ÇANKAYA UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

MASTER THESIS

PCA BASED FACE RECOGNITION: AN APPLICATION

ÖZCAN TİLKİ

JANUARY 2014

Title of the Thesis : **PCA Based Face Recognition: an Application**

Submitted by **Özcan TİLKİ**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University

[illegible]

Prof. Dr. Taner ALTUNOK
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science

[illegible]

Assist.Prof.Dr. Murat SARAN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree Master of Science.

[illegible]

Assist.Prof.Dr. Reza HASSANPOUR
Supervisor

Examination Date: 16.01.2014

Examining Committee Members

Assist. Prof. Dr. Reza HASSANPOUR (Çankaya Univ.)	[illegible]
Prof. Dr. Mehmet Reşit TOLUN (Aksaray Univ.)	[illegible]
Assist. Prof. Dr. Abdul Kadir GÖRÜR (Çankaya Univ.)	[illegible]

STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name	: Özcan TİLKİ
Signature	: [illegible]
Date	:

ABSTRACT

PCA BASED FACE RECOGNITION: AN APPLICATION

TİLKİ, Özcan

M.S.c., Department of Computer Engineering

Supervisor: Assist. Prof. Dr. Reza HASSANPOUR

January 2014, 43 pages

Face recognition is one of the most important problems of the computer based systems. In literature, there are several methods used for face recognition such as holistic, local or hybrid methods. On the other hand, recent researches revealed that The Principle Component Analysis (PCA) is a useful method to solve image recognition and compression. Thus, it is possible to realize face recognition by using PCA method. In this study, a PCA based face recognition system was developed and suggested to use for different aims. In order to evaluate and apply face recognition, PCA analysis was used, and an algorithm has been developed in the research. Results were tested, and suggestions for further analysis have been given.

Key Words: Image processing; Face recognition; PCA.

ÖZ

PCA TABANLI YÜZ TANIMA: BİR UYGULAMA

TİLKİ, Özcan

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Danışman: Yrd. Doç. Dr. Reza HASSANPOUR

Ocak 2014, 43 sayfa

Yüz tanımlama, bilgisayar tabanlı sistemlerin en önemli problemlerinden birisidir. Literatürde yüz tanımlamaya ilişkin holistik, lokal ya da hibrit yöntemler gibi bazı yöntemler mevcuttur. Öte yandan son yıllarda yapılan araştırmalar göstermektedir ki, Temel Bileşenler Analizi (PCA) imaj tanımlamada ve sıkıştırmada kullanışlı bir yöntemdir. Dolayısıyla, yüz tanımlama işlemi de PCA yöntemi kullanılarak gerçekleştirilebilir. Bu çalışmada, PCA tabanlı bir yüz tanımlama sistemi geliştirildi ve farklı alanlarda kullanılmak üzere önerildi. Yüz tanımlama işleminin gerçekleştirilebilmesi için, PCA analizi kullanıldı ve araştırmada bir algoritma geliştirildi. Sonuçlar test edildi ve ileri araştırmalar için bazı önerilerde bulunuldu.

Anahtar Kelimeler: Görüntü işleme; Yüz tanımlama; PCA.

ACKNOWLEDGMENTS

I would like to my sincere gratitude to my Advisor Assist. Prof. Dr. Reza Hassanpour for his wisely guidance, suggestions and supports for the each steps of the research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

After development of the computer and computer based systems, their using areas have been increased based on demand for the daily life. Computers ease our daily life, and change it day by day. One of the important demand for the computer based systems is face detection and recognition for security systems. Image processing and image compressing are important topics of the computer based systems. In application, face detection and recognition has many application areas mainly in security systems.

CHAPTER 2

FACE RECOGNITION

2.1. Face Recognition Concept

In this part of the study, face recognition and face detection steps have been examined in order to understand its usage area, history and current motivation.

2.1.1. Face recognition

Face recognition remains as an unsolved problem and a demanded technology. [1] The major purpose of face recognition is to identify the humans from data acquired from their faces, as humans do. A good recognition system has to be fully automatic and robust enough for real life conditions such as illumination, rotations, expressions and occlusions. [2] A generic face recognition system includes following steps. [1]

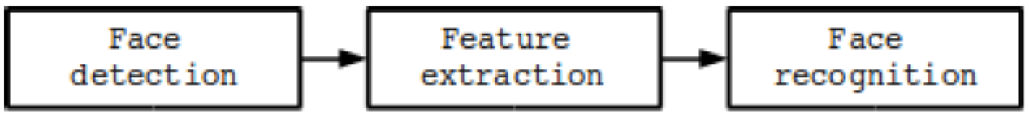

Figure 2. 1: A Generic Face Recognition System [1].

2.1.2. Face detection

Face detection must deal with several well known challenges. They are usually present in images captured in uncontrolled environments, such as surveillance video systems. These challenges can be attributed to some factors [1]:

Pose variation: Face detection is an ideal scenario where only the front will include images.

Feature occlusion: Beards, glasses or hats, the presence of elements such as high variability introduced.

Facial expression: different facial gestures cause variety of facial features.

Imaging conditions: Different cameras and environmental conditions that affect the appearance of the face, may affect the quality of an image.

There are some problems closely related to face detection besides feature extraction and face classification. For instance, face location is a simplified approach of face detection. Its goal is to determine the location of a face in an image where there's only one face. We can differentiate between face detection and face location, since the latter is a simplified problem of the former. [1]

2.2. History of Face Recognition

Engineering started to show interest in face recognition in the 1960's. The first research on this subject was Woodrow W. Bledsoe. In 1960, Bledsoe, along other researches, started Panoramic Research, Inc., in Palo Alto, California. The majority of the work done by this company involved AI-related contracts from the U.S. Department of Defense and various intelligence agencies. Bledsoe, Helen Chan and Charles Bisson, worked on using computers to recognize human faces during 1964 and 1965. In the 1980's there were a diversity of approaches actively followed, most of them continuing with previous tendencies. Some works tried to improve the methods used measuring subjective features. For instance, Mark Nixon presented a geometric measurement for eye spacing. [1]

2.3. Methods Used for Face Recognition

Although there are many image processing methods, only some of them are able to solve face recognition problem. Methods used for face recognition may be classified as below [3]:

i. Holistic Methods

ii. Local Methods

iii. Hybrid methods

2.3.1. Holistic methods

There have been many generic face recognition algorithms proposed. Among these, there are two algorithms that have had a very large impact on the face recognition research community and they have inspired countless studies. These are eigenfaces, and Fisherfaces. [3]

1.3.2. Local methods

Face recognition based on local facial regions has attracted a significant amount of interest because local features are believed to be more robust to the variations of facial expression, illumination and occlusions. Approaches that utilize local regions either use salient regions or they just partition the face image into rectangular blocks. [3]

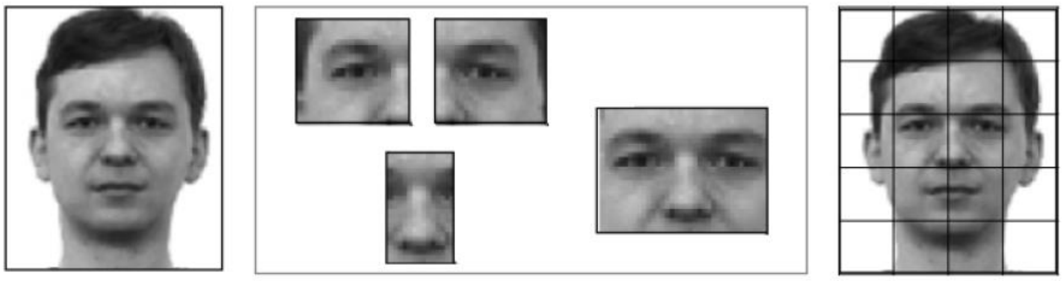

Figure 2. 2: Holistic Approach for Face Recognition

1.3.3. Hybrid methods

Hybrid methods are methods including two or more face recognition approaches. Since singular approaches can not give sufficient solution to the face recognition problem, researchers have had a tendency to combine global and local information extracted using one or more different types of features to improve the recognition performance. [3]

2.4. Principle Component Analysis (PCA)

PCA is one of the most successful techniques that have been used in image recognition and compression. PCA is a statistical method under the broad title of factor analysis. The purpose of PCA is to reduce the large dimensionality of the data space to the smaller intrinsic dimensionality of feature space, which are needed to describe the data economically. This is the case when there is a strong correlation between observed variables. [4]

In Figure 3, suppose that the triangles represent a two variable data set which we have measured in the X-Y coordinate system. The principal direction in which the data varies is shown by the U axis and the second most important direction is the V axis orthogonal to it. If we place the U-V axis system at the mean of the data it gives us a compact representation. If we transform each (X; Y) coordinate into its corresponding (U; V) value, the data is de-correlated, meaning that the co-variance between the U and V variables is zero. For a given set of data, principal component analysis finds the axis system defined by the principal directions of variance. [5]

