Université de Bourgogne
Centre Universitaire Condorcet
VIBOT Computer Vision and Robotics
Applied Maths

Face Recognition Project

Student:
Dragutin Vujovic
Flávia Dias Casagrande
Professor: Désiré Sidibé

Le Creusot
January, 2015

1 Objective

This report presents the theory and implementation of the face recognition program implemented. Tools such as Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) are employed for the development of the application.

2 Basic Definitions

2.1 Singular Value Decomposition (SVD)

Consider a matrix $\mathbf{A}$ $m \times n$. Its singular value decomposition is given by Equation

$$A = U\Sigma V^T \tag{1}$$

Where $\mathbf{U}$ is orthonormal, $m \times m$; Σ is diagonal, $m \times n$; and $\mathbf{V}$ is orthonormal, $n \times n$;

Which means SVD decomposes $\mathbf{A}$ into three transformations: a rotation$\mathbf{V}$, a scaling Σ along the coordinate axes, and a final rotation $\mathbf{U}$.

In addition, the diagonal entries σ_i, of Σ are non-negative and called the singular values of A, ordered in a way that the largest singular value is placed in $(1, 1)$ and decreases in the diagonal direction.

2.2 Principal Component Analysis (PCA)

PCA is a mathematical method that allows the re-expression of a large set of data in another basis, which is a linear combination of the original one, but optimized [1].

Let $\mathbf{X}$ be a matrix $m \times n$ that represents a set of data, where m corresponds to the variables and n to the samples. We perform then a linear transformation of $\mathbf{X}$, as in Equation (2).

$$Y = PX = \begin{pmatrix} p_1x_1 & p_1x_2 & \dots & p_1x_n \\ p_2x_1 & p_2x_2 & \dots & p_2x_n \\ \vdots & \vdots & \ddots & \vdots \\ p_mx_1 & p_2x_m & \dots & p_mx_n \end{pmatrix} \tag{2}$$

Which means $\mathbf{X}$ is projected in $\mathbf{P}$, a $m \times m$ matrix. This is, $\mathbf{P}$ is the new basis for representing $\mathbf{X}$ and its rows will be the principal component directions.

These principal components should be independent, in order to have the best optimization. With this goal, matrix $\mathbf{P}$ tries to de-correlate the original data.

Equation (3) presents the variance of a random variable Z with mean μ.

$$\sigma_Z^2 = E[(Z - \mu)^2] \tag{3}$$

Consider now a vector r of n discrete measurements, with mean μ_r. By subtracting the mean from each of the measurements, we obtain a translated set of measurements r, that has zero mean. The variance of these measurements is given by Equation (4).

$$\sigma_r^2 = \frac{1}{n} r r^T \tag{4}$$

We define then the convariance matrix Σ as Equation (5), since variance is a special case of covariance, when two variables are identical.

$$\Sigma = \frac{1}{n-1} r r^T \tag{5}$$

Where the i^{th} entry of the diagonal of Σ is the variance of the i^{th} variable and the $(ij)^{th}$ entry of Σ with $i \neq j$ is the covariance between the i^{th} and j^{th} variables.

Therefore, the matrix Σ is symmetrical and can be then diagonalized [pca2].

Let $\mu_1 \geq \mu_2 \geq ...mu_m \geq 0$ be the *eigenvalues* of Σ and $\vec{v_1}, ..., \vec{v_m}$ the corresponding orthonormal *eigenvectors*. The *eigenvectors* are the principal components of the data set.

Let T be the total variance of the data set. It also corresponds then to the sum of the *eigenvalues*. It means that the direction given by $\vec{v_1}$ (first principal direction) corresponds to an amount μ_1 of the total variance, and so on for the other *eigenvectors*. The first principal components points to the most significant direction of the data set, which will correspond to the new data set, with no or much less redundancy.

3 Problem

As for what was explained in Section 2, it is possible to understand why PCA can be used for many computer vision applications. It is evident for instance its use for image compression and face recognition.

The goal of the project here presented is to develop a code in Matlab performing face recognition.

The idea is to measure the difference between a new and an original image, along the new axes derived from a PCA analysis, which give us the original images in terms of the differences and similarities between both pictures.

In practice, we are able to leave out some of the less significant eigenvectors, and the recognition would still performs well.

Here we consider an input image I_q and we should compare it with a data set of images $I_1, I_2, ..., I_p$ and find the best match. Each pixel is considered a variable, which means we have a very high dimensional space. By performing PCA we can even reduce the dimensions of the image.

4 Code Implementation

The first step to start the program was a normalization of the pictures given as database.

The normalization consists in, from predetermined values for facial features (Figure 4), use a transformation map. The image was set manually to fit in a 64×64 window.

