May 3, 2015

ECE407 Project

Implementation of the Fisher Linear Discriminant (FLD) based algorithm for Face Recognition

Instructor: Shu Wang

Arindam Bose
UIC ID: 665387232

Chirag Agarwal
UIC ID: 670916062

Introduction

Within the last several years, numerous algorithms have been proposed for face recognitions, while much progress has been made toward recognizing faces under small variations in facial expression and pose, reliable techniques for recognition under more extreme variations have proven elusive.

In this project we tried to implement the approach taken in [1]. Our approach to face recognition exploits two observations:

1. All of the images of a Lambertian surface, taken from a fixed viewpoint lie in a 3D linear subspace of the high-dimensional image space [2].
2. Because of different facial expressions the above observation does not exactly hold. In practice, certain regions of the face may have variability from image to image that often deviates significantly from the liner subspace and are less reliable for recognition.

We make use of these observations by finding a linear projection of the faces from the high-dimensional image space to a significantly lower dimensional feature space which is insensitive to variation in facial expression. We choose projection directions that are nearly orthogonal to the within-class scatter, projecting away variations in facial expressions while maintaining discriminability. Our method Fisherfaces, a derivative of Fisher's Linear Discriminant (FLD) maximizes the ratio of between-class scatter to that of within-class scatter.

Our Method

1. We started by creating a well-defined database of 40 different classes each having 10 samples (expression faces). We reshape all 2D images of the training database into 1D column vectors. Then we put these 1D column vectors in a row to construct a 2D matrix D. Each column of D is a training image, which has been reshaped into a 1D vector. Also, lets P is the total number of $M \times N$ training images and C is the number of classes.
2. Now suppose D_i is a training image, which has been reshaped into a 1D vector. At first, centered D_i is mapped onto a $(P - C)$ linear subspace by V_{PCA} matrix: $Z_i = V_{PCA} * (D_i - m_D)$ where $m_D = \frac{1}{P}\sum_{i=1}^{P} D_i$
3. Then, Z_i is converted to Y_i by projecting onto a $(C - 1)$ linear subspace, so that images of the same class (or person) move closer together and images of different classes move further apart: $Y_i = {V_{Fisher}}^T * {V_{PCA}}^T * (D_i - m_D)$
4. Finally we compare two faces by projecting the images into facespace and measuring the Euclidean distance between them.

Result:

We used ORL face database as our training dataset which consists of 10 samples of 40 classes (Fig. 1). We chose one face from each class for the test datasets (Fig. 2).