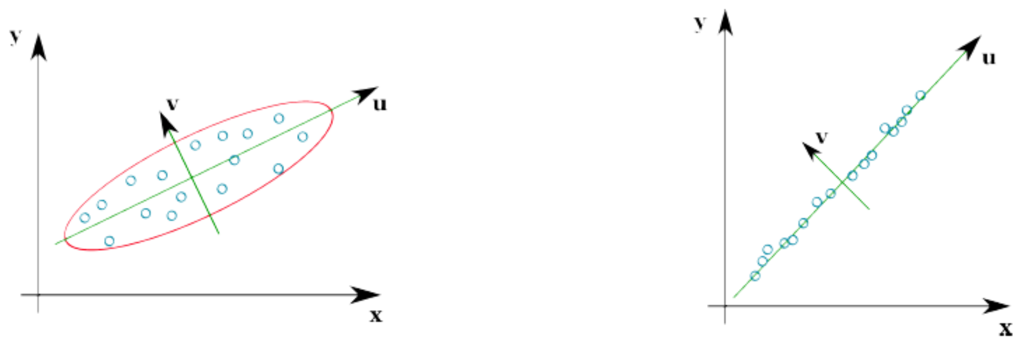

Figure 2. 3: PCA for Data Presentation (left) and Dimension Reduction (right) [6]

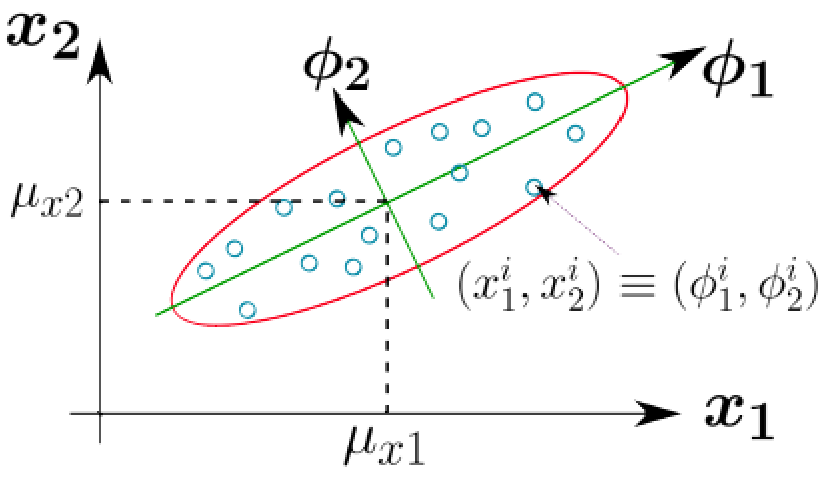

Figure 2. 4: The PCA Transformation [6]

2.4.1. Eigen value

Linear equations Ax=b come from steady state problems. In dynamic issues, Eigenvalues have vital importance. The solution of du=dt D Au depends on time—increasing, decreasing or oscillating. It is not possible to find by elimination. [7]

The basic equation is Ax D x. The number is an eigenvalue of A.

An Eigenvalue of the n × n matrix A which is called, there is a nontrivial solution x of Ax = λx. Such an x is called an eigenvector corresponding to the eigenvalue λ. [8]

Let T : R → R, Then if Ax = λx, it follows that T(x) = λx. The equation expresses that if x is an eigenvector of A, then the image of x under the transformation T is a scalar multiple of x – and the scalar involved is the corresponding eigenvalue λ. In other words, the image of x is parallel to x. Finding an equivalent equation in standard form is given below. [8]

$$Ax = \lambda x$$

$$Ax - \lambda x = 0$$

$$Ax - \lambda Ix = 0$$

$$(A - \lambda I)x = 0$$

Thus x is an eigenvector of A corresponding to the eigenvalue λ if and only if x and λ satisfy $(A-\lambda I)x = 0$.

The eigenvalue λ means whether the private vector x is stretched or shrunk or reversed or left unchanged—when it is multiplied by A. We may find λ= 2 or 1/2 or -1 or 1. The eigenvalue λ is zero. Then Ax = 0x means that this eigenvector x is in the nullspace. If A is the identity matrix, every vector has Ax= x. All vectors are eigenvectors of I . All eigenvalues "lambda" are λ=1. This is unusual to say the least. Most 2 by 2 matrices have two eigenvector directions and two eigenvalues. We will show that det.A –λI) = 0. [7]

2.5. Face Recognition Using PCA

Face recognition is one example where principal component analysis has been extensively used, primarily for reducing the number of variables. Let us consider the 2D case where we have an input image, and wish to compare this with a set of data base images to find the best match. We assume that the images are all of the same resolution and are all equivalently framed. Each pixel can be considered a variable thus we have a very high dimensional problem which can be simplified by PCA. [5] The simple explaining for the face recognition can be expressed as below:

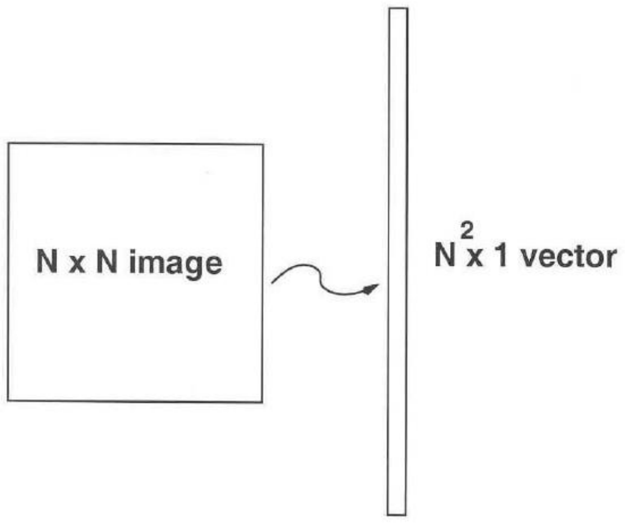

Figure 2. 5: Simple PCA Approach for Face Recognition [9]

Once the eigenfaces have been computed, several types of decision can be made depending on the application. What we call face recognition is a broad term which may be further specified to one of following tasks [4]:

i. Identification where the labels of individuals must be obtained,

ii. Recognition of a person, where it must be decided if the individual has already been seen,

iii. Categorization where the face must be assigned to a certain class.

$$= \begin{bmatrix} 150 & 152 & \cdot & 151 \\ 131 & 133 & \cdot & 72 \\ \cdot & \cdot & \cdot & \cdot \\ 144 & 171 & \cdot & 67 \end{bmatrix} \quad 128 \times 128$$

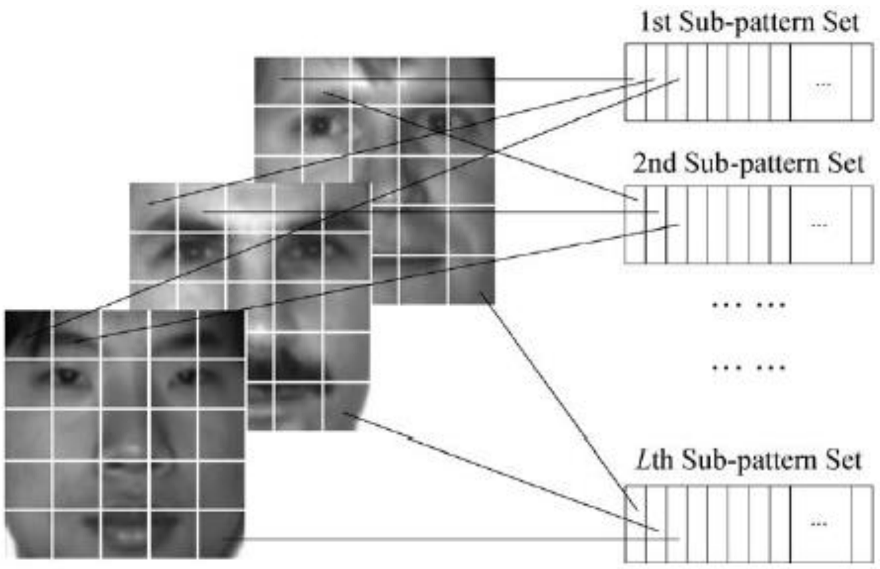

Figure 2. 6: Determinant of an Image in Face Recognition [11]

PCA computes the basis of a space which is represented by its training vectors. These basis vectors, actually eigenvectors, computed by PCA are in the direction of the largest variance of the training vectors. As it has been said earlier, we call them eigenfaces. Each eigenface can be viewed a feature. When a particular face is projected onto the face space, its vector into the face space describes the importance of each of those features in the face. The face is expressed in the face space by its eigenface coefficients. We can handle a large input vector, facial image, only by taking its small weight vector in the face space. This means that it is able to reconstruct the original face with some error, since the dimensionality of the image space is much larger than that of face space. [4]

2.6. Literature Review

Although there are many researches on face recognition and PCA method, there have been restricted researches on face recognition using PCA method. In this part of the study, some of important researches focusing on face recognition using PCA have been mentioned.

