices, exposing many of the useful and interesting properties of the original matrix. The decomposition of a matrix is often called Factorization.

In SVD, we decompose a real matrix A € R m×n(m >n) can be written as the product of three matrices as follows:

$$A = U\,S\,V^T$$

The two matrices U €R m×n and V € R n×n are orthonormal. The diagonal elements of S € R n×n are the singular values. The column vectors of U and V are the corresponding singular vectors of each singular value.

Face recognition is one example where principal component analysis has been extensively used. The choice of the Singular Value Decomposition (SVD) is very important due to its vast use in the Principal Component Analysis and because it largely defines its performance.

2.1 Normalization

Normalization actually is a projection process. Because we detected the face images with different angle, we hope that we can use a same coordinate system for different angle faces. This coordinate system is a 64 × 64 window which we predetermined five positions as facial features' location. These five locations is F. Once we have F, we can obtain affine transformation coefficients for each image .

A and b are affine transformation matrix. Where A and b are given by:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

How to compute A and b

For each feature, the affine transformation is given by:

$$f_i^P = Af_i + b$$

Where fi is the feature location in original image and fpi is the position in normalization image.

Since we already have fi and fpi is also predetermined, let f =[x,y], then the transformation can be written into:

$$x_i^P = a_{11}x_i + a_{12}y_i + b_1$$
$$y_i^P = a_{21}x_i + a_{22}y_i + b_2$$

We apply it on five features, the transformation is given by:

$$\begin{bmatrix} x1_i & y1_i & 1 \\ x2_i & y2_i & 1 \\ x3_i & y3_i & 1 \\ x4_i & y4_i & 1 \\ x5_i & y5_i & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ b_1 & b_2 \end{bmatrix} = \begin{bmatrix} x1_i^P & y1_i^P \\ x2_i^P & y2_i^P \\ x3_i^P & y3_i^P \\ x4_i^P & y4_i^P \\ x5_i^P & y5_i^P \end{bmatrix}$$

Until now, the problem convert to solve Ax = b in linear algebra. Since we got 10 equations for 6 unknowns, we employ SVD (function pinv() in matlab) to solve it.

How to find F

It is very important to get an appropriate F. We calculate F by following steps:

• Step 1: We initialize F with first image's feature to find average of all.

• Step 2: Finding transformation that project F to predefined position in 64x64 window. Then we can find this affine transformation as A and b matrices that has 6 parameters, and we apply this transformation to F.

• Step 3: For each image feature Fi, we computing the A and b matrices for affine transformation that projecting F to Fi.

• Step 4: We change F to averaged projected feature locations that we calculated previous step.

• Step 5: If error between Ft - Ft−1 is less than our threshold, we go back to step 2.



3 Principal Component Analysis

We used PCA to transform our correlated data set into smaller uncorrelated data set. So PCA helped us on reducing the large dimensions of data set into smaller dimensions of it. After finding F, we followed these steps:

1- We converted our 64 by 64 normalized images Xi into row vector sized 1x4096 and constructed matrix Dpxd(p = number of images, d = 64x64) that each row corresponds to Xi.

2- Then we defined k between 50 and 100 to kth largest eigenvalues.

3- Before computing the PCA, we computed the covariance matrix of D as follows:

$$\Sigma = \frac{1}{p-1} D^T D$$

Then we used matlab's "svd()" function to get eigenvectors from matrix. This process in our training is taking the most time. We put k eigenvectors corresponding to biggest k eigenvalues to Φ□.

We calculated PCA space by;

$$\phi_i = X_i . \Phi$$

PCA space represent the linear combination of eigenfaces of train images. Which means, for any train image, we just need know what is its linear combination of eigenfaces. After all, now we can store all our training images in the PCA space and easily search them to find closest matches.

4 Recognition

To find an image matching in the training set we the project this image into the projection data we created in the above step so we get Φ and then we calculate the Euclidean distance between Φ and the projection matrix. We sort the Euclidian distance we got Euclidian distance values tells us how much similar the test image is in the projection set so we are interested in the least n values of Euclidean distance. Now we can get the index for each least distance and find the corresponding image in the training database. Then we can do the matching images for a particular test image.

Algorithm for Recognition

Step 1: Set the number of Principal components

Step 2: Build a training data matrix D from training images

Step 3: We assign name as the label for each training image and form a labels matrix (

Step 4: Remove the mean and compute covariance matrix (D). Then compute the PCA of D, which gives PCA transformation matrix

Step 5: Feature vectors of all training images using transformation matrix and store them in a matrix

Step 6: Read test images, and for each test image:

– Calculate the feature vector '* using the PCA transformation matrix.