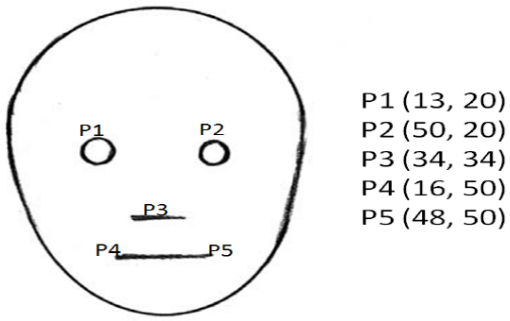

Figura 1: Predetermined facial features.

From the database, the locations of the facial features for each image should be stored. With these locations, the idea is to compute the parameters of an affine transformation that maps them to the predetermined locations.

In this sense, firstly we defined an average vector *Favg* that for the first iteration will get the values from the predetermined features location vector *Fpredetermined*.

Then, there's a *while loop* in which the *Favg* will store the mean values for the features. The stop criteria was set to when the maximum difference between the elements of the previous *Favg* and the new one is less then 0.8.

The affine transformation is defined by six parameters, as seen in Equation (6). In addition, the feature f_i is mapped to the corresponding predetermined location f_i^P, as in Equation (7).

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{6}$$

$$f_i^P = Af_i + b, \quad \text{where} f_i = \begin{pmatrix} x \\ y \end{pmatrix} \tag{7}$$

In order to simplify the calculations, we wrote it in Matlab as in Equation (8) (or variable *Ab* in the code), calculating the five features transformations.

The code for the normalization described is in the function *NormalizeImages*.

$$f_i^P = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \\ x_5 & y_5 & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ b_1 & b_2 \end{pmatrix} \tag{8}$$

Finally, the normalization step result is shown in Figure 4.

```
function [ ] = NormalizeImages( features_dir, faces_dir, test_dir, train_dir,
    doOptimisation )
%Read every feature file and put data in the vector

files = dir(features_dir);
FileName = {};

i = 1;

%locations of features (eyes, tip of nose, tips of mouth)
```

```
Fpredetermined = [13  20;
                  50  20;
                  34  34;
                  16  50;
                  48  50;]';

%get names of all image and store them in FileName array
for x = 1:length(files)
    file = files(x).name;
    if isempty(strfind(file, 'txt')) == 0
        FileName{i} = char(files(x).name);
        i = i + 1;
    end
end

j = 0;
Favg = Fpredetermined;
doIteration = true;
%normalize pictures
while (doIteration == true)

    %temporary variable used to calculate average of Fip at the end of
    %cycle
    FAvgTmp = 0;

    for i =  1:length(FileName)
        tmpFile = FileName{i};
        tmpTxtFile = fullfile(features_dir,tmpFile);
        data = importdata(tmpTxtFile);

        Fi = [data(1) data(6) 1; data(2) data(7) 1; data(3) data(8) 1; data(4)
   data(9) 1; data(5) data(10) 1]';

        %Ab - trasformation that aligns features of image Fi with Fpredetermined
        Ab =  Favg * pinv(Fi);
        %Fip - transformation of image Fi
        Fip = Ab * Fi;

        imgFile = strrep(tmpFile, '.txt', '.jpg');
        ImageResize(faces_dir, test_dir, train_dir, doOptimisation, imgFile, Ab)
    ;

        if (i == 1 && j == 0)
            Favg = Fip;
        end
        FAvgTmp = FAvgTmp + Fip;
    end

    %find difference between current and previous average, if its small
    %stop
    diff = abs((FAvgTmp / length(FileName)) - Favg);
    maxDiff = max(diff(:));
    if maxDiff < 0.8
        doIteration = false;
    end
    Favg = FAvgTmp / length(FileName);
    j = j+1;
```

```
end
clear all;
end
```

Figura 2: Normalized faces.

Besides that, inside the *NormalizeImages* function, there is another function called *ImageResize*. It only puts the image in the picture size required (64×64).

```
function [   ] = ImageResize( faces_dir, test_dir, train_dir, doOptimisation,
    imageName, Ab)

imageSrcName = fullfile(faces_dir,imageName);

 %check if jpg file exist, if not use png file
if exist(imageSrcName, 'file') == 0
    imageSrcName = strrep(imageSrcName, '.jpg', '.png');
end

image = imread(imageSrcName);
image = rgb2gray(image);

if doOptimisation == true
    %adjust contrast to optimise face recognition
    image = imadjust(image);
end

out_image = zeros(64, 64);

A = Ab([1, 3; 2 4]);
b = Ab([5; 6]);

for x = 1:64
    for y = 1:64
        out_pixel = int32(A \ ([x ; y] - b));

        if (out_pixel(1) <= 0 || out_pixel(1) >= 240 || out_pixel(2) >= 320 ||
    out_pixel(2) <= 0)
            pixel_value = 254;
        else
            pixel_value = image(out_pixel(2), out_pixel(1));
        end

        out_image(x, y) = uint8(pixel_value);
    end
end

out_image = mat2gray(out_image');
```

```
if isempty(strfind(imageName, '4')) == 0 || isempty(strfind(imageName, '5')) ==
    0
    %put image in training set
    imageDestinationName = fullfile(test_dir,imageName);
else
    %put image in test set
    imageDestinationName = fullfile(train_dir,imageName);
end

imwrite(out_image, imageDestinationName);

end
```