Zhou et al examined face recognition based on PCA image reconstruction and LDA. In the research, they used PCA and Linear Discriminant Analysis (LDA) for the face recognition. In their suggested method, they used the inner-classes covariance matrix for feature extraction as generating matrix, and eigenvectors from each person obtained. Afterwards, they obtained reconstructed images. The suggested method is effective on ORL face database. [11]

Wang et al proposed novel generalized PCA based face recognition algorithm in their research. They improved symmetrical image correction (SIC) and bit-plane feature fusion (BPFF) in order to improve the illumination robustness of the algorithm. Afterwards, they applied Generalized PCA to the virtual faces to achieve face recognition. Results of the illustration showed that, combined approach to face recognition algorithm is effective to reduce the differences in sensitivity and thus needed to achieve the same recognition rate by comparing the vector projection approach may be less than suggested. [12]

Hsieh and Tung developed a new hybrid approach depend on sub-pattern technique and whitened PCA for face recognition in their research. They combined sub-pattern technique with PCA I and PCA II respectively for the face recognition. The study showed that sub-pattern technique was useful for PCA I, but not useful for PCA II and PCA. Afterwards, they combined PCA II and Sp-PCA I for face recognition. The results of the experiments showed that the proposed approach gives more sophisticated recognition performance than that obtained using other traditional methods. [13]

Luh and Lin proposed a face recognition method with help of artificial immune networks based on PCA. They optimized antibodies of the immune networks using genetic algorithms. The results of the study showed that proposed method have better recognition rate in contrast with most of the developed methods. [14]

Kusuma and Chua proposed in their research image at the level of dependency between modalities are investigating an image recombination for face recognition. They account for the maximum amount of variance in images by finding the axis transformation of data into a more independent and distinctive facial expressions 2D and 3D images recombine. They also combine 2D and 3D face recognition methods. Results of the study face recognition system using image pixels recombination or other face recognition systems based on score level fusion showed a better performance. [15]

Kusuma and Chua proposed image at the level of dependency between modalities are investigating an image recombination for face recognition. They again combined 2D and 3D face recognition methods, and results were useful as in their research a year before. [16]

Oh et al proposed as one of the functional components of general face recognition system based polynomial radial basis function neural networks.Their system consisted of preprocessing and recognition module. They presented efficiency of PCA-LDA combined algorithm compared with other algorithms such as PCA, LPP, 2D-PCA and 2D-LPP. [17]

Zhang and Zhou proposed in their research an alternative 2DPCA which is working in the column direction of images after indicating that 2DPC line in the direction of the images actually works. ORL and FERET face databases, the experimental results of a sub-image display is set for the former than the latter needs a much reduced coefficient of the 2DPC, 2DPC get more showed the same or even higher recognition accuracy. [18]

Tan and Chen examined adaptively weighted sub-pattern PCA for face recognition in their research. In the research, hundred people (50 men and 50 women) in the experiment a total of 126 people in AR face database is selected. Used images per fourteen. ORL database of 400 images of 40 adults, includes 10 images per person in the Yale face database, 15 165 images of adult per capita, there are 11 images. AR and Yale databases frontal view images with different facial expression and lighting conditions are facing. In addition to these variations, the images in the database ORL facial details and head pose changes. Experiments on three standard face databases show that the method proposed in the research is competitive. [19]

Abeer et al examined multi-linear neighborhood preserving projection for face recognition in their research. In the research, they propose a novel method of supervised and unsupervised multi-linear neighborhood preserving projection for face recognition. They determined the number of subspaces derived by the method using the order of the tensor space. The performance of the supposed method in the research was analyzed using ORL, Ar and FERET. The results of the study showed that the MNPP outperforms the standard approaches in terms of the error rate. [20]

Banerjee et al proposed a frequency domain nonlinear correlation technique for face recognition under varying lighting conditions in their research. The techniques used in this study class, a particular subspace projection image correlation during the operation of an optimal filter and the optimum re-image correlation filter are based on the correlation between the phase. Performance improvement of image pixels was obtained by exploiting the point-wise nonlinearities. Further optimization of the correlation peak is maximized by minimizing the correlation energy is performed in the plane. [21]

Givens et al examined biometric performance evaluation for the face recognition using statistical methods. They research, Good the Bad and the Ugly Face of Battle, state-of-the-art face recognition algorithms still images used to test an elite set of facial recognition generalized linear mixed model analysis are shown. [22]

Li et al developed a method for face recognition using multi-scale Weber local descriptors (WLDs) and multi-level information fusion. In their research, they introduce a method including four steps as follows [23]:

1. Image partition

2. Feature extraction

3. Features measurement and

4. Voting

Their suggested model and its results showed effectiveness upon three popular data sets.

Luan et al examined face recognition with contiguous occlusion using linear regression and level set methods. In their method, they first analyze that error image derived from the LRC is a better choice than original image for identifying occluded regions. [24]

CHAPTER 3

METHODOLOGY

In this part of the study, a short explanation on methodology used in the research has been given.

3.1. Problem Definition

Face recognition is an important problem of the computerized systems. Many researches on face recognition have been performed and most of them focus on methodology used for face recognition.

3.2. Program Design and Algorithm

C# algorithm and Microsoft Visual Studio 2010 program have been used to develop design of the program used in the research. Main methodology of the research is PCA based face recognition. In the research, a GUI (Guide User Interface) also have been developed in the program which is shown in the Figure 3.1.

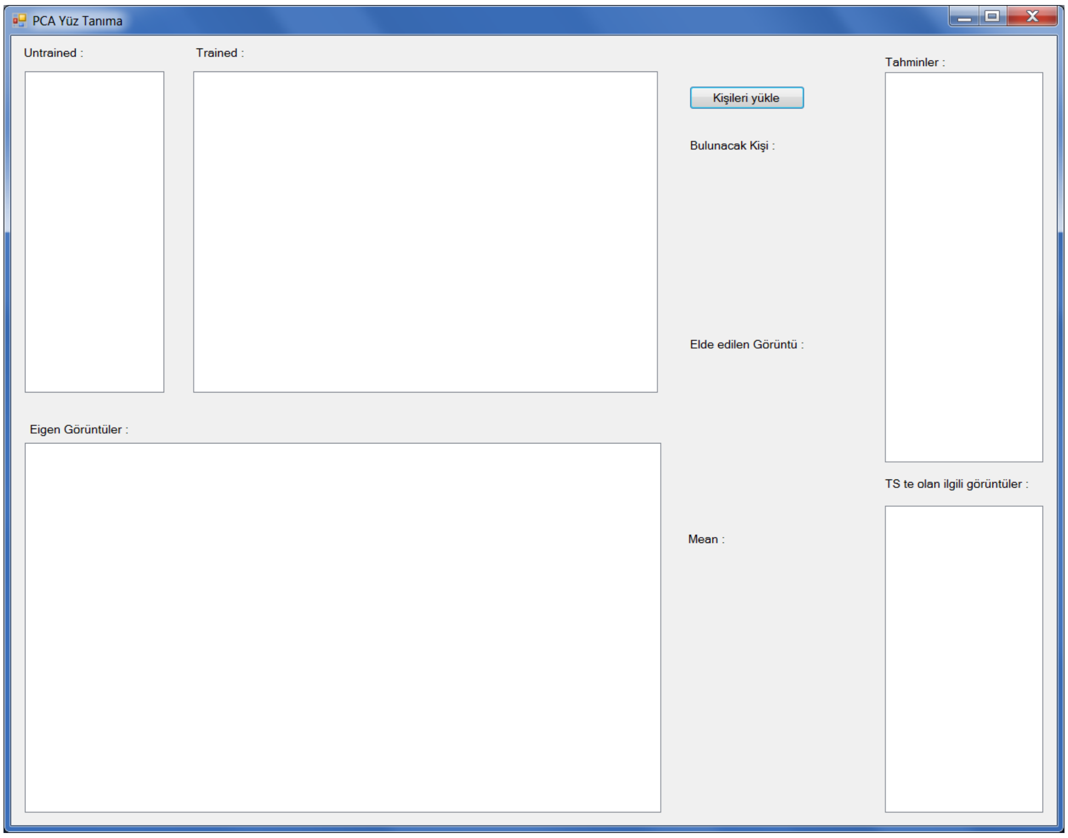

Figure 3. 1: GUI Developed in the Research

In the program GUI, there are five handle areas and three picture areas. The first handle loads images of the training set which is compiled by the user. The next handle below the trained set includes Eigen images of the desired solution. An example of a solution is given in the Figure 3.2.

