MINISTRY FOR EDUCATION AND SCIENCE OF RUSSIA

SAINT PETERSBURG ELECTROTECHNICAL UNIVERSITY «LETI»

Course project
Biometric Technologies

Student
Group 4300
Bathaie N.

Teacher
PhD, associate prof.
of CE department
Shegoleva N.L.

Saint Petersburg
ETU «LETI»
2019

MULTI-DIRECTIONAL TWO-DIMENSIONAL PCA WITH MATCHING SCORE LEVEL FUSION FOR FACE RECOGNITION

Contents

1. Goals of the Article

Many algorithms are used to reduce the dimensions of features in the application of face recognition, the most important of which is the Principal Component Analysis algorithm (PCA). If we use the brightness levels of the pixels directly as a feature, the vector dimensions of the features will be very high, and applying the PCA method on it will involve a lot of computational complexity.

Two-dimensional PCA algorithms (2DPCAs) applied to image rows or columns solve the problem of large feature dimensions. In these algorithms, each row or column of the image is considered as a sample in the Hyper Space. In this case, only the features can be extracted from the characteristics that are horizontal or vertical. This article provides an algorithm that can extract structural features in any direction of the image. Then, by combining PCA results in different directions, it is possible to combine structural specifications in different aspects of the image, and as a result, better results can be achieved. In this paper, the aim is not to provide an accurate algorithm for face recognition, but in this article, the authors try to show the effectiveness of the proposed algorithm compared to other similar algorithms, in the form of a simple classifier results.

2. Two-dimensional Principal Components Analysis

Assuming that X is a single n-dimensional vector and A is a matrix with dimensions of $m \times n$, the following projects the matrix A linearly on vector X:

$$Y = \boldsymbol{A}X$$

Using this equation, the next m vector of Y is obtained, which is called the Projected Feature Vector of the image. Different criteria can be used to find the right X vector. In the two-dimensional principal component analysis method, we use sample dispersion as a criterion for measuring the vector power of projecting X. The scattering of the projected samples can be obtained from the TRACE of the covariance matrix. As a result, we use the following criteria to find the right X vector:

$$J(X) = tr(S_x)$$

The expression S_x represents the covariance matrix of the projected samples. By maximizing the $J(X)$ criterion, the most suitable vector for X projection is obtained, in which the projection of the samples leads to the most general result. The covariance matrix (S_x) can be obtained as follows:

$$S_x = E\{(Y - \mu_Y)\{(Y - \mu_Y)^T\}$$

$$= E\{(\boldsymbol{A}X - E\{\boldsymbol{A}X\})\{(\boldsymbol{A}X - E\{\boldsymbol{A}X\})^T\}$$

$$= E\{[(\boldsymbol{A} - \mu_{\boldsymbol{A}})X][(\boldsymbol{A} - \mu_{\boldsymbol{A}})X]^T\}$$

On the other hand, due to the trace property of multiplication of two matrices we have:

$$tr(AB) = tr(BA)$$

Therefore, $J(X)$ can be calculated as following:

$$J(X) = tr(S_x) = tr(E\{[(\boldsymbol{A} - \mu_{\boldsymbol{A}})X][(\boldsymbol{A} - \mu_{\boldsymbol{A}})X]^T\})$$

$$= tr(E\{[(\boldsymbol{A} - \mu_{\boldsymbol{A}})X]^T[(\boldsymbol{A} - \mu_{\boldsymbol{A}})X]\}) = E\{[(\boldsymbol{A} - \mu_{\boldsymbol{A}})X]^T[(\boldsymbol{A} - \mu_{\boldsymbol{A}})X]\}$$

$$= X^T E\{(\boldsymbol{A} - \mu_{\boldsymbol{A}})^T(\boldsymbol{A} - \mu_{\boldsymbol{A}})\}X$$

Now we can simplify $J(X)$

$$J(X) = X^T G_t X$$

Where G_t is the Image Covariance Matrix:

$$G_t = E\{(\boldsymbol{A} - \mu_{\boldsymbol{A}})^T(\boldsymbol{A} - \mu_{\boldsymbol{A}})\} \cong \frac{1}{l}\sum_{i=1}^{l}(\boldsymbol{A}_i - \overline{\boldsymbol{A}})^T(\boldsymbol{A}_i - \overline{\boldsymbol{A}})$$

In order to find the X_{opt} vector that optimizes $J(X)$, we define the optimization problem as follows:

$$\mathrm{X_{opt}} = max_X\big(J(X)\big) \quad when \quad X^T X = 1$$

This optimization problem can be solved using the Lagrange equation:

$$X_{opt} = max_X\{X^T G_t X + \lambda X^T X\}$$

$$\rightarrow 2G_t X_{opt} + 2\lambda X_{opt} = 0$$

$$\rightarrow G_t X_{opt} = \lambda X_{opt}$$

Now we can see that the X_{opt} is the same as the G_t. It is usually not enough to use an optimal projection vector to write matrix A. In other words, we need a set of vectors ($\{X_1, X_2, \dots, X_d\}$) to project onto the matrix A, while they are pairwise orthogonal:

$$\begin{cases} \{X_1, X_2, \dots, X_d\} = argmax\{J(x)\} \\ X_i^T X_j = 0, i \neq j \quad , i, j = 1,2, \dots, d \end{cases}$$

In this case, these projection vectors $\{X_1, X_2, \dots, X_d\}$ will be the d feature vectors corresponding to the d highest eigen values of the G_t matrix.

3. Feature extraction using Two Dimensional Principal Component Analysis

Using the method presented in the previous section, we can calculate the covariance matrix (G_t) using the images in the training set, and then we can calculate the $\boldsymbol{d}$ feature vectors corresponding to $\boldsymbol{d}$ highest eigen values of the G_t matrix.