Once the normalization is done for all the pictures, there is the face recognition step. For this step the pictures were split in two folders. One called *train _ images* which will contain three images for each subject (frontal and two side vies) and the other called *test _ images* which contains the other two pictures.

We consider an image I_i as the vector X_i shown in (9), in the d-dimensional space, where $d = MN$.

$$X_i = [I_i(1,1), I_i(1,2), ..., I_i(M,N)] \tag{9}$$

Where $I_i(x,y)$ in the pixel intensity at position (x,y). We can assume that most of the pixels in an image are highly correlated, and we can perform then PCA in order to reduce the dimension of the image (which is $M \times N$).

The images in which the PCA were performed are called *eigenfaces*. We define a matrix D as in Equation (10), where we write all the data set comprised by p images.

$$D = \begin{pmatrix} I_1(1,1) & I_1(1,2) & \dots & I_1(1,N) & \dots & I_1(M,1) & I_1(M,2)\dots & I_1(M,N) \\ I_2(1,1) & I_2(1,2) & \dots & I_2(1,N) & \dots & I_2(M,1) & I_2(M,2)\dots & I_2(M,N) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ I_p(1,1) & I_p(1,2) & \dots & I_p(1,N) & \dots & I_p(M,1) & I_p(M,2)\dots & I_p(M,N) \end{pmatrix} = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{pmatrix} \tag{10}$$

To apply the PCA, we would have to compute the covariance matrix as in Equation (11).

$$\Sigma = \frac{1}{p-1} D^T D \tag{11}$$

After that the eigenvalues and eigenvectors are calculated and the k eigenvectors corresponding to the highest eigenvalues would be kept. Finally, an image represented by X_i can be projected in the PCA space as in Equation (12), where Φ is the projection matrix $d \times k$.

$$\phi_i = X_i \Phi \tag{12}$$

Although, matrix Σ would be a $d \times d$ matrix, which is too large for computations. There is another way of calculating it by computing the eigenvectors of matrix Σ' - Equation (13), which is a $p \times p$ matrix.

$$\Sigma' = \frac{1}{p-1} D D^T \tag{13}$$

Then, the eigenvectors of Σ' are placed as the columns of Φ' and the eigenvectors of Σ are therefore, given by $D^T\Phi'$.

This procedure is exactly what is done in function *EigenFaces*, the code below. In summary, matrix D is generated, matrix Σ' is calculated as well as its eigenvalues and eigenvectors. The eigenvectors of Σ are then computed, which composed the eigenfaces. The result of the applied PCA is in Figure (4).

```
function [PCA_database, Labels, TopEigenVectors, EigenValues, AvgFace] =
    EigenFaces(train_dir, useKEigenVectors, doOptimisation)
% Calculate Eigen Faces for training set
% reference: http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/IDAPILecture15.
    pdf

files = dir(train_dir);

FileName = {};

%each row of D corresponds to one training image
D = [];
Labels = [];

%get all training images and store them in matrix D. Store names of images
%in matrix Labels
for x = 1:length(files)
    file = files(x).name;

    if (isempty(strfind(file, 'jpg'))) == 0 || (isempty(strfind(file, 'png')))
   == 0
        imageSrcName = fullfile(train_dir,file);
        image = imread(imageSrcName);

        if doOptimisation == true
            %perform histogram equalisation to optimise recognition
            image = adapthisteq(image);
        end

        D = [D; image(:)'];
        Labels = [Labels; cellstr(file)];
    end
end

trainingSetSize = size(D);

AvgFace = mean(D);

%remove mean from training set
Dmean = double(D) - repmat(AvgFace, trainingSetSize(1), 1);

%covariance matrix - reduced form
sigmaPrim = (double(Dmean) * double(Dmean')).*(1 / (trainingSetSize(1)-1));

%Calculating eigenvectors
%eigenValues contains the eigenvalues in ascending order.
%EigenVectors is a matrix whose columns are the corresponding eigenvectors.
%we can use our function Eigen instead build-in function eig to calculate
%eigen vectors
```

```
[EigenVectors, EigenValues] = eig(sigmaPrim); %Eigen(sigmaPrim) is %very slow

%Eigen vectors of covariance matrix
EigenVectors = double(Dmean') * EigenVectors;

%normalize eigenvectors before projecting
for i = 1:size(EigenVectors, 2)
    EigenVectors(:, i) = EigenVectors(:, i)/norm(EigenVectors(:, i));
end

%transformation (projection) matrix. We use top useKEigenVectors eigenvectors
   from covariance matrix
TopEigenVectors = EigenVectors(:, 1:useKEigenVectors);

PCA_database = double(Dmean) * TopEigenVectors;
```

Figura 3: Eigenfaces generated.