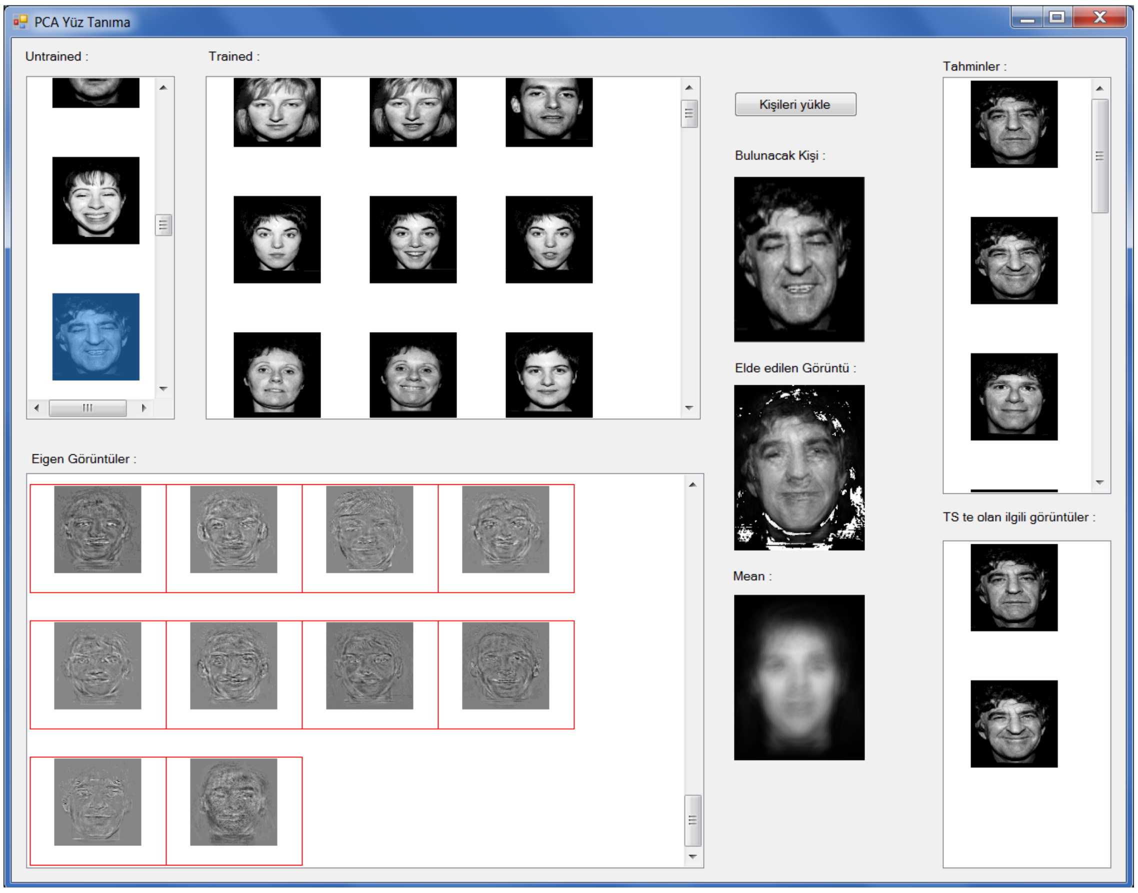

Figure 3. 2: An Example Solution

CHAPTER 4

FINDINGS

4.1. Program Description

In the program, each of the individuals has three images. The first image and other images which are highlighted blue are included in the training set.

Eigen images include images of the solution set. The program GUI highlights in red images which are selected for the solution set. It is possible to increase probability of the solution by adding more images to the solution set.

When clicking an image, image which is desired to recognize is loaded to the required field. Mean part of the GUI shows mean image determined from current solution set.

In the results section of the GUI, images according to the chosen image have been generated. Images added to the training set are shown at this area. The first image is always added to the training set. In the prediction field, ten the closest results to the chosen image are shown.

4.2. Solutions

In the program, a load button is added to add current training set to the face recognition system. A screenshot is given in the Figure 4.1.

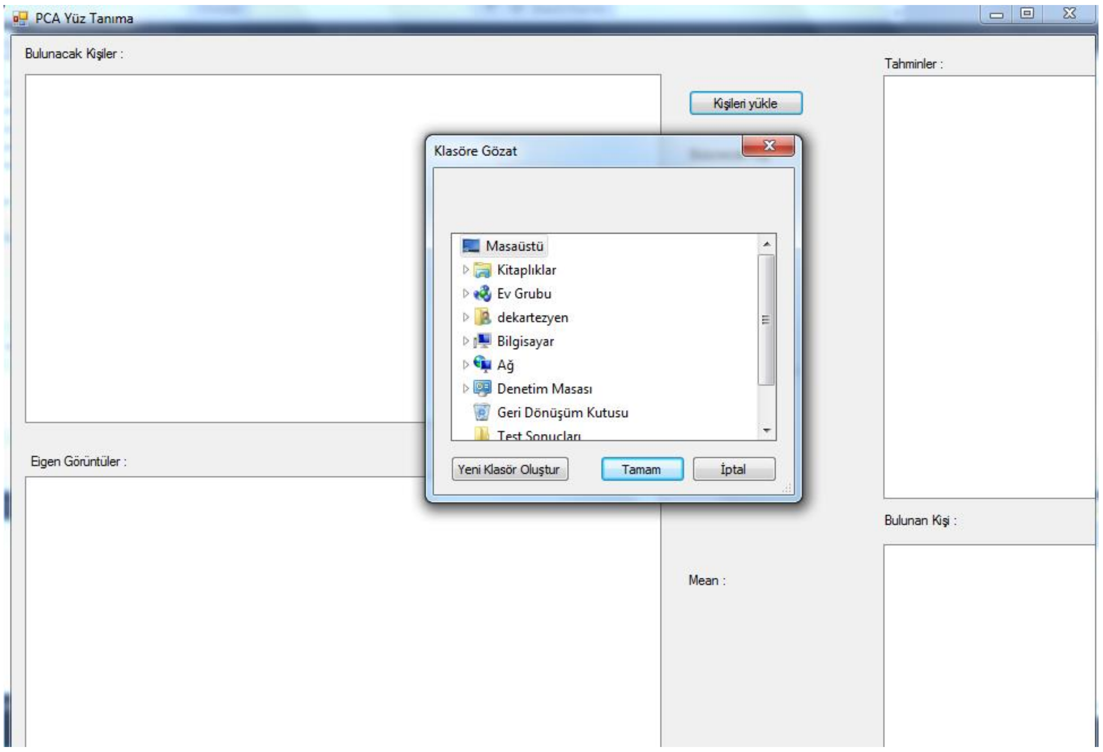

Figure 4. 1: Loading Training Set to the Program

By adding current folder to the program, untrained set appears in the untrained area, and Eigen values of the chosen image is given in the handle below. An example is given in the Figure 4.2.

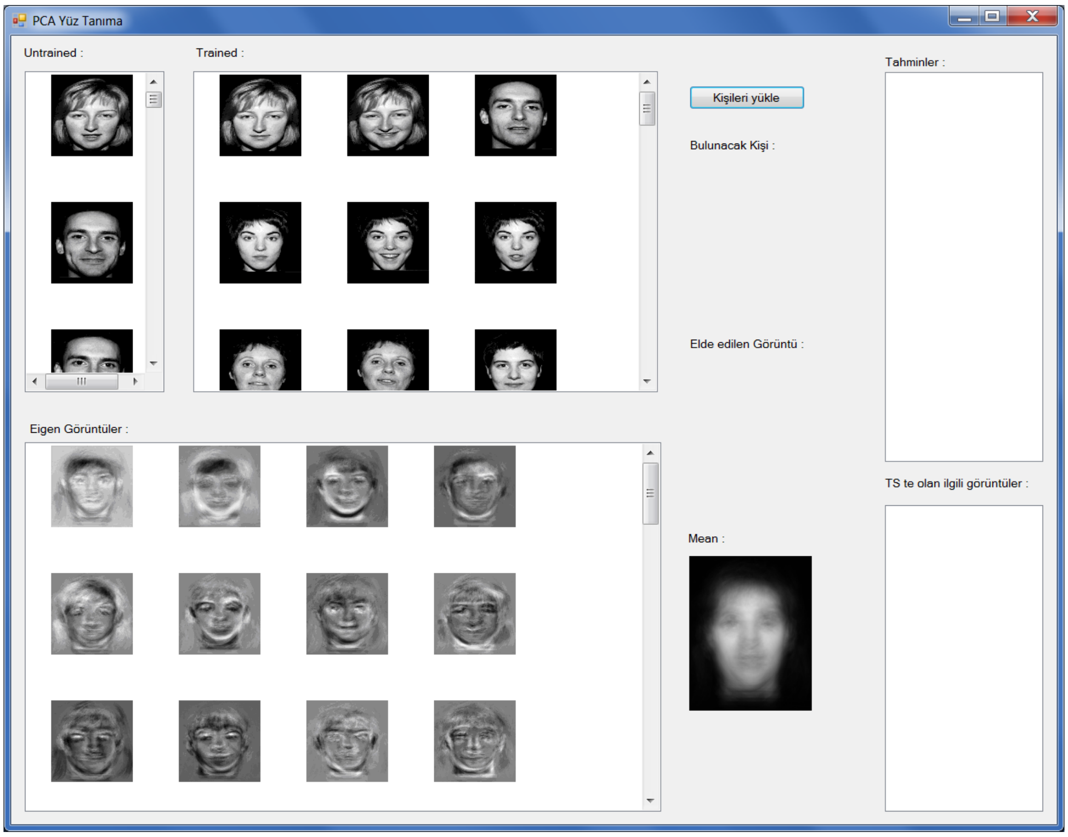

Figure 4. 2: Choosing Image from Untraining Image Set

After choosing of the desired image to be recognized, Eigen set for the solution must be selected. In order to choose Eigen set, the last image including current Eigen set must be chosen. In the program, it is possible to choose more Eigen images. An example is given in the Figure 4.3.