Finally, the feature vectors $\{Y_1, Y_2, \dots, Y_d\}$ are calculated according to the following equation:

$$Y_1 = \boldsymbol{A}X_1$$

$$Y_2 = \boldsymbol{A}X_2$$

...

$$Y_d = \boldsymbol{A}X_d$$

The feature vectors $\{Y_1, Y_2, \dots, Y_d\}$ are called the Principal Components of A. Unlike PCA, where the main components were scalar, in this method, the main components are vectors. Feature Extraction of vectors $\{Y_1, Y_2, \dots, Y_d\}$ can be expressed as a matrix multiplication:

$$\boldsymbol{B} = \boldsymbol{AX}$$

$$\boldsymbol{B} = [Y_1, Y_2, \dots, Y_d]$$

$$\boldsymbol{X} = [X_1, X_2, \dots, X_d]$$

Subsequently, image $\boldsymbol{A}$ with dimension of $m \times n$ is projected to matrix $\boldsymbol{B}$ with smaller dimension of $m \times d$ using the analysis of the main two-dimensional components. Matrix B is called the Feature Matrix of A.

4. Features Classification

The feature matrix of all images in the training set can be obtained using Equation mentioned above. In order to classify a test sample, we obtain the matrix of its properties and classify it using the Nearest Neighbor classifier.

For this purpose, we obtain the distance of the test sample feature matrix from the feature matrix of all the samples in the training set and finally assign it to the class with the shortest distance.

Is. The distance between the two feature matrices B_i and B_j is calculated using Equation below:

$$d(B_i, B_j) = \sum_{k=1}^{d} \left\| Y_k^{(i)} - Y_k^{(j)} \right\|$$

Where $\left\| Y_k^{(i)} - Y_k^{(j)} \right\|$ is the Euclidean Distance between the two feature matrices.

5. Reconstruct images using two-dimensional Principal component analysis

As described in the previous sections, in order to calculate the projection vectors $\{X_1, X_2, \dots, X_d\}$, first we calculate the $\boldsymbol{d}$ feature vectors corresponding to $\boldsymbol{d}$ highest eigen values of the covariance matrices of images and by projecting the image A onto the vector set. In the projection, we obtain the matrix of its properties.

Since the projection vectors are one-dimensional, image A can be reconstructed from its feature matrix according to Equation below:

$$\widetilde{\boldsymbol{A}} = \boldsymbol{B}\boldsymbol{X}^T$$

Where $\widetilde{\boldsymbol{A}}$ is a reconstructed image from its feature matrix ($\boldsymbol{B} = [Y_1, Y_2, \dots, Y_d]$), which can be rewritten as:

$$\widetilde{\boldsymbol{A}} = \sum_{i=1}^{d} \widetilde{\boldsymbol{A}}_i = \sum_{i=1}^{d} Y_i X_i^T$$

In other words, the reconstructed image $\widetilde{\boldsymbol{A}}$ is obtained from the d sub-images $\widetilde{\boldsymbol{A}}_i = Y_i X_i^T$. The higher the number of sub-images, the better the reconstructed image of A will be. If d is equal to n (the number of columns in image A), images A and $\widetilde{\boldsymbol{A}}$ are equal.

6. Alternative Tow Dimensional Principal Component Analysis

The 2DPCA algorithm described in the previous sections has a much lower computational complexity than the PCA method. The 2DPCA algorithm is applied to the image lines so that each line of an image $\boldsymbol{A}_{m\times n}$ can be considered as a sample in an n-space. The 2DPCA algorithm can also be applied to image columns. For this purpose, the algorithm can be applied to the transposed images.

Applying the 2DPCA algorithm on the transposed images is called Alternative_2DPCA. In this algorithm, rows and columns are replaced, and as a result, the principal components are analyzed on the image columns. As a result, each column of the image $\boldsymbol{A}_{m\times n}$ can be considered as a sample in an m-space.

7. Directional Two Dimensional Principal Component Analysis Matrix

The D2DPCA algorithm is obtained by applying the Tow Dimensional Principal Component Analysis method on the rotated samples of the images at a certain angle.

For this purpose, we first rotate all the images in the train set with θ,Which can be done using the relational rotation matrix :

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Then we obtain the covariance matrix of the rotated images ($\boldsymbol{G}_{D2DPCA}$) according to the equation below:

$$\boldsymbol{G}_{D2DPCA} = \frac{1}{l}\sum_{i=1}^{l}\left(r_\theta(\boldsymbol{A}_i) - \overline{r_\theta(\boldsymbol{A})}\right)^T\left(r_\theta(\boldsymbol{A}_i) - \overline{r_\theta(\boldsymbol{A})}\right)$$

Finally, in order to calculate the projection vectors, we calculate the $\boldsymbol{d}$ feature vectors corresponding to $\boldsymbol{d}$ highest eigen values of the $\boldsymbol{G}_{D2DPCA}$ matrix.

If we displays the d feature vector as $\boldsymbol{X} = \left[X_1^{D2DPCA}, X_2^{D2DPCA}, \dots, X_d^{D2DPCA}\right]$, the feature matrix of the image $r_\theta(\boldsymbol{A}_i)$ is obtained as follows:

$$\boldsymbol{Y} = \left[Y_1^{D2DPCA}, Y_2^{D2DPCA}, \dots, Y_d^{D2DPCA}\right] = r_\theta(\boldsymbol{A}_i)\boldsymbol{X}$$

8. Multi-Directional two Dimensional Principal Component Analysis

In the Multi-Directional Tow Dimensional Principal Component Analysis (MD2DPCA) proposed in this paper, the characteristic properties of the extracted D2DPCA algorithm are used in different directions.

For this purpose, for each test image, we form a bank of D2DPCA feature matrices, and for each, we obtain the score of its compliance with all instances of the training set.

Finally, by fusing these scores, we obtain the final match score and use it to determine the sample category or class of the test.