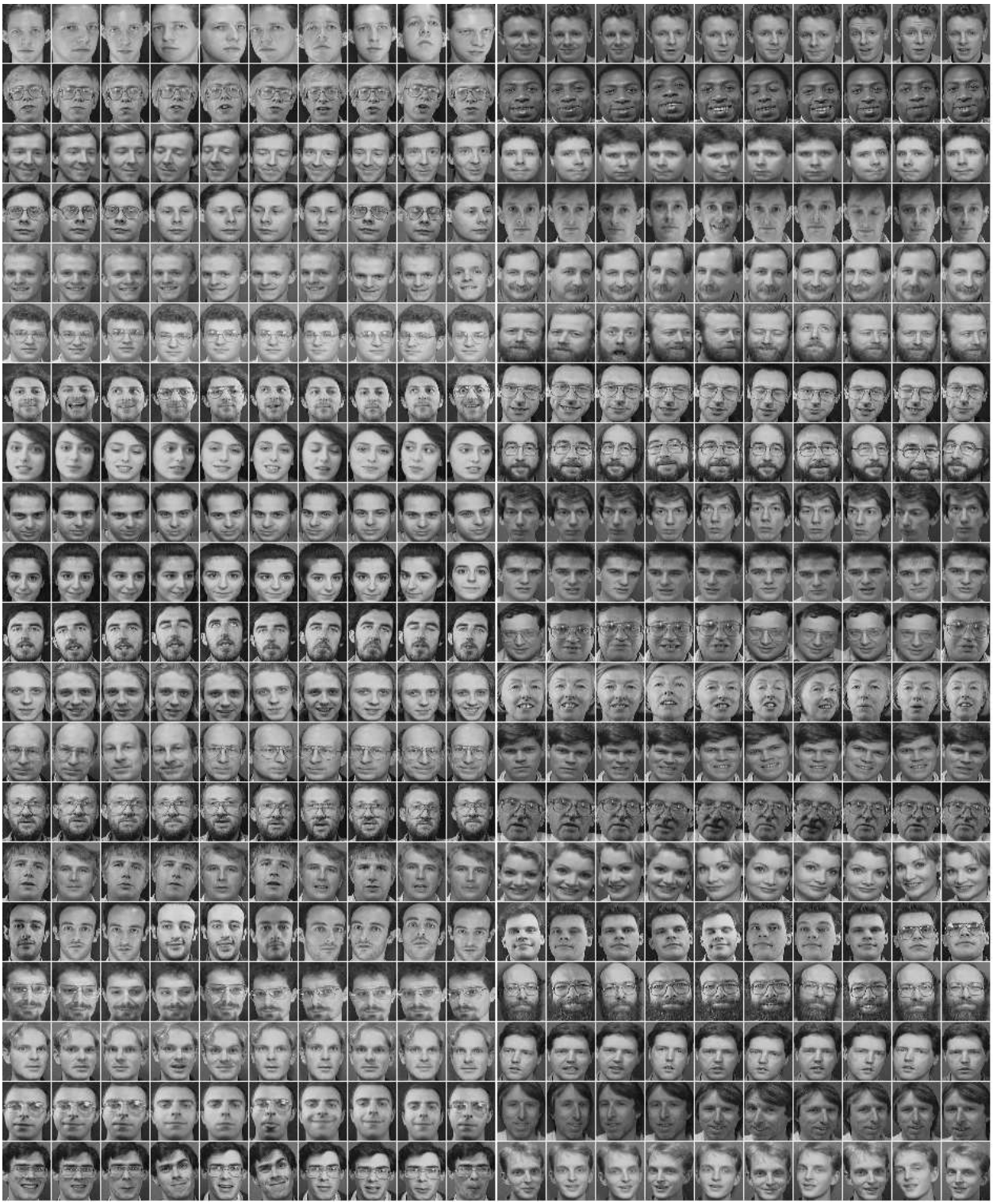

Fig. 1: Training Database

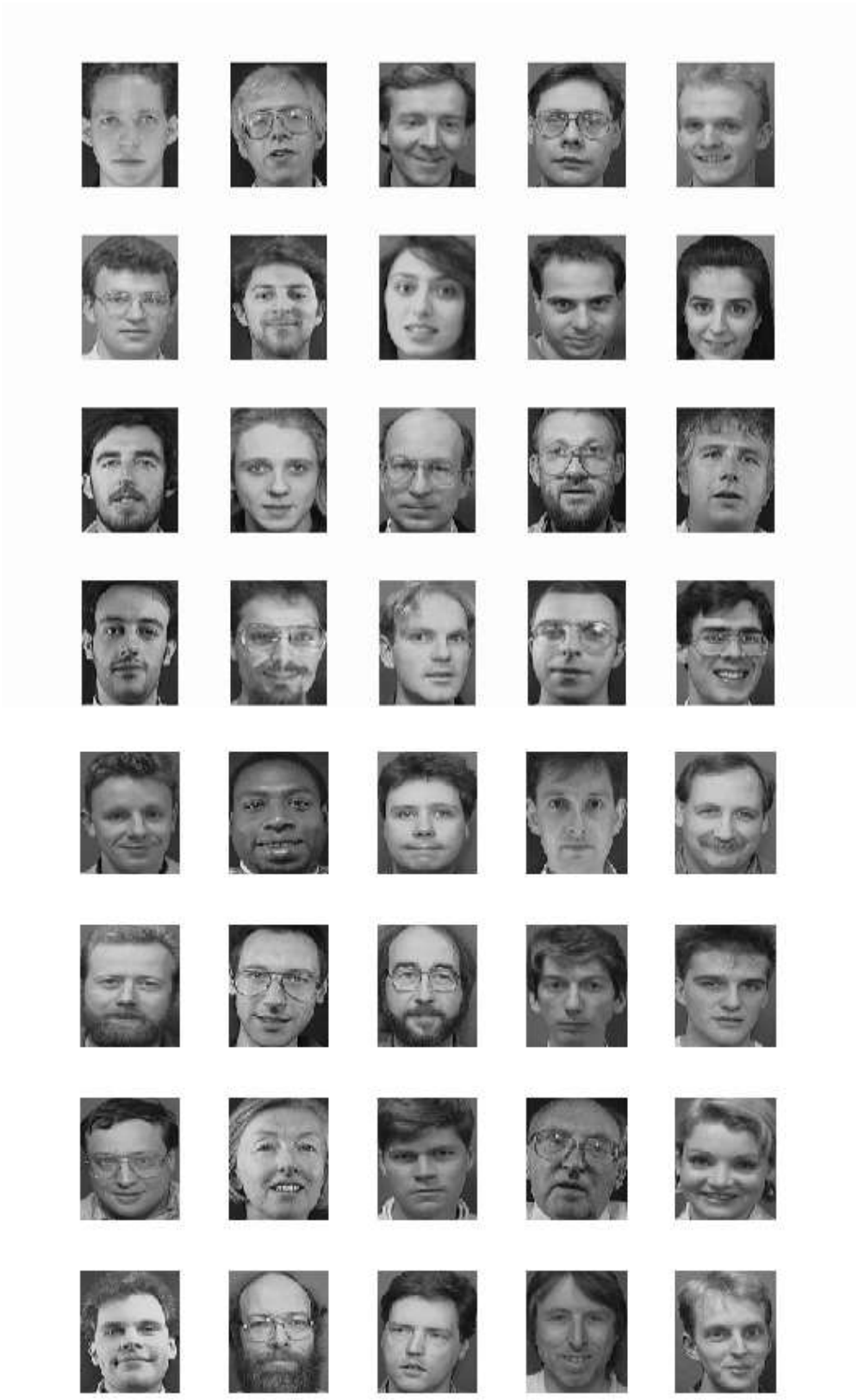

Fig.2: Test Database

Now we divide our test procedure in two cases:

1. Inclusive Test Case: In this case, all the test faces are itself present in the training database.
2. Exclusive Test Case: We remove all the faces which are chosen as test faces from training database. In our case we took one