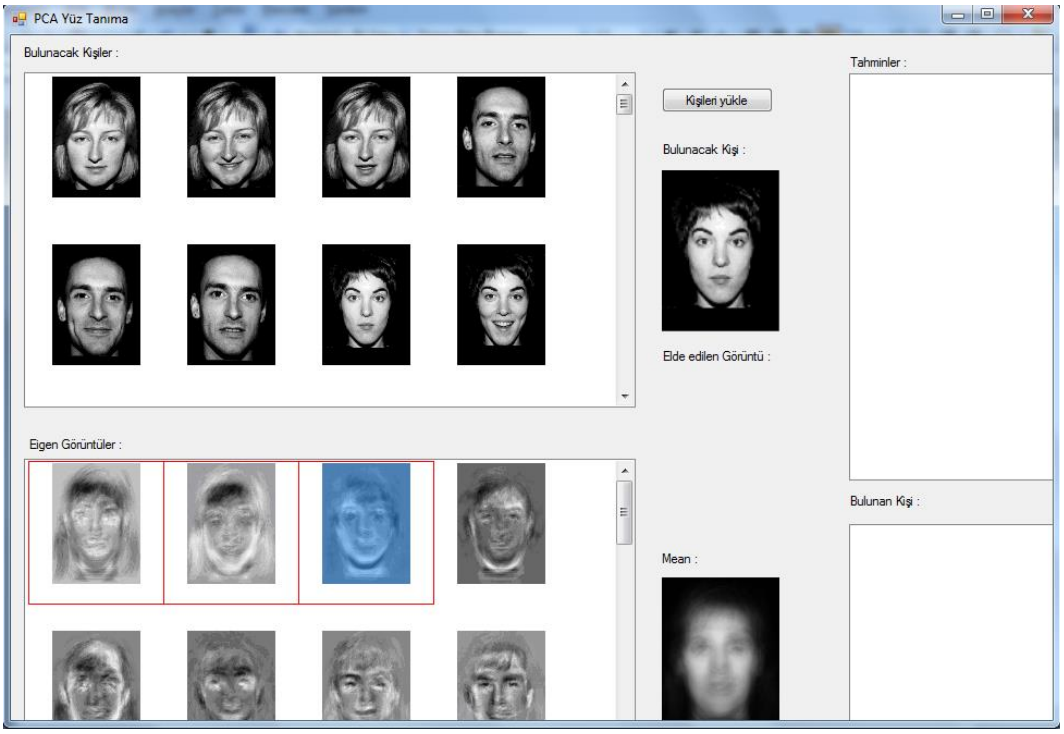

Figure 4. 3: Chosing Eigen Images

After choosing of Eigen image set, alternatives generated from the program are given with determined Eigen value and Mean Value. The result is shown in the result handle of the program as shown in the Figure 4.4.

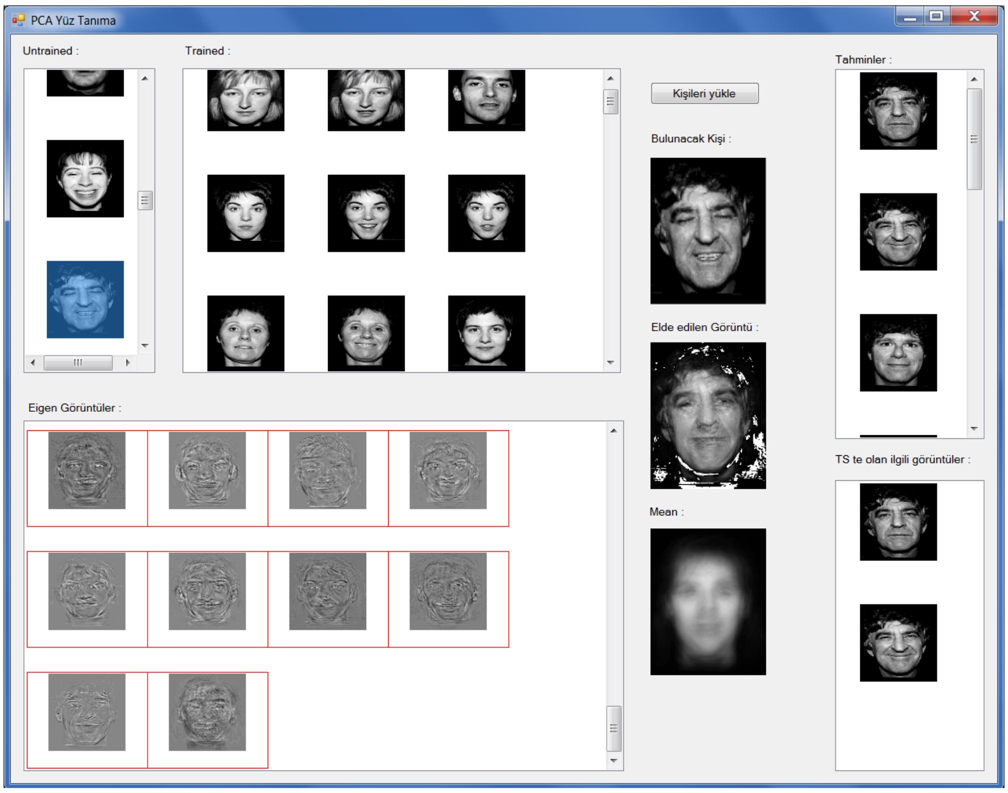

Figure 4. 4: Result Generated by the Program

4.3. Recognition Power of the Program

After developing the program, recognition power was calculated. In order to calculate recognition power of the program, 50 images were chosen randomly, and they were recognized from the program for ten different iteration levels as shown in the Table 4.1. As seen in the Table 4.1, recognition power of the program varies from 0,7 to 0,92. This means that program correctly recognizes an image at 70% to 92% consistent. Change of recognition power based on iteration is given in Scheme

Table 4. 1: Recognition Power of the Program

Number of sample	Number of iterations	Correct recongized number among 50 images	Percentage
1	10	41	0,82%
2	10	44	0,88%
3	10	35	0,70%
4	10	46	0,92%
5	10	39	0,78%
6	10	44	0,88%
7	10	38	0,76%
8	10	39	0,78%
9	10	42	0,84%
10	10	43	0,86%

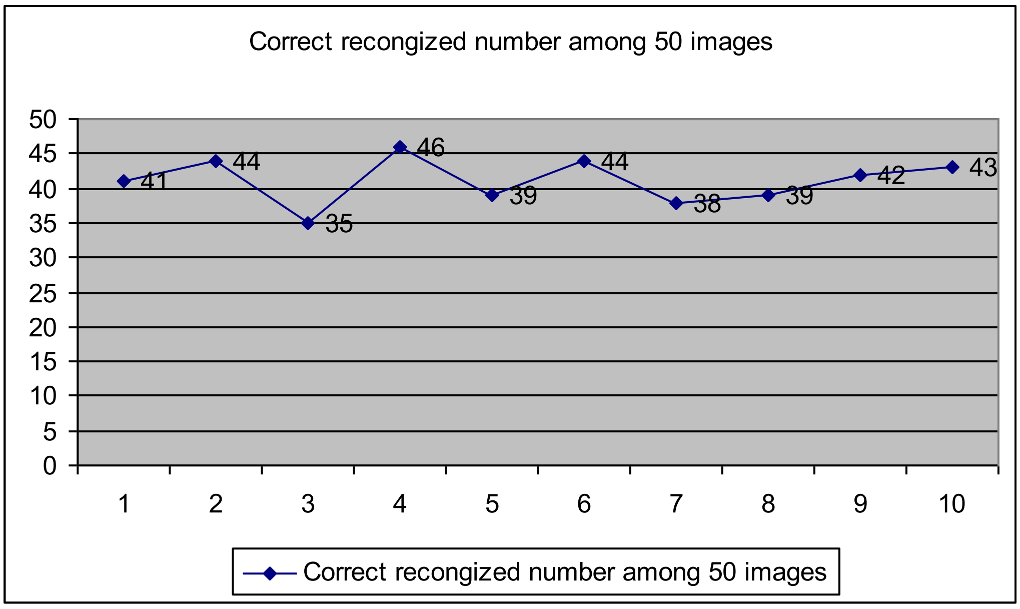

Figure 4. 5: Recognition Power of the Program

Scheme shows that 4th sample gives the best solution. This is not mean that each 4th sample reaches the maximum value. However, it is seen that there is a maximum point which reachs to the 92%. At avarege, it may be argued that proposed program gives recognition solutions at a rate over %80.

CONCLUSIONS

In the program, PCA based face recognition have been examined, and an algorithm included a GUI has been developed. In the literature part of the study, it was shown that face recognition is an important issue of the computerized systems since it has many application areas such as security systems, identification and detection systems.

In the developed program, it is possible to recognize an image by using a training image set. However, there some parts of the program must be developed. Firstly, program uses a training set created by the user. In case there is not any relevant image in the training set, result and recognition process may fail. Another thing must be developed is Eigen values chosen by the user. In the program, it was thought that choosing limited Eigen value decreases recognition duration, so it is possible to minimize recognition process. Another point is that the program training image set may be extended based on user demands.

REFERENCES

[1] **FIN DE CARRERA, P.** (2010), Face Recognition Algorithms, Universidad del Paris Vasco.

[2] **DIBEKOĞLU, H.** (2006), Part-Based 3d Face Recognition Under Pose And Expression Variations, B.S, in Computer Engineering, Yeditepe University.

[3] **ARAR, NM. (2010),** Fusing Local Appearance Models For Face Recognition, B.S. Computer Engineering, Bilkent University.

[4] **KIM, K.** (2009), Face Recognition using Principle Component Analysis, Department of Computer Science University of Maryland, College Park MD 20742, USA.