The implementation steps of the MD2DPCA algorithm can be described as follows:

1. We initialize the angle $\boldsymbol{\theta}$ in $\boldsymbol{C}$ different directions from $\mathbf{0}\ \boldsymbol{to}\ \mathbf{2}\boldsymbol{\pi}$ and implement the algorithm D2DPCA for each of the angles of the set $\boldsymbol{\theta} = \{0, \frac{\boldsymbol{\pi}}{\boldsymbol{C}}, \dots, \frac{\boldsymbol{\pi}(\boldsymbol{C}-\mathbf{1})}{\boldsymbol{C}}\}$. Then, we obtain projection vectors for each angle.

$$\boldsymbol{X}^{\theta_0} = \left[X_1^{\theta_0}, X_2^{\theta_0}, \dots, X_d^{\theta_0}\right] \rightarrow \quad D2DPCA\ \ feature\ vectors\ with\ \theta_0 = 0$$

$$\boldsymbol{X}^{\theta_1} = \left[X_1^{\theta_1}, X_2^{\theta_1}, \dots, X_d^{\theta_1}\right] \rightarrow \quad D2DPCA\ \ feature\ vectors\ with\ \theta_1 = \frac{\boldsymbol{\pi}}{\boldsymbol{C}}$$

....

$$\boldsymbol{X}^{\theta_{C-1}} = \left[X_1^{\theta_{C-1}}, X_2^{\theta_{C-1}}, \dots, X_d^{\theta_{C-1}}\right] \rightarrow \quad D2DPCA\ \ feature\ vectors\ with\ \theta_{C-1} = \frac{\boldsymbol{\pi}(\boldsymbol{C}-\mathbf{1})}{\boldsymbol{C}}$$

2. For each of the angles of the $\boldsymbol{\theta}$ set, we measure the image of all the images in the training set on the corresponding projection vectors and obtain their feature matrices.

In the following equation, $\mathbf{B}_{\mathrm{i}}^{\theta_{\mathrm{k}}}$expresses the matrix of $\mathrm{A_i}$ image properties of the training set. $\mathbf{B}_{\mathrm{Test}}^{\theta_{\mathrm{k}}}$ also shows the matrix of $\mathrm{A_{Test}}$ test image features.

$$\boldsymbol{B_i^{\theta_k}} = r_{\theta_k}(\boldsymbol{A_i})\boldsymbol{X}^{\theta_k} \quad for \quad \forall \boldsymbol{A_i} \in \{Train\ Images\} \ \ \& \ \ \forall \theta_k \in \{0, \frac{\pi}{C}, \dots, \frac{\pi(C-1)}{C}\}$$

$$\boldsymbol{B_{Test}^{\theta_k}} = r_{\theta_k}(\boldsymbol{A_{Test}})\boldsymbol{X}^{\theta_k} \quad \forall \theta_k \in \{0, \frac{\pi}{C}, \dots, \frac{\pi(C-1)}{C}\}$$

3. We calculate the dissimilarity between the matrix of test sample properties and all training samples for all angles:

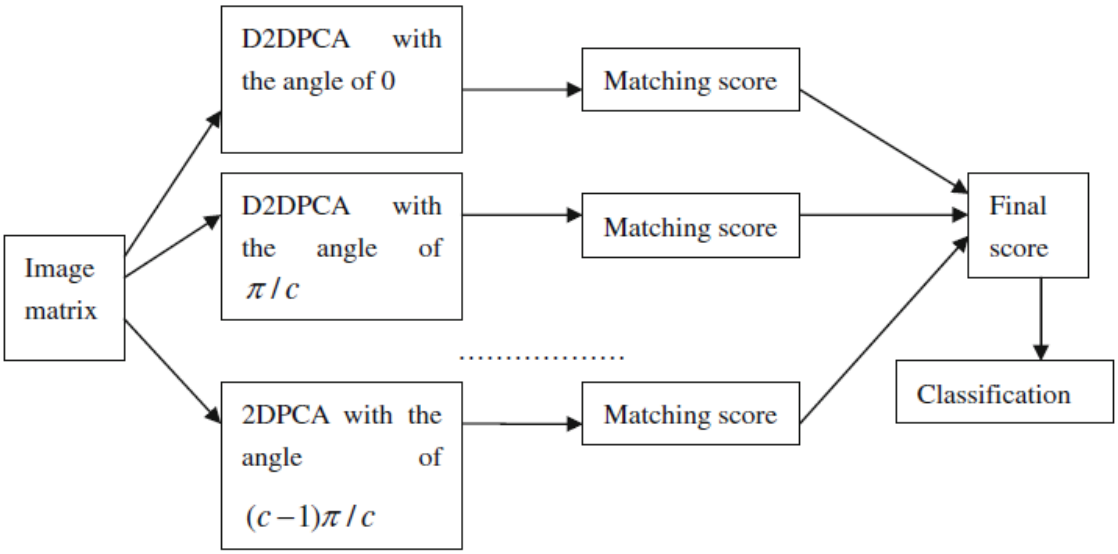
$$S(\boldsymbol{A_{Test}}, \boldsymbol{A_i}, \theta_k) = \sum_{k=1}^{d} \left\| \boldsymbol{B_{Test}^{\theta_k}}(:, k) - \boldsymbol{B_i^{\theta_k}}(:, k) \right\|$$

4. The final dissimilarity between the test sample and all educational samples is obtained by summing the dissimilarity in all directions:

$$S(A_{Test}, \boldsymbol{A_i}) = \sum_{\boldsymbol{k=0}}^{\boldsymbol{C-1}} S(\boldsymbol{A_{Test}}, \boldsymbol{A_i}, \theta_k)$$

5. Using the nearest neighbor classifier, we obtain the category or class assigned to the $\boldsymbol{A_{Test}}$ **(**test image). For this class, we consider the test sample to be the same as the training sample class that has the shortest distance from the test sample.

   The framework of MD2DPCA is as below:



9. The database

In order to evaluate the proposed algorithm, we use the AR database. The database available on the net contains 2,600 images of the faces of 100 different people, including 50 men and 50 women. For each person, there are 26 images in different lighting conditions, different facial expressions (such as smile, frown, scream) and different obstructions (obstruction with scarf, glasses, etc.).