After the eigenvectors are projected in the PCA space, the recognition will be processed. The similarity between the pictures was calculated by the Euclidean distance between the picture analysed (which was in the test data set) and all the pictures in the training set. The smallest Euclidean distance corresponds to the best match.

This process is performed by the function *AccuracyFaceRecognition*.

```
function [ accuracy ] = AccuracyFaceRecognition(test_dir, PCA_database, Labels,
   TopEigenVectors, AvgFace, checkNSimilarFaces, doOptimisation)
%ACCURACYRECOGNITION Provide face-recognition accuracy for given testing set

%FIND SIMILAR IMAGE FROM TEST SET

files = dir(test_dir);

totalErrors = 0;

%get all testing images and test face recognition
for x = 1:length(files)
    file = files(x).name;

    if (isempty(strfind(file, 'jpg'))) == 0 || (isempty(strfind(file, 'png')))
   == 0
        imageSrcName = fullfile(test_dir,file);
        test_image = imread(imageSrcName);

        if doOptimisation == true
            %perform histogram equalisation to optimise recognition
            test_image = adapthisteq(test_image);
        end
```

```
        %getting principal components of test image
        Dtest = (double(test_image(:)') - AvgFace) * TopEigenVectors;

        %finding most similar image from training set
        euclidianDistance = sqrt(sum((PCA_database' - repmat(Dtest, size(
    PCA_database, 1), 1)') .^ 2));
        sortedEuclidianDistance = sort(euclidianDistance);

        %find m most similar images to test one
        indexOfFace = find(euclidianDistance <= sortedEuclidianDistance(
    checkNSimilarFaces));

        recognisedImages = Labels(indexOfFace);

        pos1 = strfind(file, '.') - 2;

        matchedImage = 0;
        for i = 1:length(recognisedImages)
           recognisedImage = char(recognisedImages(i));
           pos2 = strfind(recognisedImage, '.') - 2;

           if strcmp(file(1:pos1),recognisedImage(1:pos2)) == 1
               matchedImage = 1;
           end
        end

        if matchedImage == 0
            totalErrors = totalErrors + 1;
        end
    end
end

accuracy = (1 - totalErrors/size(PCA_database, 1)) * 100;

end
```

We also implemented a gender recognition as function *AccuracyTestRecognition*. First, the PCA data set (training) is split in two parts, female and male. Then, similar to face recognition, the Euclidean distance is measured between the projection of the test image and the PCA of female and male training images. Once again, the smallest distance is considered the best match.

The code can be seen below.

```
function [ accuracySexRecognition ] = AccuracySexRecognition(test_dir,
    PCA_database, Labels, TopEigenVectors, AvgFace, doOptimisation)
%ACCURACYSEXRECOGNITION Provide sex-recognition accuracy for given testing
%set

%get all testing images and test sex recognition

female = {'Flavia', 'WINATA', 'Richa'};
femaleIndexes = [];

for i = 1:length(female)
    femaleIndexes = union(femaleIndexes, strmatch(char(female(i)), Labels));
```

```
end

PCA_female = PCA_database(femaleIndexes, :);
PCA_male = PCA_database(setdiff(1:size(PCA_database, 1), femaleIndexes), :);

totalErrors = 0;

files = dir(test_dir);

%get all testing images and test sex recognition
for x = 1:length(files)
    file = files(x).name;

    if (isempty(strfind(file, 'jpg'))) == 0 || (isempty(strfind(file, 'png')))
   == 0
        imageSrcName = fullfile(test_dir,file);
        test_image = imread(imageSrcName);

        if doOptimisation == true
            %perform histogram equalisation to optimise recognition
            test_image = adapthisteq(test_image);
        end

        %getting principal components of test image
        Dtest = (double(test_image(:)') - AvgFace) * TopEigenVectors;

        %finding average distance between test image and male/female
        %training set
        euclidianDistanceFemale = sqrt(sum((PCA_female' - repmat(Dtest, size(
   PCA_female, 1), 1)') .^ 2));
        euclidianDistanceMale = sqrt(sum((PCA_male' - repmat(Dtest, size(
   PCA_male, 1), 1)') .^ 2));


        totalDistanceFemale = mean(mean(euclidianDistanceFemale));
        totalDistanceMale = mean(mean(euclidianDistanceMale));

        sex = 'male';
        if (isempty(strfind(file, 'Flavia'))) == 0 || (isempty(strfind(file, '
   WINATA'))) == 0 || (isempty(strfind(file, 'Richa'))) == 0
            sex = 'female';
        end

        if (totalDistanceFemale < totalDistanceMale && strcmp(sex, 'male'))
            %detected female, test image is male
            totalErrors = totalErrors + 1;
        end

        if (totalDistanceFemale > totalDistanceMale && strcmp(sex, 'female'))
            %detected male, test image is female
            totalErrors = totalErrors + 1;
        end
    end
end

accuracySexRecognition = (1 - totalErrors/size(PCA_database, 1)) * 100;
end
```