[5] Principal Component Analysis, http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/IDAPILecture15.pdf. [17.12.2013]

[6] http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/IDAPILecture15.pdf. [22.12.2013]

[7] Eigenvalues and Eigenvectors, http://math.mit.edu/linearalgebra/ila0601.pdf. [28.12.2013]

[8] Eigenvalues and Eigenvectors, http://www.math.harvard.edu/archive/20_spring_05/handouts/ch05_notes.pdf. [19.12.2013]

[9] **TURK, M. AND PENTLAND, A.** (1991), "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991.

[10] **TAN, K. AND CHEN, S.** (2005), Adaptively weighted sub-pattern PCA for face recognition, Neurocomputing 64 (2005) 505–511.

[11] **ZHOU, C, WANG, L. ZHANG, Q. AND WEI, X.** (2013), Face recognition based on PCA image reconstruction and LDA, Optik 124 (2013) 5599– 5603.

[12] **WANG, H, LENG, Y, WANG, Z AND WU, X.** (2007), Application of image correction and bit-plane fusion in generalized PCA based face recognition, Pattern Recognition Letters 28 (2007) 2352–2358.

[13] **HSIEH, PC AND TUNG, PC,** (2009), A novel hybrid approach based on sub-pattern technique and whitened PCA for face recognition, Pattern Recognition 42 (2009)978—984.

[14] **LUH, GC AND LIN, CY.** (2011), PCA based immune networks for human face recognition, Applied Soft Computing 11 (2011) 1743–1752.

[15] **KUSUMA, G AND CHUA, P CS.** (2011), PCA-based image recombination for multimodal 2D+3D face recognition, Image and Vision Computing 29 (2011) 306–316

[16] **KUSUMA, G AND CHUA, P CS.** (2011), PCA-based image recombination for multimodal 2D+3D face recognition, Image and Vision Computing 29 (2011) 306–316

[17] **OH, SK AND YOO, SH.** (2013), Witold Pedrycz, Design of face recognition algorithm using PCA -LDA combined for hybrid data pre-processing and polynomial-based RBF neural networks : Design and its application, Expert Systems with Applications 40 (2013) 1451–1466.

[18] **ZHANG, D AND ZHOU, ZH. (2005),** (2D) 2 PCA: Two-directional two-dimensional PCA for efficient face representation and recognition, Neurocomputing 69 (2005) 224–231

[19] **TAN, K AND CHEN, S.** (2005), Adaptively weighted sub-pattern PCA for face recognition, Neurocomputing 64 (2005) 505–511.

[20] **ABEER, MAL, SHIHA, WL, WOO AND DLAY, SS.** (2014), Multi-linear neighborhood preserving projection for face recognition, Pattern Recognition 47 (2014) 544–555.

[21] **PRADIPTA, K, BANERJEE, A AND DATTA, K.** (2014), Class specific subspace dependent nonlinear correlation filtering for illumination tolerant face recognition, Pattern Recognition Letters 36 (2014) 177–185.

[22] **GIVENS, GH, BEVERIDGE, JR, PHILLIPS, PJ, DRAPER, B, LUI, YM AND BOLME, D.** (2013), Introduction to face recognition and evaluation of

algorithm performance, Computational Statistics and Data Analysis 67 (2013) 236–247.

[23] **LI, S, GONG, D AND YUAN, Y.** (2013), Face recognition using Weber local descriptors, Neurocomputing 122 (2013) 272–283.

[24] **LUAN, X, FANG, B, LIU, L AND ZHOU, L.** (2013), Face recognition with contiguous occlusion using linear Regression and level set method, Neurocomputing122(2013)386–397.

APPENDICES

APPENDIX A. Solution Algorithm

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Drawing;

namespace PCA
{
    public class MyPGMs
    {
        private ObservableCollection<MyPGM> _images = new
ObservableCollection<MyPGM>();
        public ObservableCollection<MyPGM> Images { get { return _images; } }
        public int Count { get { return _images.Count; } }
        public MyPGMs() { }
    }

    public class MyPGM : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(String info)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(info));
            }
        }

        public static byte[] S_header = null;

        public static System.Drawing.Bitmap
MakeGrayscale(System.Drawing.Bitmap original)
        {
            // check if it is already gray scaled by sampling
            Random rand = new Random();
            int grayCount = 0;
            for (int i = 0; i < 10; i++)
            {
                System.Drawing.Color sampleColor =
original.GetPixel(rand.Next(original.Width), rand.Next(original.Height));
                if (sampleColor.R == sampleColor.G && sampleColor.R ==
sampleColor.B)
                    grayCount++;
            }
            if (grayCount >= 10)
                return original;

            //make an empty bitmap the same size as original
            System.Drawing.Bitmap newBitmap = new
System.Drawing.Bitmap(original.Width, original.Height);

            for (int i = 0; i < original.Width; i++)
            {
                for (int j = 0; j < original.Height; j++)
                {
                    //get the pixel from the original image
                    System.Drawing.Color originalColor = original.GetPixel(i,
j);

                    //create the grayscale version of the pixel
                    int grayScale = (int)((originalColor.R * .3) +
(originalColor.G * .59)
                        + (originalColor.B * .11));

                    //create the color object
                    System.Drawing.Color newColor =
System.Drawing.Color.FromArgb(grayScale, grayScale, grayScale);

                    //set the new image's pixel to the grayscale version
                    newBitmap.SetPixel(i, j, newColor);
                }
            }

            return newBitmap;
        }

        public static System.Drawing.Bitmap GetARGB_Bitmap(string filePath)
        {
            System.Drawing.Image imgPhoto = new
System.Drawing.Bitmap(filePath);
            int width = imgPhoto.Width;
            int height = imgPhoto.Height;
            System.Drawing.Rectangle drawingArea = new
System.Drawing.Rectangle(0, 0, width, height);

            System.Drawing.Bitmap canvasARGB = new
System.Drawing.Bitmap(width, height,
System.Drawing.Imaging.PixelFormat.Format32bppArgb);
            canvasARGB.SetResolution(96, 96);
            System.Drawing.Graphics penARGB =
System.Drawing.Graphics.FromImage(canvasARGB);
            penARGB.Clear(System.Drawing.Color.Black);
            penARGB.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;
            penARGB.DrawImage(imgPhoto, drawingArea, drawingArea,
System.Drawing.GraphicsUnit.Pixel);
            penARGB.Dispose();

            return canvasARGB;
        }


        private int _width, _height;
        private byte[] _header;
        private byte[] _pixels;
        private string _filePath, _fileName;
        public int Width { get { return _width; } }
        public int Height { get { return _height; } }
        public string FilePath { get { return _filePath; } }
        public string FileName { get { return _fileName; } }
        public byte[] Pixels { get { return _pixels; } }

        private bool _isSelected = false;
        public bool IsSelected { get { return _isSelected; } set { _isSelected
= value; this.NotifyPropertyChanged("IsSelected");
this.NotifyPropertyChanged("IsSelectedVisible"); } }

        public bool _isTrained = false;
        public bool IsTrained { get { return _isTrained; } set { _isTrained =
value; this.NotifyPropertyChanged("IsTrained");
this.NotifyPropertyChanged("IsTrainedVisible"); } }

        public bool _isTrained2 = false;
        public bool IsTrained2 { get { return _isTrained2; } set { _isTrained2
= value; this.NotifyPropertyChanged("IsTrained2");
this.NotifyPropertyChanged("IsTrainedVisible2"); } }

        public MyPGM(string filePath)
        {
            // Paths
            System.IO.FileInfo finfo = new System.IO.FileInfo(filePath);
            _filePath = finfo.FullName;
            _fileName = finfo.Name;

            // Image
            System.Drawing.Image im = null;
            try
            {
                im = ShaniSoft.Drawing.PNM.ReadPNM(filePath);
            }
            catch (Exception ex)
            {
                System.Drawing.Bitmap ARGB_Bitmap = GetARGB_Bitmap(filePath);
                im = MakeGrayscale(ARGB_Bitmap);
            }

            byte[] tempBytes = imageToByteArray(im);
            _width = im.Width;
            _height = im.Height;
            _pixels = new byte[_width * _height];
            _header = new byte[tempBytes.Length - (_pixels.Length * 4)];
            for (int i = 0; i < _header.Length; i++)
                _header[i] = tempBytes[i];
            for (int i = _header.Length, j = 0; i < tempBytes.Length; i = i +
4, j++)
```

```
                    _pixels[j] = tempBytes[i];

                // S_header
                if (S_header == null)
                {
                    S_header = new byte[_header.Length];
                    for (int i = 0; i < _header.Length; i++)
                        S_header[i] = _header[i];
                }
            }

            public MyPGM(byte[] pixels)
            {
                _filePath = "none";
                _fileName = "none";
                _width = -1;
                _height = -1;

                _header = new byte[S_header.Length];
                _pixels = new byte[pixels.Length];
                for (int i = 0; i < S_header.Length; i++)
                    _header[i] = S_header[i];
                for (int i = 0; i < pixels.Length; i++)
                    _pixels[i] = pixels[i];
            }

            public byte GetPixel(int index)
            {
                return _pixels[index];
            }

            public byte GetPixel(int x, int y)
            {
                return _pixels[y * _width + x];
            }

            public void SetPixel(int index, byte value)
            {
                _pixels[index] = value;
            }

            public void SetPixel(int x, int y, byte value)
            {
                _pixels[y * _width + x] = value;
            }

            public System.Drawing.Image DrawingImage
            {
                get
                {
                    byte[] imageByte = new byte[_header.Length + (_pixels.Length *
4)];
                    _header.CopyTo(imageByte, 0);
                    for (int i = 0, j = _header.Length; i < _pixels.Length; i++, j
= j + 4)
                    {
```