In order to maintain the initial possibility of implementing here and the article, we randomly select all the images of men and images of 40 women.

The following figure shows a sample of all 26 images of a person's faces on the AR database.

Out of 26 images for each person, 13 images randomly separated for the train set rest used for the test set. The images in the database have three color channels with dimensions of 120 × 165.

Like the implementation in the article, I converted these images to gray levels and reduced their dimensions to 50 × 40. Images of training and testing, after being changed into gray scale and reduced in size, are placed in two folders, **train_44** and **test_44**.

10. Implemented Code

In order to implement all matrix-based PCA algorithms (2DPCA, Alternative_2DPCA, 2D2DPCA, 4D2DPCA and 6D2DPCA), two main functions have been written, one for training and the other for testing samples.

The **TrainAlgorithms** function is responsible for training samples and the **TestAlgorithms** function written for testing samples. For simplicity, each of these

functions has different input arguments that have the flexibility to implement different algorithms with different settings.

10.1. TrainAlgorithms Function

This function is defined as the following:

$$[\text{ImTrainPca}, \text{TrainLabel}, \text{Vprj}] = \text{TrainAlgorithms}(\text{dir1}, \text{method}, \text{NumOfEig}, \text{Percent})$$

The input arguments for this function are as following:

1. dir1:

   The address of the folder containing the images for train. The folder can contain any number of images for the train set. The name of each image file contains the label that determines the category of these images, and as a result, the image tags are not given to the function separately.

2. Method:

   A String variable that allows us to determine the type of algorithm. This field can be one of the following:

   - 2DPCA for implementing the two-dimensional Principal Component Analysis algorithm.
   - Alt2DPCA to implement the Alternative two-dimensional Principal Component Analysis algorithm.
   - 2D2DPCA for implementing two-Directional two Dimensional Principal Component Analysis.

- 4D2DPCA for implementing the four-Directional two Dimensional Principal Component Analysis (proposed algorithm in the article)

- 6D2DPCA to implement six-Directional two Dimensional Principal Component Analysis (proposed algorithm in the article)

3. NumOfEig:

   Which is an integer variable to determine the number of eigen vectors in the projection vector extraction step. In this case, we leave the ***Percent*** argument empty.

4. Percent:

   If we want to select the number of eigen vectors from the energy diagram of the eigen values in such a way that Percent maintains the percentage of energy of the samples, we set this value to zero and a number between zero and one. In this case, we leave ***NumOfEig*** empty. The output arguments of this function are:

   - ImTrainPca:

     A cell with the number of valves equal to the number of angles in each algorithm, each of which contains a matrix of properties corresponding to the D2DPCA.

   - TrainLabel:

     A cell with the number of drives equal to the number of instances of the instruction set that contains the class tag of each instance.

- Vprj:

  A cell with the number of valves equal to the number of angles in each algorithm, each of which contains the projection matrices corresponding to D2DPCA in that direction.

The ***TrainAlgorithm*** function is as below:

```
function [ImTrainPca,TrainLabel,Vprj]=TrainAlgorithms(dir1,method,NumOfEig,Percent)
%% The Method Defines Number Of Directions Inwhich 2DPCA Should Be Implemented
switch method
    case '2DPCA'
        disp('Implementing Two Dimensional Principal Algorithm');
        ImTrainPca=cell(1,1);
        theta=0;
    case 'Alt2DPCA'
        disp('Implementing Alternative Two Dimensional Principal Algorithm');
        ImTrainPca=cell(1,1);
        theta=90;
    case '2D2DPCA'
        disp('Implementing 2 Directional Two Dimensional Principal Algorithm');
        ImTrainPca=cell(1,2);
        theta=[0,90];
    case '4D2DPCA'
        disp('Implementing 4 Directional Two Dimensional Principal Algorithm');
        ImTrainPca=cell(1,4);
        theta=0:180/4:3*180/4;
        theta=[theta,90];
    case '6D2DPCA'
        disp('Implementing 6 Directional Two Dimensional Principal Algorithm');
        ImTrainPca=cell(1,6);
        theta=0:180/6:5*180/6;
        theta=[theta,90];
end
%% Calculating Image Covariance Matrix
[Gmat,Images,TrainLabel]=ImCovMat(dir1,theta);
%% Calculating Projection Vectors
Vprj=ProjectionMatrix(Gmat,NumOfEig,Percent);
%% Projecting All Training Samples on Projection Vectors
disp('Calculating Feature Matrix Of Training Samples')
for k=1:length(theta)
    disp([num2str(floor(k/length(theta)*100)),' Percent...'])
    sample=Images(:,:,1);
    I=imrotate(sample,theta(k),'bicubic');
    TrainPca=zeros(size(I,1),NumOfEig,size(Images,3));
    Vproj=Vprj{k};
    for i=1:length(dir1)
        I=double(Images(:,:,i));
        I=imrotate(I,theta(k),'bicubic');
```

```
42          I(I>=230)=0;
43          TrainPca(:,:,i)=I*Vproj;
44        end
45        ImTrainPca{1,k}=TrainPca;
46    end
47    %"ImTrainPca" is a cell that each of its elements is a m*d*N matrix where m
48    %is number of rows of images, d is number of eigen vectors and N is number of trainig images
49    disp('Feature Matrix Of Training Samples Calculated!')
```

The code written on lines 3 to 26 is only for determining the type of algorithm. Depending on the user's chosen algorithm, the number of angles is obtained. For example, if the user selects the method equal to '4D2DPCA', the angles are initialized as below:

$$theta = [0\ 45\ 90\ 135]$$

In line 28, the ***ImCovMat*** function is called to calculate the covariance matrix of images. This function gives three outputs in the form of [***Gmat, Images, TrainLabel***]. Where ***Gmat*** is a cell with the size of $\mathbf{1} \times \boldsymbol{NumOfTheta}$, and each of its elements contains the covariance matrix of the samples corresponding to the angle. ***Images*** is also a $\boldsymbol{m} \times \boldsymbol{n} \times \boldsymbol{N}$ matrix that contains all the images in the train set (m and n represent the number of rows and columns in the images, respectively. N also shows the total number of images in the tutorial).