random face from each class and remove them from the training database. If the test face is matched with the correct class, we call it a true positive and otherwise we call it a false positive.

In Inclusive Test Case, we got 100% true positive i.e. all the faces matched with its corresponding class (Fig. 3).

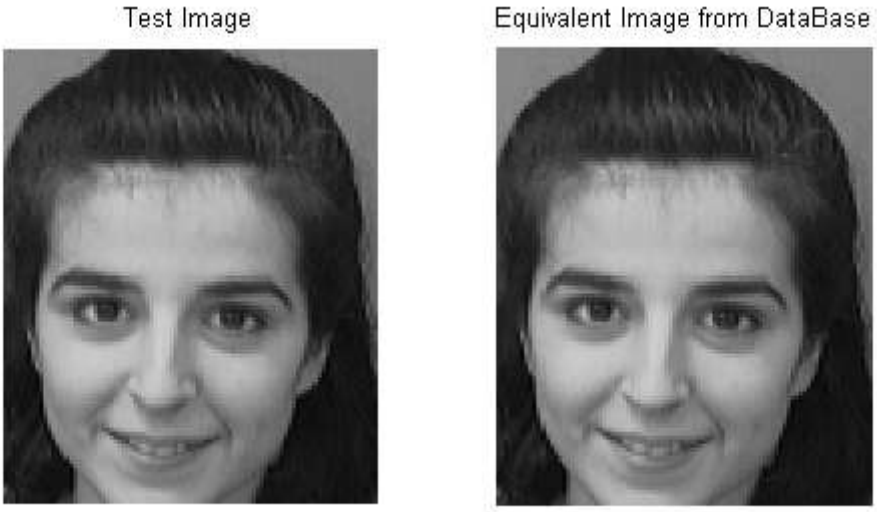

Fig.3: Inclusive Test Case for test face: 10 (True Positive)