```
                    imageByte[j + 0] = _pixels[i];
                    imageByte[j + 1] = _pixels[i];
                    imageByte[j + 2] = _pixels[i];
                    imageByte[j + 3] = 255;
                }
                return byteArrayToImage(imageByte);
            }


        }


        public static byte[] imageToByteArray(System.Drawing.Image imageIn)
        {
            System.IO.MemoryStream ms = new System.IO.MemoryStream();
            imageIn.Save(ms, System.Drawing.Imaging.ImageFormat.Bmp);
            return ms.ToArray();
        }

        public static System.Drawing.Image byteArrayToImage(byte[]
byteArrayIn)
        {
            System.IO.MemoryStream ms = new
System.IO.MemoryStream(byteArrayIn);
            System.Drawing.Image returnImage =
System.Drawing.Image.FromStream(ms);
            return returnImage;
        }
    }
}
```

APPENDIX B. Form 1.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using ILNumerics;

namespace PCA
{
    public partial class Form1 : Form
    {
        Collection<Collection<string>> filePaths = new 
Collection<Collection<string>>();
        Collection<string> personPaths = new Collection<string>();
        int _pictureSize = 0;
        MyPGMs _myPGMs = new MyPGMs();
        MyPGMs _myPGMsTrained = new MyPGMs();
        MyPGM[] _myPGMsTrainedArray = null;

        // Display
        int _currentFirstDisaply = 0;
        MyPGMs _myPGMsDisplay;
        MyPGMs _myPGMsEigen;
        MyPGMs _myPGMsWant;
        MyPGMs _myPGMsFind;

        // SVD
        float[] _mean = null;
        float[] _eigenValues = null;
        Collection<float[]> _eigenVectors = new Collection<float[]>();
        Collection<float[]> _eigenWeights = new Collection<float[]>(); // 
cached eigen Weights per trained face
        int _selectedVectors = 0;
        int _selectedIndeks = 0;

        public Form1()
        {
            InitializeComponent();
        }
        private void ShowImages(int from, int toExclusive)
        {
            trainingList.Clear();
            ImageList im = new ImageList();
            im.ImageSize = new Size(81, 111);
            trainingList.LargeImageList = im;
            for (int i = from; i < toExclusive; i++)
            //_myPGMsDisplay.Images.Add(_myPGMs.Images[i]);
            {
                im.Images.Add(_myPGMs.Images[i].DrawingImage);
                ListViewItem li = new ListViewItem();
                li.Tag = _myPGMs.Images[i]._isTrained2;
                li.ImageIndex = i;
                trainingList.Items.Add(li);
            }



            //for (int t = 0; t < filePaths.Count; t++)
            //{

            //}

        }

        private void LoadFiles()
        {

            if (fbDialog.ShowDialog() == DialogResult.OK)
            {
                //string paath = "E:\\FaceDetect\\Faces Gray1";
                System.IO.FileInfo[] files = new
System.IO.DirectoryInfo(fbDialog.SelectedPath).GetFiles();
                Collection<Collection<string>> filePaths = new
Collection<Collection<string>>();
                Collection<string> personPaths = new Collection<string>();
                personPaths.Add(files[0].FullName);
                filePaths.Add(personPaths);

                for (int i = 1; i < Math.Min(files.Length, 250); i++)
                {
                    string personName = personPaths[0];
                    string nextName = files[i].FullName;
                    int numberOff = 0;
                    for (int i2 = 0; i2 < Math.Min(personName.Length,
nextName.Length); i2++)
                    {
                        if (personName[i2] != nextName[i2])
                            numberOff++;
                        if (numberOff > 2)
                            break;
                    }
                    if (numberOff > 2)
                    {
                        personPaths = new Collection<string>();
                        personPaths.Add(nextName);
                        filePaths.Add(personPaths);
                    }
                    else
                        personPaths.Add(nextName);
                }


                Random rand = new Random();
                for (int i = 0; i < filePaths.Count; i++)
                {
                    for (int i2 = 0; i2 < filePaths[i].Count; i2++)
                    {
                        MyPGM tempPGM = new MyPGM((filePaths[i])[i2]);
                        if (_myPGMs.Count > 0 && (tempPGM.Width !=
_myPGMs.Images[0].Width || tempPGM.Height != _myPGMs.Images[0].Height))
                            throw new Exception(string.Format("Wrong Image
Dimensions!! Width {0} vs. {1} Height {2} vs. {3}", tempPGM.Width,
_myPGMs.Images[0].Width, tempPGM.Height, _myPGMs.Images[0].Height));

                        _myPGMs.Images.Add(tempPGM);

                        if (i2 == 0)
                        {
                            tempPGM.IsTrained = true;
                            _myPGMsTrained.Images.Add(tempPGM);
                        }
                        else
                        {
                            if (rand.Next(100) < 20)
                            {
                                tempPGM.IsTrained2 = true;
                                _myPGMsTrained.Images.Add(tempPGM);
                            }
                        }
                        if ((i + 1) % 100 == 0)
                            Console.WriteLine("LoadFiles " + (i + 1));
                    }
                }
                _myPGMsTrainedArray = new MyPGM[_myPGMsTrained.Count];
                _myPGMsTrained.Images.CopyTo(_myPGMsTrainedArray, 0);
                _pictureSize = _myPGMs.Images[0].Pixels.Length;
                ShowImages(0,  _myPGMs.Count);
                ProcessFaces();
                AddEigenFaces();
            }
        }
        private float[] GetMean()
        {
            float[] mean = new float[_pictureSize];
            for (int i = 0; i < mean.Length; i++)
                mean[i] = 0;

            for (int i = 0; i < _myPGMsTrainedArray.Length; i++)
            {
                MyPGM tempImage = _myPGMsTrainedArray[i];
                for (int j = 0; j < _pictureSize; j++)
                    mean[j] += tempImage.GetPixel(j);
            }

            for (int i = 0; i < mean.Length; i++)
                mean[i] = mean[i] / _myPGMsTrainedArray.Length;

            return mean;
        }
        private void SetMeanFace(float[] mean)
        {
            byte[] meanPixels = new byte[mean.Length];
            for (int i = 0; i < meanPixels.Length; i++)
                meanPixels[i] = (byte)Math.Round(mean[i]);

            MyPGM meanPGM = new MyPGM(meanPixels);
            untrainedImage.Image = meanPGM.DrawingImage;
        }
        private float[,] GetDiff(float[] subtractedByMeanVector)
        {
            float[,] diff = new float[_myPGMsTrainedArray.Length,
_pictureSize];
            for (int i = 0; i < _myPGMsTrainedArray.Length; i++)
                for (int j = 0; j < _pictureSize; j++)
                    diff[i, j] = _myPGMsTrainedArray[i].GetPixel(j) -
subtractedByMeanVector[j];
            return diff;
        }
        private void ProcessFaces()
        {
            _mean = GetMean();
            SetMeanFace(_mean);

            // Get Diff
            float[,] diff = GetDiff(_mean);

            // Move To ILArray
            ILArray<float> A = diff;
            diff = null;

            // SVD
            //ILArray<float> cov =
ILNumerics.BuiltInFunctions.ILMath.multiply(A, A.T);
            ILArray<float> covSmall =
ILNumerics.BuiltInFunctions.ILMath.multiply(A.T, A);
            ILArray<float> U = new ILArray<float>();
            ILArray<float> S = new ILArray<float>();
            ILArray<float> V = new ILArray<float>();
            S = ILNumerics.Algorithms.ILAlgorithm.svd(covSmall, ref U, ref V);

            // eigenValues
            _eigenValues = null;
            S.Diagonal.ExportValues(ref _eigenValues);

            float[] eigenValuesPow = new float[_eigenValues.Length];
            for (int i = 0; i < eigenValuesPow.Length; i++)
                eigenValuesPow[i] = (float)Math.Pow(_eigenValues[i], -0.5);

            // Add EigenVectors
            _eigenVectors.Clear();
            for (int i = 0; i < eigenValuesPow.Length; i++)
            {
                ILArray<float> oneEigenVector =
(ILArray<float>)V.Subarray(":", i.ToString());
```

```
                ILArray<float> oneFace =
ILNumerics.BuiltInFunctions.ILMath.multiply(A, oneEigenVector);
                ILArray<float> oneFace2 =
ILNumerics.BuiltInFunctions.ILMath.multiplyElem(oneFace, eigenValuesPow[0]);
                float[] eigenVectorArray = null;
                oneFace2.ExportValues(ref eigenVectorArray);

                // need to normalize by dividing using "distance".
                double distance = 0;
                for (int j = 0; j < eigenVectorArray.Length; j++)
                    distance += Math.Pow(eigenVectorArray[j], 2);
                distance = Math.Sqrt(distance);

                for (int j = 0; j < eigenVectorArray.Length; j++)
                    eigenVectorArray[j] /= (float)distance;

                _eigenVectors.Add(eigenVectorArray);
            }

            // Add EigenWeights
            _eigenWeights.Clear();
            for (int i = 0; i < _myPGMsTrainedArray.Length; i++)
            {
                float[] existingWeight =
GetEigenWeight(_myPGMsTrainedArray[i].Pixels, _eigenVectors.Count);
                _eigenWeights.Add(existingWeight);
            }
        }
        private float[] GetEigenWeight(byte[] pixels, int numOfVectors)
        {
            float[] result = new float[numOfVectors];

            float[] diff = new float[pixels.Length];
            for (int i = 0; i < diff.Length; i++)
                diff[i] = (float)pixels[i] - _mean[i];

            for (int j = 0; j < numOfVectors; j++)
            {
                float W = 0;