– Compute the distance between '*and all training feature vectors, which are stored in

– Find the best match + (minimum distance between feature vectors)

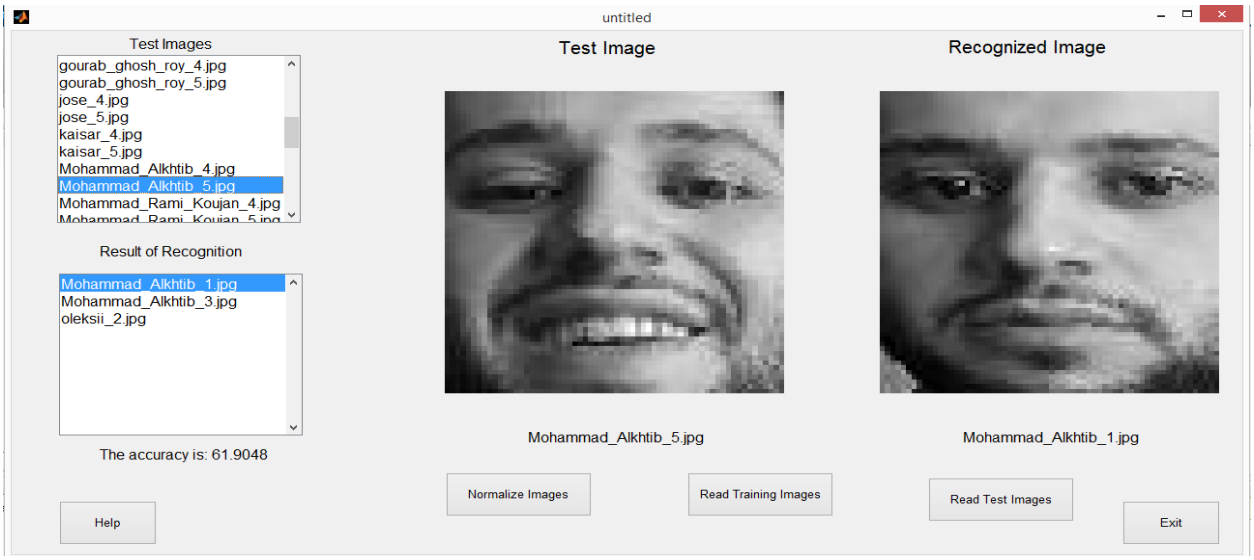
– Check if the label of the image is the same as the label of the mstch. Otherwise, increment an error count, by 1.

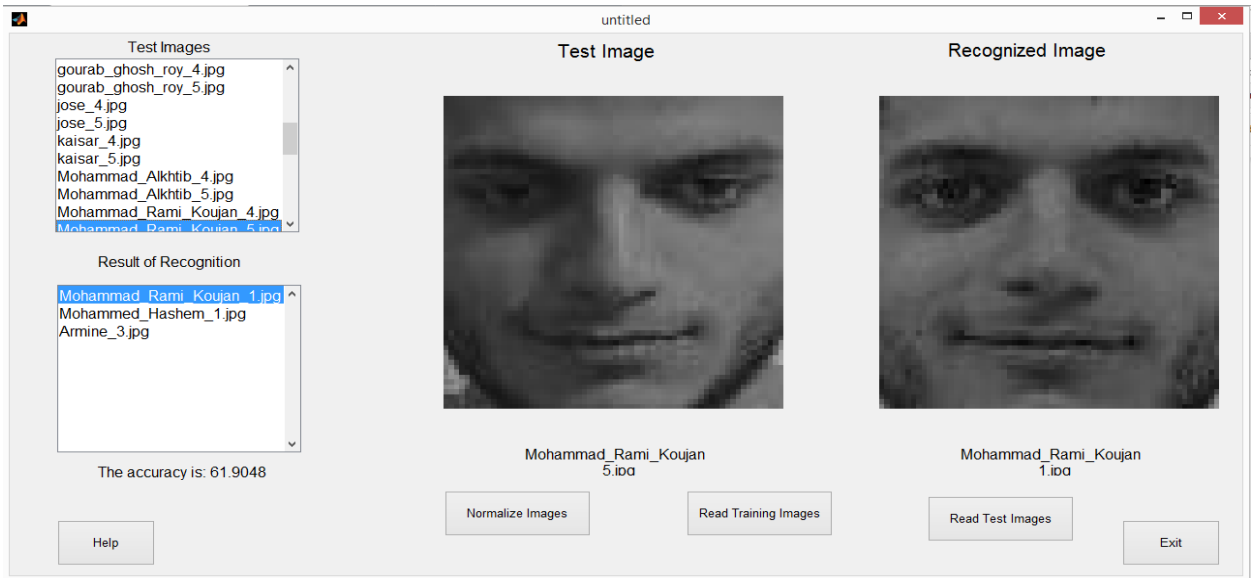
Step 7: We find the Accuracy

$$accuracy = \left(1 - \frac{\epsilon}{total\ number\ of\ test\ images}\right) * 100$$

5 Results

The overall accuracy for our PCA project is 62 per cent. This result is based on taking the first three images for each subject as a train set and the other two as the test set. Therefore, what we have eventually in the train test is 63 images and the test set is 42 images. For each image, we took the first three matches, since there are only three images for each subject in the training set, and displayed those images in a separate window, which will make it easier to choose between them.