But in Exclusive Test Case, we got 73% true positive i.e. 29 out of 40 faces are correctly matched with their corresponding class. Fig. 4 and Fig. 5 are the true positive and false positive matches respectively.

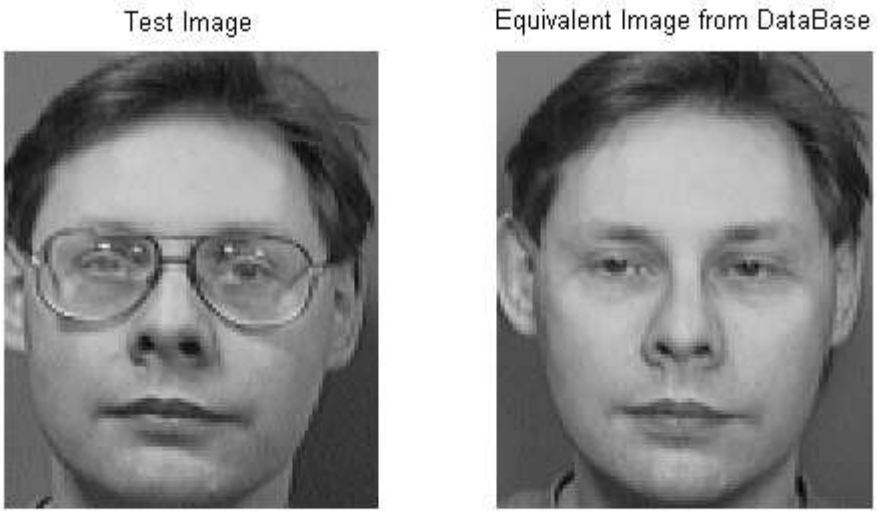

Fig.4: Exclusive Test Case for test face: 4 (True Positive)

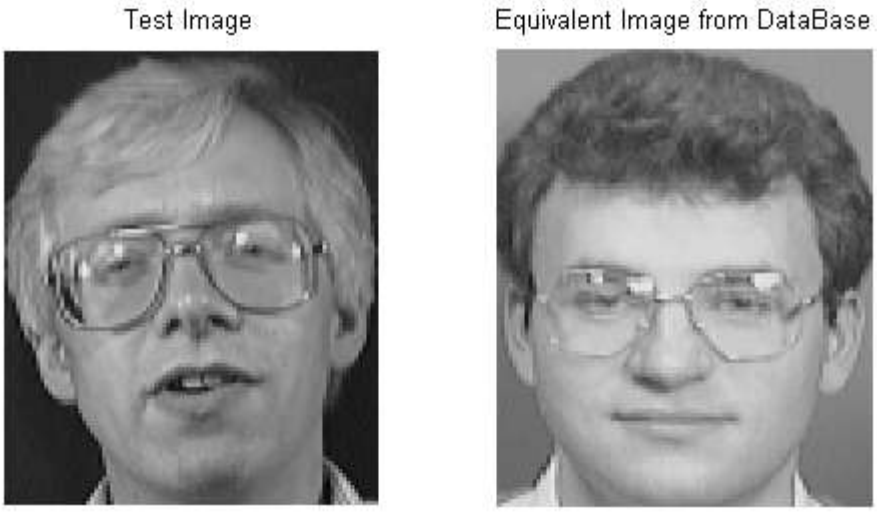

Fig.5: Exclusive Test Case for test face: 2 (False Positive)