The functions *ShowEigenFaces*, *ShowPictures*, *ShowEigenValues* and *ShowFeaturePoints* are display-auxiliary function. They show eigenfaces, normalized images, eigenvalues and feature points, respectively.

The main function is called *PCA_ Recognition*, shown in the following code. It consists in do the normalization (*NormalizeImage*) of the data set, then PCA is perfomed (*EigenFaces*) and finally the faces and gender recognition (*AccuracyFaceRecognition* and *AccuracySexRecognition*.

```
function [] = PCA_Recognition( doNormalization,  doOptimisation)
%PCA_RECOGNITION Main function that performs face recognition and sex
%recognition
%Parameters:
%@doNormalization - (true or false) normalize input pictures and divide them
    into training
%and test set
%@doOptimisation - (true or false) optimise image (optimise intensity level,
    contrast etc.)
%before performing face recognition

close all;

features_dir = 'features_data';
faces_dir = 'faces_data';
train_dir = 'train_images';
test_dir = 'test_images';

if doNormalization == true
    %normalize images - adjust location of features of each image
     NormalizeImages(features_dir, faces_dir, test_dir, train_dir, doOptimisation
    );
end

ShowPictures(train_dir, test_dir);

accuracies = [];
checkNsimilarFacesSet = [];
useTopEigenVectors = 30;

[PCA_database, Labels, TopEigenVectors, EigenValues, AvgFace] = EigenFaces(
    train_dir, useTopEigenVectors, doOptimisation);

% GET GLOBAL STATISTICS FOR FACE RECOGNITION

for checkNsimilarFaces = 1:5
        accuracy = AccuracyFaceRecognition(test_dir, PCA_database, Labels,
    TopEigenVectors, AvgFace, checkNsimilarFaces, doOptimisation);
        accuracies = [accuracies accuracy];
        checkNsimilarFacesSet = [checkNsimilarFacesSet checkNsimilarFaces];
end

ShowEigenValues(EigenValues);
ShowEigenFaces(PCA_database, TopEigenVectors);

%show relation between number of eigen vectors, number of similar faces and face
```

```
    recognition accuracy
figure('units','normalized','outerposition',[0 0 1 1], 'Name','Accuracy of face
    recognition');
scatter(checkNsimilarFacesSet, accuracies);
title('Relation between number of similar faces and accuracy')
xlabel('Nuber of smilar faces');
ylabel('Accuracy');
set(gca,'xtick',0:5);
grid on;

% GET GLOBAL STATISTICS FOR SEX RECOGNITION

sexAccuracy = AccuracySexRecognition(test_dir, PCA_database, Labels,
    TopEigenVectors, AvgFace, doOptimisation);

%show relation between number of eigen vectors and sex accuracy
figure('units','normalized','outerposition',[0 0 1 1], 'Name','Accuracy of sex
    recognition');
bar(useTopEigenVectors, sexAccuracy);

title('Accuracy of sex recognition');
xlabel('Number of Eigen vectors');
ylabel('Accuracy');
grid on;

clear all;

end
```

The function takes two boolean parameters. The first one corresponds to *doNormalization*, to normalize the images and split them into two folders. The second one is the *doOptimization*, which performs histogram equalizations in the images after they are normalized.

5 Results

5.1 Face Recognition

The accuracy of the face recognition was measured for different cases.

For the analysis, the graph shows the accuracy for number of match training images with the test image.

Figures 5.1 and 5.1 were plotted considering the optimization (histogram equalization).