TrainLabel is also a cell with the size of the number of samples in the category or class of each sample. The names of the categories are extracted from the names of each image file.

In line 29, the ***ProjectionMatrix*** function is called. I have written this function to extract feature vectors from covariance matrices. This function takes the covariance matrix (Gmat), the number of eigen vectors (NumOfEig) and the percentage of energy of the eigen values (Percent) as input, and returns the Vprj cell with the size of $\mathbf{1} \times \boldsymbol{NumOfTheta}$ as output. Each of the cells in the cell contains an m × d matrix, containing $m - s$ eigen vector corresponding to the covariance matrix at one of the desired angles.

In lines 33 to 46, each image of the training set is multiplied to the projection matrices, and we get the feature matrix of that image in all the required directions and save them in the ***ImTrainPca*** cell.

10.2. ImCovMat Function

This function receives the address of the image folder of the training set along with the corresponding angles of the desired algorithm as input and returns the covariance matrices as output. This function is as following:

```
1   function [GMat,Images,TrainLabel]=ImCovMat(dir1,theta)
2   %ImCovMat  takes "dir1" folder of training images as input
3   %ImCovMat gives Image Covariance Matrix, "G" as output
4   %% Calculating The Mean of Training Images
5   Imean=0;
6   I=imread(['./train_44/',dir1(1).name]);
7   Images=zeros(size(I,1),size(I,2),length(dir1));
8   TrainLabel=cell(1,length(dir1));
9   disp('Reading Training Images...');
10  unit=floor(length(dir1)/10);
11  for i=1:length(dir1)
12     if mod(i,unit)==0
13        disp([num2str(i/length(dir1)*100),' Percent...']);
14     End
15     ImName=dir1(i).name;
16     I=imread(['./train_44/',ImName]);
17     TrainLabel{i}=ImName(1:5);
18     %     I=rgb2gray(I);
19     I=double(I);
20     %     I=double(imresize(I,[50,40]));
21     Imean=Imean+I;
```

```
22      Images(:,:,i)=I;
23  End
24  Imean=Imean/length(dir1);
25  disp('All Images Has Been Read!');
26  %"Imean" is the mean of all traning images
27  %.....
28  %.....
29  %% Calculating the Image Covariance Matrix "G"
30  disp('Calculating Image Covariance Matrix, Please Wait...');
31  GMat=cell(1,length(theta));
32  for k=1:length(theta)
33      disp([num2str(floor(k/length(theta)*100)),' Percent...'])
34      G=0;
35      Imeank=imrotate(Imean,theta(k),'bicubic');
36      Imeank(Imeank>=230)=0;
37      for i=1:length(dir1)
38          I=Images(:,:,i);
39          I=imrotate(I,theta(k),'bicubic');
40          I=double(I);
41          I(I>=230)=0;
42          %     I=double(imresize(I,[50,40]));
43          G=G+(I-Imeank)'*(I-Imeank);
44      End
45      G=G/length(dir1);
46      GMat{k}=G;
47  End
48  disp('Image Covariance Matrix Calculated!');
49  % "G" is the Image Covariance Matrix
50  %if image size is m*n, then the size of G is n*n
51  %.....
52  %.....
```

On lines 5 to 26, the average image is obtained in non-rotated mode. On lines 30 to 48, covariance matrices are obtained in all directions. For example, if the algorithm selected by the user is the 4D2DPCA algorithm, theta angle is initialized as [0 45 90 135]. Then, for each of these angles, all the images in the tutorial set are rotated and the covariance matrix of the rotated images is obtained. The covariance matrix is stored per line 46 in one of the GMat cell valves.

Since in the next steps we need the images of the training set, in order to increase the execution speed of the program, we will pour all the images of the training set after reading inside a three-dimensional matrix called ***Images*** in line 22.

After calling each file, we can get the label of each instruction from its name and save it in a cell called ***TrainLabel***.

10.3. ProjectionMatrix function

This function receives the cell containing the covariance matrices (GMat) as well as the number of eigen vectors (NumOfEig) as input and returns the cell containing the projection matrix (Vprj) as output.

This function is as follows:

```
1   function Vprj=ProjectionMatrix(Gmat,NumOfEig,Percent)
2   %% ProjectionMatrix calculates NumOfEig Projection Vectors  corresponding to G
3   disp('Calculating Projection Vectors...')
4   Vprj=cell(1,length(Gmat));
5   for k=1:length(Gmat)
6      G=Gmat{k};
7   [V,D]=eig(G);
8   D=diag(D);
9   %sort eigen vectors according to largest eigen value o smallest one
10  [DVal,Dind]=sort(D,'descend');
11  Vsort=V(:,Dind);
12  if (isempty(NumOfEig))&(~isempty(Percent))
13     %calculating cumulative D
14     SumD=zeros(size(DVal));
15  for k=1:length(DVal);
16     SumD(k)=sum(DVal(1:k))/sum(DVal);
17  End
18  figure;plot(SumD,'LineWidth',2);
19  grid on;hold on
20  [MinVal,MinInd]=min(abs(SumD-Percent));
21  plot([0,1],[MinInd,MinInd],'--')
22  pause(.5)
23  NumOfEig=MinInd;
24  elseif ~((~isempty(NumOfEig))&(isempty(Percent)))
25     error('Input values NumOfEig, Percent are not correct');
26  end
27  Vprj{k}=Vsort(:,1:NumOfEig);
```

```
28  End
29  disp('Projection Vectors Calculated!')
30  %Vprj is the projection matrix
31  %.....
32  %.....
```

In line 5 there is a loop on all angles defined in the algorithm. In line 6 of the program, we store the corresponding covariance matrix of each angle in the variable G and in line 7 we calculate eigen values and the eigen vectors. In line 11, we sort these eigen vectors from the highest to the lowest, and in line 27, we select the $\boldsymbol{NumOfEig}$ number of these vectors as the projection vector and store them as matrices in the $\boldsymbol{Vpr}$j cell.