6 Conclusion

In our project we have implemented a PCA (Principal Component Analysis) based project in order to achieve face recognition application.

The difficulties that we have faced so far are as follow:

1- In some of the images under test, the five used features are not clear in all the five images of each subject.
2- Some of the images have different dimensions (240*320), which urged us to resize them again.
3- The algorithm is sensitive to the accuracy of the given features locations. For example, some of the locations were not very accurate, which affected first of all the normalization phase and consequently the final accuracy of the matching.

To sum up:

1- PCA is an effective algorithm to reduce the correlation between different variables which make it perfect for Face Recognition Application.
2- As more Features we use as more accurate result we got.
3- The more the features locations accurate the more accuracy we got.

Appendix

Matlab Code

```
function [ Img ] = Normalization(direc)
mkdir(direc,'Normalized_images');
list_jpg=dir(strcat(direc,'*.jpg'));
Fave=[13 20;50 20;34 34;16 50;48 50];
h=waitbar(0,'please wait...');
set(h, 'WindowStyle','modal', 'CloseRequestFcn','');
for counter=1:length(list_jpg)
    h=waitbar(counter/length(list_jpg),h);
    list_jpg(counter).name
    img=imread(strcat(direc,'\',list_jpg(counter).name));
    img = rgb2gray(img);
    Img = uint8(zeros(64, 64));
    file=strrep(list_jpg(counter).name,'.jpg','.txt');
    Fi=textread(strcat(direc,'\',file));
    [A,b] = calculate_AB(Fi, Fave);
      for i=1:64
          for j=1:64
              % solve the equation xy64 = A*xy + b to obtain the pixel
              %positions in the bigger image
              xy = (pinv(A)*( [ i; j ] - b ));

              % extract the x and y coordinate
              x240 = int32(xy(1,:));
              y320 = int32(xy(2,:));

              % Although very rare, these values can fall down to negative values. So if it
              % happens, just make it zero. One pixel won't make a huge difference.
              if(x240 <= 0)
                  x240 = 1;
              end

              if(y320 <= 0)
                  y320 = 1;
              end

              if(x240 > 240)
                  x240 = 240;
              end

              if(y320 >320)
                  y320 = 320;
              end
```

```
                % copy the value of the pixel in the bigger image to the
                % normalized image
                Img(i,j) = uint8(img(y320,x240));
            end

        end

        %Img=Img';
        %[pathstr,name,ext] = fileparts(direc);
        imDir=strcat(direc,'\Normalized_images\',list_jpg(counter).name)
        Img=imrotate(Img,-90);
        imwrite(Img,imDir);
    end

    delete(h);
end
```

```
function [A,b] = calculate_AB(Fi, Fpre)
    AB = pinv([Fi, [1;1;1;1;1]])*Fpre;
    A = AB(1:2,:)';
    b = AB(3,:)';
end
```

```
function [cm]=concat(m)
% function to concatenate images read inside the PCA algorithm

m=m';
m=m(:);
cm=m';

end
```

```
function [phi_i,eigenvec,label_mat]=train(FILENAME, PATHNAME)

h=waitbar(0,'please wait...');
set(h, 'WindowStyle','modal', 'CloseRequestFcn','');
FILENAME=cellstr(FILENAME);
p=length(FILENAME);   %number of training images
s=size(imread(char(strcat(PATHNAME,FILENAME(1)) ))); %size of each training
D=zeros(p,s(1)*s(2));
for i=1:length(FILENAME)
    h=waitbar(i/length(FILENAME),h);
    img=imread(char(strcat(PATHNAME,FILENAME(i))));
    img=concat(img);
    D(i,:)=img;
end
    D_par=D-repmat(mean(D),[p,1]);
    sigma_prime=(1/(p-1))*(D_par*D_par');  % covariance matrix
    [vec,val]=eig(sigma_prime);
    eigenvec=D_par'*vec;      % O|       d*p

    for i=1:p
      eigenvec(:,i)=eigenvec(:,i)/norm(eigenvec(:,i));
      %normalization of the eigenvectors in order to be orhonormal
    end
    phi_i=D*eigenvec;  %feature vectors
    label_mat=FILENAME;
    delete(h);
end
```