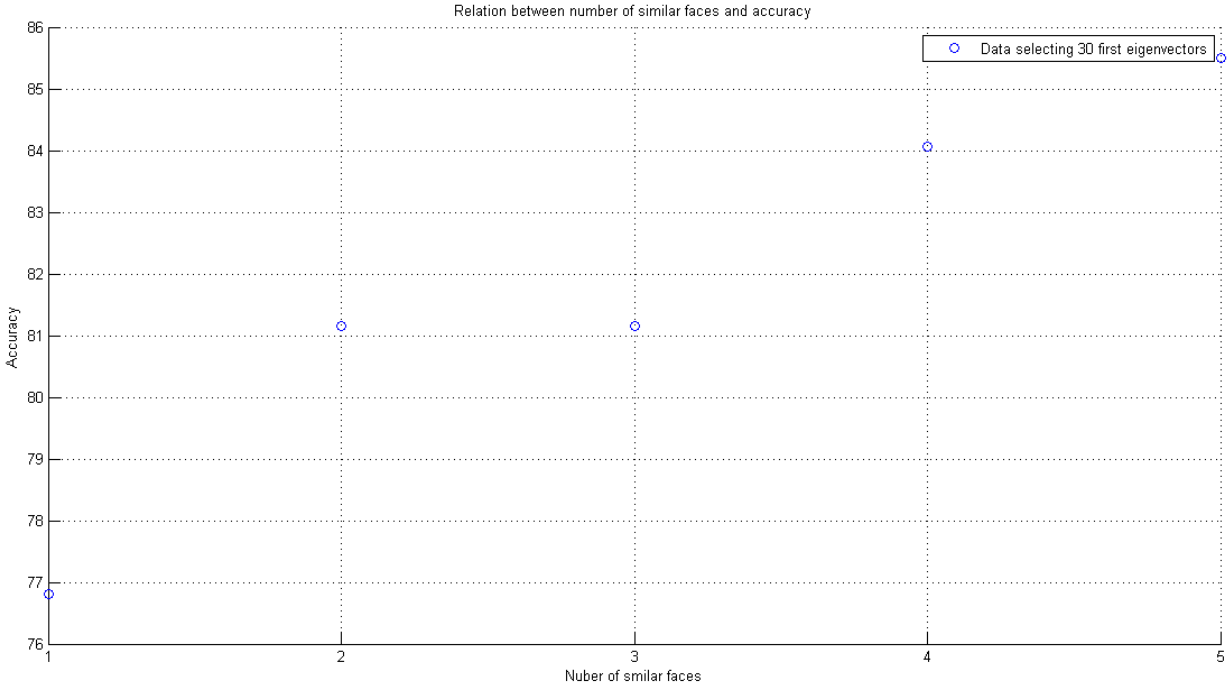

Figura 4: Accuracy measured considering the 30 first eigenvectors and with optimization.

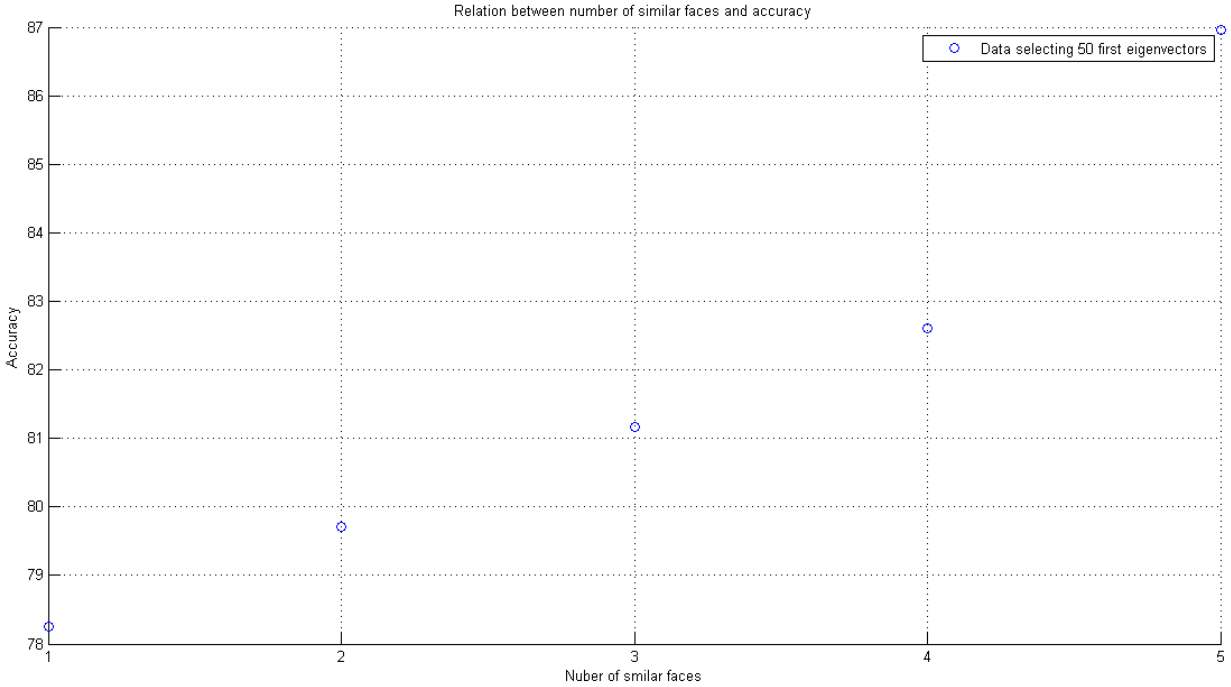

Figura 5: Accuracy measured considering the 50 first eigenvectors and with optimization.