10.4. TestAlgorithms function for testing samples

The TestAlgorithms function receives a folder containing test images along with a cell containing projection vectors, test image feature matrices, and test sample tags as input, and returns error rate, predicted tags, and main test image tags as output. This function is as bellow:

```
1   function
    [Error,PredictedLabel,TestLabel,Mask]=TestAlgorithms(dir2,Vprj,method,ImTrainPca,TrainLabel)
2   %% The Method Defines Number Of Directions Inwhich 2DPCA Should Be Implemented
3   switch method
4      case '2DPCA'
5         disp('Testing Two Dimensional Principal Algorithm');
6         ImTestPca=cell(1,1);
7         theta=0;
8      case 'Alt2DPCA'
9         disp('Testing Alternative Two Dimensional Principal Algorithm');
10        ImTestPca=cell(1,1);
11        theta=90;
12     case '2D2DPCA'
13        disp('Testing 2 Directional Two Dimensional Principal Algorithm');
14        ImTestPca=cell(1,2);
15        theta=[0,90];
16     case '4D2DPCA'
17        disp('Testing 4 Directional Two Dimensional Principal Algorithm');
18        ImTestPca=cell(1,4);
19        theta=0:180/4:3*180/4;
20        %theta=[theta,90];
```

```
21      case '6D2DPCA'
22          disp('Testing 6 Directional Two Dimensional Principal Algorithm');
23          ImTestPca=cell(1,6);
24          theta=0:180/6:5*180/6;
25          %theta=[theta,90];
26  End
27  %% Reading test images and projecting them on projection vectors
28  disp('Reading All Test Images & their Calculating Feature Matrix')
29  TestLabel=cell(1,length(dir2));
30  NumOfEig=size(Vprj{1},2);
31  for k=1:length(theta)
32      disp([num2str(floor(k/length(theta)*100)),' Percent...'])
33      I=imread(['./test/',dir2(1).name]);
34      I=imrotate(I,theta(k),'bicubic');
35      TestPca=zeros(size(I,1),NumOfEig,length(dir2));
36      Vproj=Vprj{k};
37  for i=1:length(dir2)
38      ImName=dir2(i).name;
39      I=imread(['./test/',ImName]);
40      TestLabel{i}=ImName(1:5);
41      I=imrotate(I,theta(k),'bicubic');
42      I=double(I);
43      I(I>=230)=0;
44      TestPca(:,:,i)=I*Vproj;
45  End
46  ImTestPca{k}=TestPca;
47  End
48  disp('Feature matrix of Test Images Calculated!')
49  pause(.1);
50  disp('Classifying The Testing Samples...')
51  %% Classifying Test Samples Based on the Distance between each test feature matrix and traing feature
    matirixes
52  [Error,PredictedLabel,TestLabel,Mask]=ClassifySamples(ImTestPca,TestLabel,ImTrainPca,TrainLabel);
```

In this program, the value of ***theta*** is initialized in lines 3 to 26 based on the algorithm selected by the user. In lines 31 to 47, we multiply each image in the projection matrix and obtain the corresponding feature matrices in all the required directions and store them in the ***ImTrainPca*** cell.

In line 52, the ***ClassifySamples*** function is called. This function receives the cell containing ***ImTestPca*** feature Matrices, as well as the ***ImTrainPca*** feature Matrix cell along with the Test and Training Sample Labels as input, and Error and Prediction Labels. This function Returns the test sample as output.

10.5. ClassifySamples function

```
1   function
    [Error,PredictedLabel,TestLabel,Mask]=ClassifySamples(ImTestPca,TestLabel,ImTrainPca,TrainLabel)
2   %% This Function Calculates the Match Score between each test sample and all training samples
3   PredictedLabel=cell(1,length(TestLabel));
4   NumOfTest=size(ImTestPca{1},3);
5   NumOfTrain=size(ImTrainPca{1},3);
6   NumOfAngles=length(ImTestPca);
7   unit=floor(NumOfTest/10);
8   %% for each sample test, "MatchScore" is a (NumOfAngles)*(NumOfTrain) matrix
9   %each row of MatchScore score is the distances between each test sample and
10  %all traing samples in a particular direction
11  for i=1:NumOfTest
12      if mod(i,unit)==0
13          disp([num2str(floor(i/NumOfTest*100)),' Percent...']);
14      end
15      MatchScore=zeros(NumOfAngles,NumOfTrain);
16      for k=1:NumOfAngles
17          TestImages=ImTestPca{k};
18          TrainImages=ImTrainPca{k};
19          ImTest=TestImages(:,:,i);
20          for j=1:NumOfTrain
21              ImTrain=TrainImages(:,:,j);
22              %the distance is calculated here
23              MatchScore(k,j)=sum(sqrt(sum((ImTest-ImTrain).^2)));
24          end
25      end
26      MatchScoreMean=repmat(mean(MatchScore')',1,NumOfTrain);
27      MatchScore=MatchScore-MatchScoreMean;
28      MatchScores=sum(MatchScore,1);
29      %sum of match scores for all directions
30      [MInVal,MinInd]=min(MatchScores);
31      %finding the nearest training sample
32      PredictedLabel{i}=TrainLabel{MinInd};
33  end
34  %%
35  disp('Calculating Error Rate...')
36  %calculate the error rate by seeing how many elements of "PredictedLabel"
37  %and "TestLabel" are different
38  [Error,Mask]=ErrorCal(PredictedLabel,TestLabel);
```

In line 11, the loop is written on the number of test samples. For each test sample, a $\boldsymbol{k} \times \boldsymbol{N}$ matrix called $\boldsymbol{MatchScore}$ is created, where k is the number of directions and N is the number of samples in the training set. Each row of this matrix is a test sample's distance from all N instruction samples in a particular direction. In line 23, this Euclidean distance is obtained between the i^{th}-sample from the test set and the j^{th}-sample from the train set.