MATLAB Code:

```
% A FLD-based face recognition system (Fisherface method)
clear all;
clc;
close all;

%% Read Training and Test images path
trainDatabasePath = uigetdir(pwd, 'Select Training Database Path');
testDatabasePath = uigetdir(pwd, 'Select Test Database Path');
% trainDatabasePath = 'TrainingFaces';
% testDatabasePath = 'TestFaces';

dialogTitle = 'Input Test Image';
dialogPrompt = {'Enter Test Image Name (between 1 to 40):'};
defaultText = {'1'};

testImageName  = inputdlg(dialogPrompt, dialogTitle, 1, defaultText);
if str2double(testImageName{1}) >= 1 && str2double(testImageName{1}) <= 40
    testImageName = fullfile(testDatabasePath, [testImageName{1} '.pgm']);
    testImage = imread(testImageName);
else
    error('Input should be between 1 and 40');
end

%% Create database
[dataBase, r, c] = createDatabase(trainDatabasePath);

%% Determine the most discriminating features between images of faces.
[meanDatabase, eigenFaces, V_Fisher, projectedImagesFisher] =
fisherfaceCore(dataBase);

%% Recognizing the face from training samples
[distMin , index] = recognition(testImage, meanDatabase, eigenFaces,
V_Fisher, projectedImagesFisher);

%% Print test image and equivalent recognized face from database
figure,
subplot(1,2,1), imshow(testImage);
title('Test Image');
subplot(1,2,2), imshow(uint8(reshape(dataBase(:,index), r, c)));
title('Equivalent Image from DataBase');

%% Print the minimum distance between the two images
disp(['Matched index is : ' num2str(index)]);
disp(['Minimum distance is : ' num2str(distMin)]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [dataBase, row, col] = createDatabase(trainDatabasePath)
% Creates the training database which consists of 10 different faces of 40
% different persons making total of 400 faces.
%
% Argument:  trainDatabasePath   - Path of the training database
%
```

```
% Returns:   dataBase            - A 2D matrix, containing all 1D image
vectors. The length of 1D column vectors is MN and dataBase will be a MNxP 2D
matrix.
%            row                 - number of rows (M) in a single image
%            col                 - number of columns (N) in a single image
%
%% Gathering the names of all the image files and sort them
trainFiles = dir(trainDatabasePath);
isubFile = ~[trainFiles(:).isdir];
nameFiles = {trainFiles(isubFile).name};
num = zeros;
for i = 1:size(nameFiles,2)
    str = nameFiles(i);
    str  = sprintf(str{1}, '%s*');
    num(i)  = sscanf(str, '%d*');
end
[dummy, index] = sort(num);
nameFiles = nameFiles(index);

%% Construction of 2D matrix from 1D image vectors
dataBase = [];
for i = 1 : size(nameFiles,2)
    fileName = nameFiles(i);
    filePath = fullfile(trainDatabasePath, fileName{1});
    img = imread(filePath);
    [row, col] = size(img);
    temp = reshape(img, row*col,1);
    dataBase = [dataBase temp];
end
dataBase = double(dataBase);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [meanDatabase, eigenFaces, V_Fisher, projectedImagesFisher] =
fisherfaceCore(dataBase)
% Uses Principle Component Analysis (PCA) and Fisher Linear Discriminant
(FLD) to determine the most
% discriminating features between images of faces.
%
% Argument: dataBase              - (MNxP) A 2D matrix, containing all 1D
image vectors. All of 1D column vectors have the same length of M*N
%                                         and dataBase will be a MNxP 2D
matrix.
%
% Returns:  meanDatabase          - (MNx1) Mean of the training database
%           eigenFaces            - (M*Nx(P-C)) Eigen vectors of the
covariance matrix of the training database
%           V_Fisher               - ((P-C)x(C-1)) Largest (C-1) eigen
vectors of matrix J = inv(Sw) * Sb
%           projectedImagesFisher  - ((C-1)xP) Training images, which are