Therefore, for Figure 5.1, the accuracy that the best match is the right match is of 70%. Considering the five best matches, the accuracy is almost 86%. It is evident that selecting more

eigenvectors the accuracy is increased. The accuracies would be then 78% and 87%, for first and five first best matches.

Figures 5.1 and 5.1 shows the accuracy without performing the optimization.

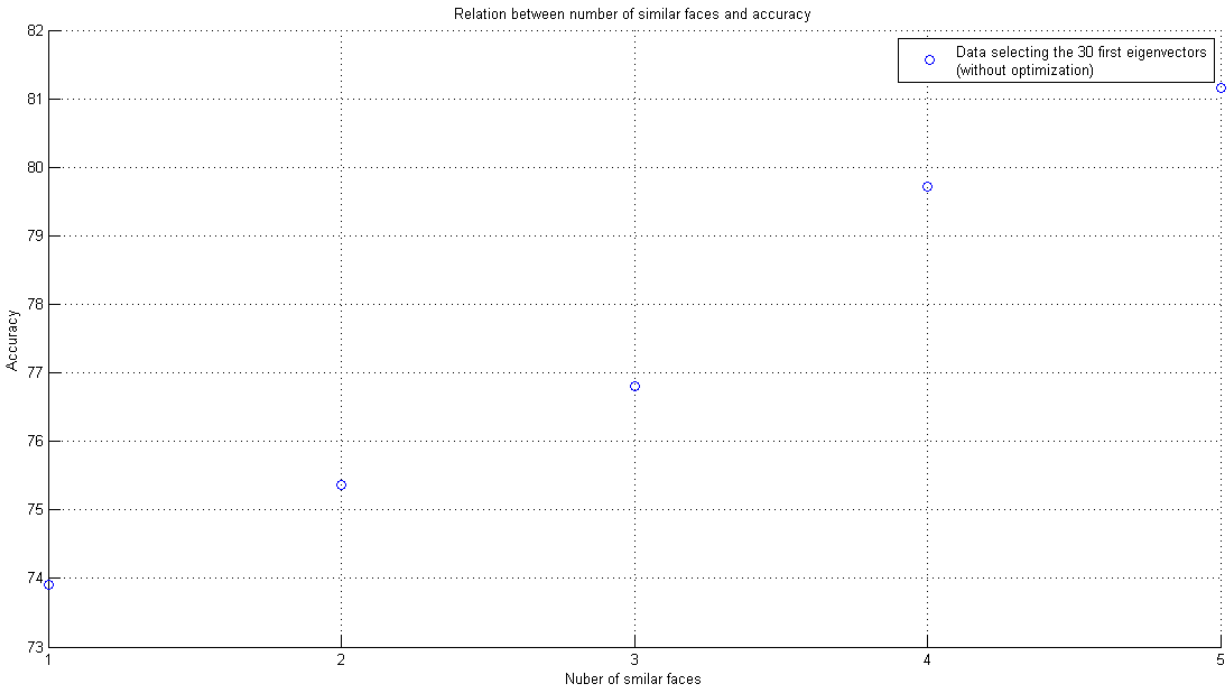

Figura 6: Accuracy measured considering the 30 first eigenvectors and without optimization.