In line 28, all the $\boldsymbol{Matchscore}$ obtained in all directions are added together. Finally, in line 30, the nearest neighbor from the training set to the test sample is obtained and the label of the training sample is assigned to the test sample.

After completing these steps, we reach line 38 for all test samples, in which the $\boldsymbol{ErrorCal}$ function is called. This function compares the predicted labels of PredictedLabel samples with the original TestLabel labels, and based on the degree of contradiction between those two, the error rate is obtained.

10.6. ErrorCal function

This function obtains the error rate based on the number of contradictions between the original labels and the predicted labels.

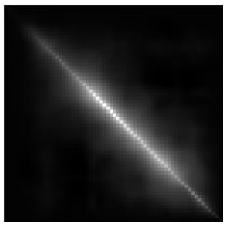
```
1   function [Error,Mask]=ErrorCal(PredictedLabel,TestLabel)
2   %% Error Rate is calculated
3   True=0;
4   False=0;
5   Mask=zeros(1,length(PredictedLabel));
6      for i=1:length(PredictedLabel)
7      if strcmp(PredictedLabel{i},TestLabel{i})
8         %Mask(i) is 1 if the predicted class for i'th sample is correct
9         Mask(i)=1;
10        True=True+1;
11     Else
12        False=False+1;
13     End
14  End
15  Error=False/(True+False);
```

11. Simulation Results

Before evaluating the proposed algorithm and comparing it with other previous algorithms, I first show the results of the simulations in calculating the projection vectors and extracting the feature vectors and calculating the distance between the test samples from all the training samples.

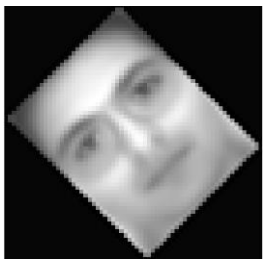
The training image folder contains 1170 images of 50 different men and 40 women. 13 images of faces are available from each of these individuals under different conditions. Using the training set of images, we obtain the covariance matrix for the zero rotation angle (non-rotation) and from it we obtain the matrix containing the projection vectors. The following figure shows the matrix of the averaged images and their covariance matrix.

The matrix of the averaged images with dimensions of 50 by 40 for $\boldsymbol{angle} = 0$

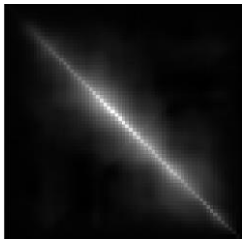


Covariance matrix images with dimensions of 40 by 40 for $\boldsymbol{angle} = 0$

We repeat the same process for other angles. For example, for a 45-degree angle, the the matrix of the averaged images and their covariance matrix would be as follows:

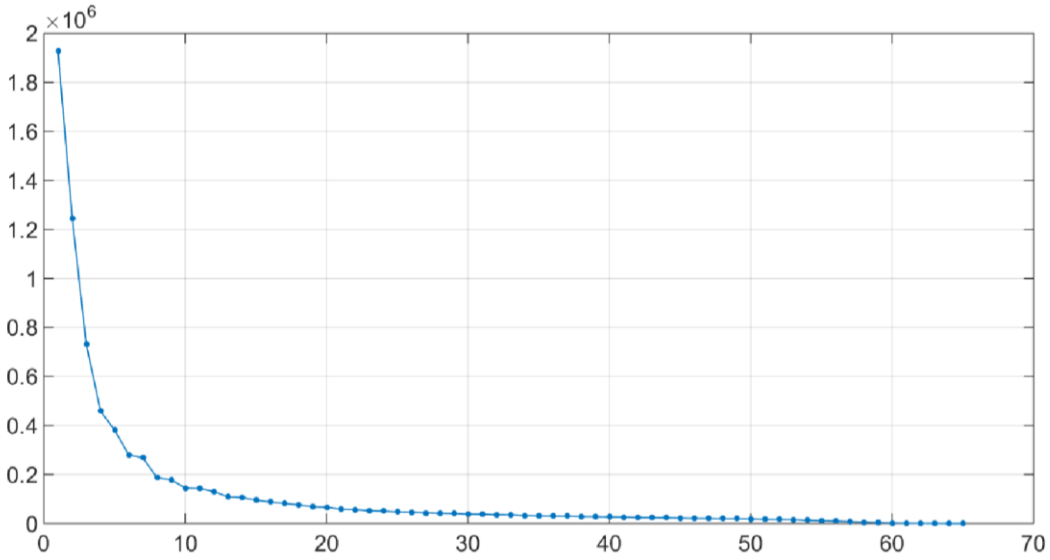


The matrix of the averaged images with dimensions of 50 by 40 for $\boldsymbol{angle} = 45$

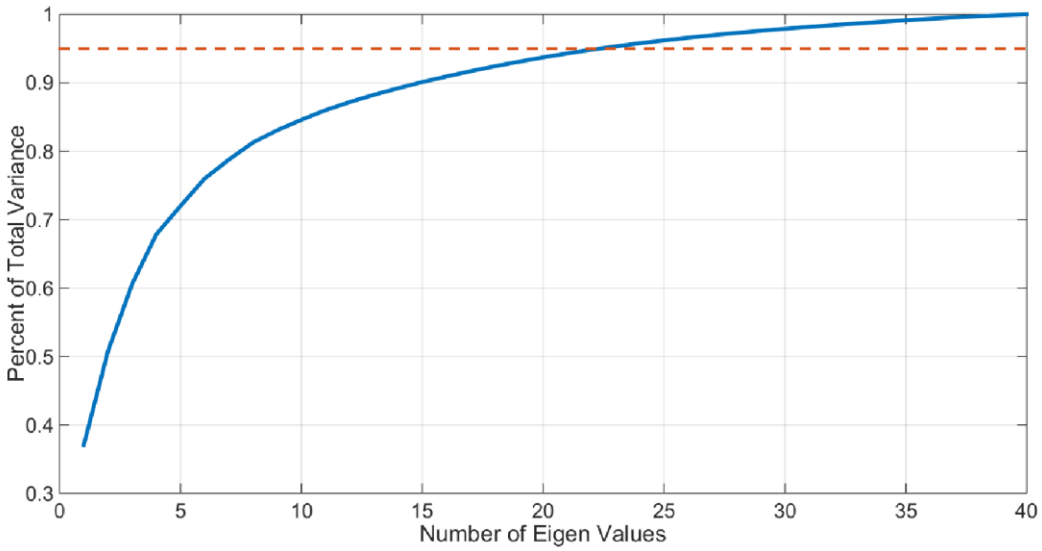


Covariance matrix images with dimensions of 40 by 40 for $\boldsymbol{angle} = 45$