projected onto Fisher linear space
%
%% Number of Classes and Class population
classPopulation = 10;
classCount = (size(dataBase,2))/classPopulation;
P = size(dataBase,2);
```

```
%% Calculating the mean image
meanDatabase = mean(dataBase,2);

%% Calculating the deviation of each image from mean image
A = dataBase - repmat(meanDatabase, 1, P);

%% Calculating the eigenvectors, eigen values and taking the larger values
[eigenVectors, eigenValues] = eig(A'*A);
eigenVectors = eigenVectors(:, 1:P-classCount);

%% Calculating the eigenfaces
eigenFaces = A * eigenVectors;

%% Projecting centered image vectors onto eigenspace
projectedImagesPCA = [];
for i = 1 : P
    temp = eigenFaces'*A(:,i);
    projectedImagesPCA = [projectedImagesPCA temp];
end

%% Calculating the mean of each class in eigenspace
meanPCA = mean(projectedImagesPCA,2);
m = zeros(P-classCount, classCount);
Sw = zeros(P-classCount, P-classCount);
Sb = zeros(P-classCount, P-classCount);

for i = 1 : classCount
    m(:,i) = mean(( projectedImagesPCA(:, ((i-1) * classPopulation+1) :
i*classPopulation)), 2)';

    S  = zeros(P-classCount, P-classCount);
    for j = ((i-1)*classPopulation+1) : (i*classPopulation)
        S = S + (projectedImagesPCA(:,j) - m(:,i)) * (projectedImagesPCA(:,j)
- m(:,i))';
    end

    Sw = Sw + S;
    Sb = Sb + (m(:,i)-meanPCA) * (m(:,i)-meanPCA)';
end

%% Calculating Fisher discriminant basis
[J_eigenVector, J_eigenValue] = eig(Sb,Sw); % Cost function J = inv(Sw) * Sb
J_eigenVector = fliplr(J_eigenVector);

%% Eliminating zero eigens and sorting in descend order
V_Fisher = J_eigenVector(:, 1:classCount-1);

%% Projecting images onto Fisher linear space
projectedImagesFisher = zeros(classCount-1, P);
for i = 1 : classCount*classPopulation
    projectedImagesFisher(:,i) = V_Fisher' * projectedImagesPCA(:,i);
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [distMin , index] = recognition(testImage, meanDatabase, eigenFaces,
V_Fisher, projectedImagesFisher)
% Compares two faces by projecting the images into facespace and measuring
the Euclidean distance between them.
%
% Argument:  testImage              - Path of the input test image
%            meanDatabase           - (MNx1) Mean of the training database
%            eigenFaces             - (MNx(P-1)) Eigen vectors of the
covariance matrix of the training database
%            V_Fisher               - ((P-1)x(C-1)) Largest (C-1) eigen
vectors of matrix J = inv(Sw) * Sb
%            projectedImagesFisher  - ((C-1)xP) Training images, which are
projected onto Fisher linear space
%
% Returns:   distMin                - Minimum distance between the nearest
image and the test image
%            index                  - index of the recognized image in the
training database.
%
%% Extracting the FLD features from test image
trainNumber = size(projectedImagesFisher,2);
[row, col] = size(testImage);
inputImage = reshape(testImage, row*col,1);
diffImage = double(inputImage) - meanDatabase;
projectedTestImage = V_Fisher' * eigenFaces' * diffImage;

%% Calculating Euclidean distances
dist = [];
for i = 1 : trainNumber
    q = projectedImagesFisher(:,i);
    temp = ( norm( projectedTestImage - q ) )^2;
    dist = [dist temp];
end

[distMin , index] = min(dist);
end
```

References:

[1] P. N. Belhumeur, J. Hespanha, and D. J. Kriegman. "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection." In ECCV (1), pages 45-58, 1996.

[2] A. Shashua, "Geometry and Photometry in 3D Visual Recognition," PhD thesis, Massachusetts Institute of Technology, 1992.

[3] The ORL Database of Faces: An archive of AT&T Laboratories, Cambridge hosted in conjunction with Cambridge University Computer Laboratory, The Digital Technology Group.