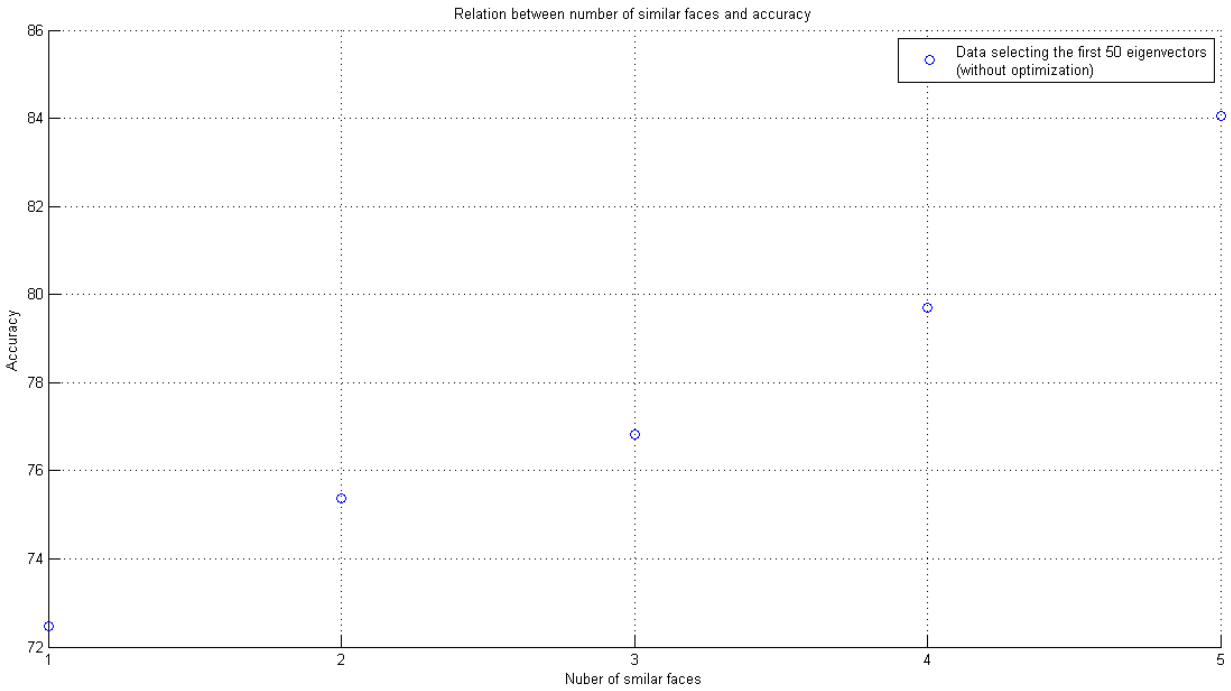

Figura 7: Accuracy measured considering the 50 first eigenvectors and without optimization.

From the graphs it can be observed that the accuracy is much smaller when the optimization is not performed. When keeping the 30 first eigenvectors, for instance, the accuracy for first and five first images is 74% and 81%.

5.2 Gender Recognition

The accuracy for gender recognition achieved 95%, for both cases of getting either 30 or 50 first eigenvectors.

5.3 Result script

The Matlab script *example* gets one image from the test data folder and gives as result the five best matches from the training data set and the gender. The result is shown in Figure 5.3.

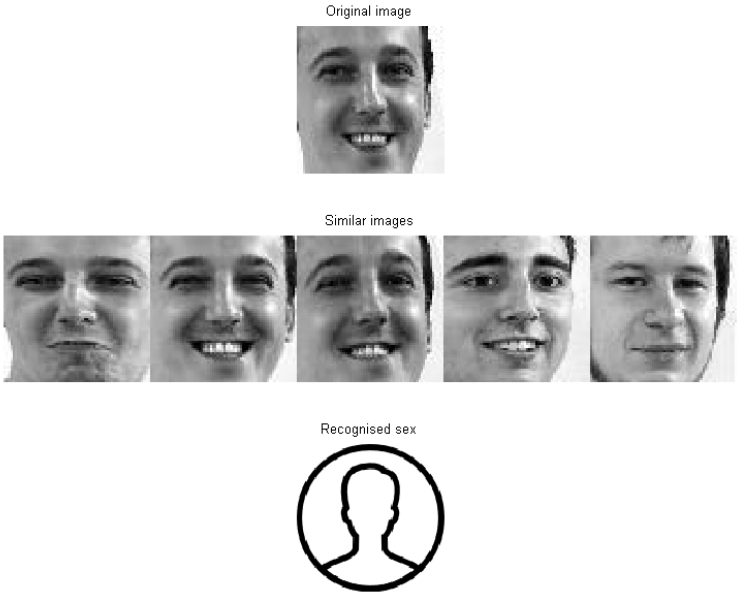

Figura 8: Output of the script *example*.

Then, the only three images of the training set corresponding to the same person were well recognised (first three matches).

6 Conclusion

Currently, PCA is not the best way of performing a face recognition program. Other methods are more suitable for it, such feature-base techniques or multiple classifier systems ([3]). However, PCA is a traditional, fast and relatively simple solution, considering a constrained environment.

The accuracy of the program implemented is not so high, but it should be taken in account that many face recognition techniques do not require a perfect identification. It depends also on the application.

Finally, it was a really interesting project to be implemented, gathering two important definitions (SVD and PCA) and using them in a practical application.

Referências

[1] Mark Richardson *Principal Component Analysis* May, 2009.

[2] Jeff Jauregui *Principal component analysis with linear algebra* August, 2012.

[3] Rabia Jafri and Hamid R. Arabnia *A Survey of Face Recognition Techniques* June 2009.