In the next step, we extract the eigen vectors of the covariance matrices for each angle. This is done by the $\boldsymbol{ProjectionMatrix}$ function. The eigen values of the corresponding covariance matrix with a rotation angle of 0, after sorting from large to small, are shown in the figure below:



As can be seen in this figure, a large number of eigen values are very small. As a result, an accumulative diagram can be used to determine the number of the effective eigen vectors.



The dotted line in this figure shows the amount of 0.95 of the total energy or the variance of the samples. We see that only 20 eigen vectors are enough to keep the sample variance at 95%.

Consequently, we extract ***d*** vectors corresponding to the largest eigen values as projection vectors. In the next step, we will map all the pictures of the training set on the vectors and get the matrix of features. Figure below shows a picture of an AR database with its feature matrix for d = 20.

This image can be reconstructed using the feature matrix. The reconstructed images for a number of different eigen vectors are shown in the following figure:

reconstructed Image with 6 feature vectors

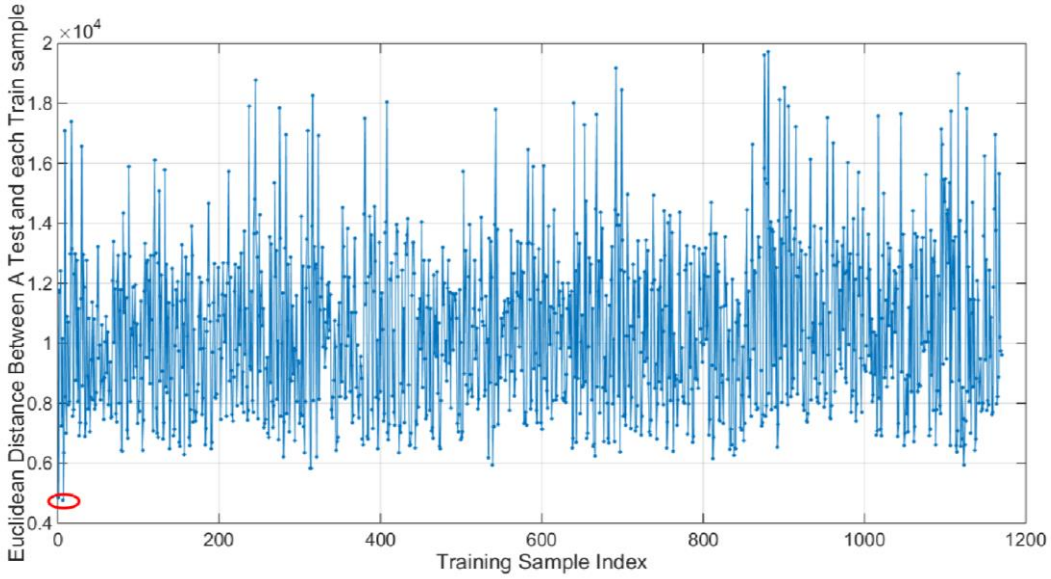
reconstructed Image with 12 feature vectors

reconstructed Image with 20 feature vectors

We calculate the feature matrix for all the images in the train set and save them. Then, in order to identify the label of a test sample, we calculate its feature matrix and the distance of it from the feature matrix of all other training samples. We label a sample of the training set the same as the one that has the shortest distance from the test sample.

The following figure shows the test sample distance (belonging to the first category (person number 1 in the database)) from all 1170 samples in the training set.

There are 90 classes and 13 training samples per class. As shown in this diagram, the test sample is the shortest distance from one of the first 13 samples in the training set, and the category for this training sample is number one. As a result, the class predicted for this test sample will be class one, and the sample will be categorized correctly.



The performance of the proposed algorithm is compared with the two-dimensional principal component analysis (2DPCA) algorithms, the alternative alternative analysis (2DPCA), and the two-dimensional principal component analysis (2D2DPCA).

We randomly separate half of the database images for testing and the other half for training. The MD2DPCA algorithm is tested and evaluated in two different modes for angles C = 4 and angles C = 6. We evaluate each algorithm for the number of different eigen vectors. For this purpose, we use the ***RunAlgorithms*** function.

The table below shows the results of the proposed algorithm (MD2DPCA) for the number of different eigen vectors compared to the other three algorithms:

number of feature vectors	6	8	10	12	14	16	20
2DPCA	0.5162	0.535	0.5521	0.5752	0.5846	0.5889	**0.5880**
Alternative 2DPCA	0.5034	0.5171	0.5402	0.5684	0.5906	0.5889	**0.6043**
2D2DPCA	0.53	0.5402	0.5590	0.5795	0.5906	0.5990	**0.6068**
4D2DPCA	0.5291	0.5403	0.5547	0.5809	0.5817	0.5995	**0.6094**
6D2DPCA	**0.5350**	**0.5513**	**0.5667**	**0.5872**	**0.5915**	**0.6034**	**0.6121**

As can be seen in this table, as the number of eigen vectors increases, the detection rate of all 5 methods increases. In this table, we see that for each number of feature vectors, the 6D2DPCA algorithm gives a higher rate than other algorithms, and this is because this algorithm extracts image properties from 6 different directions and creates better features for the image. Subsequently, its detection rate is higher than other models.