                float[] vectorI = _eigenVectors[j];
                for (int i = 0; i < diff.Length; i++)
                    W += diff[i] * vectorI[i];
                result[j] = W;
            }
            return result;
        }
        private void btnLoad_Click(object sender, EventArgs e)
        {
            LoadFiles();
        }
        private void AddEigenFaces()
        {

            int maxDisplayEigens = 0;
            int total = _eigenVectors.Count;
            if (maxDisplayEigens > 0)
```

```
                total = Math.Min(maxDisplayEigens, total);
            eigenList.Clear();
            ImageList im = new ImageList();
            im.ImageSize = new Size(81, 111);
            eigenList.LargeImageList = im;

            for (int i = 0; i < total; i++)
            {
                MyPGM tmpPGM = AddEigenFace(_eigenVectors[i]);
                im.Images.Add(tmpPGM.DrawingImage);
                ListViewItem li = new ListViewItem();
                li.Tag = false;
                li.ImageIndex = i;
                eigenList.Items.Add(li);

            }
        }
        private MyPGM AddEigenFace(float[] eigenVectorArray)
        {
            float min = 0, max = 0;
            for (int i = 0; i < eigenVectorArray.Length; i++)
            {
                if (max < eigenVectorArray[i])
                    max = eigenVectorArray[i];
                if (min > eigenVectorArray[i])
                    min = eigenVectorArray[i];
            }

            byte[] eigenPixels = new byte[eigenVectorArray.Length];
            for (int i = 0; i < eigenPixels.Length; i++)
                eigenPixels[i] = (byte)(255 * ((eigenVectorArray[i] - min) / 
(max - min)));

            MyPGM eigenFace = new MyPGM(eigenPixels);
            return eigenFace;


        }
        private void SetReconstructFace(MyPGM pgm)
        {
            if (_selectedVectors > 0)
            {
                float[] newWeight = GetEigenWeight(pgm.Pixels, 
_selectedVectors);
                byte[] newFace = GenerateNewFace(newWeight);
                MyPGM newPGM = new MyPGM(newFace);
                trainedImage.Image = newPGM.DrawingImage;
            }
        }
        private byte[] GenerateNewFace(float[] newWeight)
        {
            float[] result = new float[_mean.Length];
            for (int i = 0; i < result.Length; i++)
                result[i] = _mean[i];

            for (int j = 0; j < newWeight.Length; j++)
```

```
                {
                    float W = newWeight[j];
                    float[] vectorI = _eigenVectors[j];
                    byte[] eigenFace = (new
MyPGM(MyPGM.imageToByteArray(eigenList.LargeImageList.Images[j]))).Pixels;
                    for (int i = 0; i < result.Length; i++)
                        result[i] += W * (float)vectorI[i];
                }

                byte[] resultB = new byte[_mean.Length];
                for (int i = 0; i < resultB.Length; i++)
                    resultB[i] = (byte)result[i];
                return resultB;
            }
            private double GetDistance(float[] newWeight, float[] existingWeight)
            {
                double result = 0;
                for (int j = 0; j < newWeight.Length; j++)
                    result += Math.Pow(((double)newWeight[j] -
(double)existingWeight[j]) / (double)_pictureSize, 2);
                result = Math.Sqrt(result);
                result /= Math.Sqrt(newWeight.Length);
                return result;
            }
            private void SetWantFindFace(MyPGM pgm)
            {
                if (_selectedVectors > 0)
                {
                    estimatedList.Clear();
                    ImageList ig = new ImageList();
                    ig.ImageSize = new Size(81, 111);
                    estimatedList.LargeImageList = ig;
                    int selectedIndex = _selectedIndeks;
                    for (int i = selectedIndex; i >= 0; i--)
                    {


                        if (_myPGMs.Images[i]._isTrained)
                        {
                            ig.Images.Add(_myPGMs.Images[i].DrawingImage);
                            ListViewItem li = new ListViewItem();
                            li.ImageIndex = ig.Images.Count-1;
                            estimatedList.Items.Add(li);
                            for (int i2 = i + 1; i2 < _myPGMs.Count; i2++)
                            {
                                if (_myPGMs.Images[i2]._isTrained == true)
                                    break;
                                if (_myPGMs.Images[i2]._isTrained2 == true)
                                {

ig.Images.Add(_myPGMs.Images[i2].DrawingImage);
                                    ListViewItem lt = new ListViewItem();
                                    lt.ImageIndex = ig.Images.Count-1;
                                    estimatedList.Items.Add(lt);
                                }
                            }
                            break;
                        }
                    }

                    double matchDistance = 0.05;

                    float[] newWeight = GetEigenWeight(pgm.Pixels,
    _selectedVectors);
                    Collection<double> sortedWeights = new Collection<double>();
                    Collection<MyPGM> sortedPGMs = new Collection<MyPGM>();
                    for (int i = 0; i < _myPGMsTrainedArray.Length; i++)
                    {
                        double distance = GetDistance(newWeight,
    _eigenWeights[i]);
                        if (distance <= matchDistance)
                        {
                            if (sortedWeights.Count == 0)
                            {
                                sortedWeights.Add(distance);
                                sortedPGMs.Add(_myPGMsTrainedArray[i]);
                            }
                            else
                            {
                                for (int j = 0; j < sortedWeights.Count; j++)
                                {
                                    if (distance < sortedWeights[j])
                                    {
                                        sortedWeights.Insert(j, distance);
                                        sortedPGMs.Insert(j,
    _myPGMsTrainedArray[i]);
                                        break;
                                    }
                                }
                            }
                        }
                    }
                    solutionList.Clear();
                    ImageList im = new ImageList();
                    im.ImageSize = new Size(81, 111);
                    solutionList.LargeImageList = im;
                    for (int i = 0; i < Math.Min(10, sortedWeights.Count); i++)
                    {
                        im.Images.Add(sortedPGMs[i].DrawingImage);
                            ListViewItem li = new ListViewItem();
                            li.ImageIndex = i;
                            solutionList.Items.Add(li);


                    }
                }
            }
            private void trainingList_ItemSelectionChanged(object sender,
    ListViewItemSelectionChangedEventArgs e)
            {
                if (trainingList.SelectedIndices.Count > 0)
                {
                    _selectedIndeks = trainingList.SelectedIndices[0];
                    selectedImage.Image = 
    e.Item.ImageList.Images[_selectedIndeks];

                    SetReconstructFace(_myPGMs.Images[_selectedIndeks]);
                    SetWantFindFace(_myPGMs.Images[_selectedIndeks]);
                }
            }

            private void eigenList_ItemSelectionChanged(object sender, 
    ListViewItemSelectionChangedEventArgs e)
            {
                if (eigenList.SelectedIndices.Count > 0)
                {
                    _selectedVectors = eigenList.SelectedIndices[0] + 1;
                    for (int i = 0; i < eigenList.Items.Count; i++)
                    {
                        if (i < _selectedVectors)
                            eigenList.Items[i].Tag = true;
                        else
                            eigenList.Items[i].Tag = false;
                    }
                }
                eigenList.Invalidate();
            }

            private void trainingList_DrawItem(object sender, 
    DrawListViewItemEventArgs e)
            {
                e.DrawDefault = true;
                if((bool)e.Item.Tag == true)
                e.Graphics.DrawRectangle(new Pen(new SolidBrush(Color.Blue)), 
    e.Item.Bounds);
            }

            private void eigenList_DrawItem(object sender, 
    DrawListViewItemEventArgs e)
            {
                e.DrawDefault = true;
                if ((bool)e.Item.Tag == true)
                    e.Graphics.DrawRectangle(new Pen(new SolidBrush(Color.Red)), 
    e.Item.Bounds);
            }


        }
    }
```

APPENDIX C. PCA Code

```xml
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Import
Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.prop
s"
Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.C
ommon.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == ''
">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{0141CA43-B44E-4418-B19A-AB60A7EB8C06}</ProjectGuid>
    <OutputType>WinExe</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>PCA</RootNamespace>
    <AssemblyName>PCA</AssemblyName>
    <TargetFrameworkVersion>v4.0</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <TargetFrameworkProfile />
    <PublishUrl>publish\</PublishUrl>
    <Install>true</Install>
    <InstallFrom>Disk</InstallFrom>
    <UpdateEnabled>false</UpdateEnabled>
    <UpdateMode>Foreground</UpdateMode>
    <UpdateInterval>7</UpdateInterval>
    <UpdateIntervalUnits>Days</UpdateIntervalUnits>
    <UpdatePeriodically>false</UpdatePeriodically>
    <UpdateRequired>false</UpdateRequired>
    <MapFileExtensions>true</MapFileExtensions>
    <ApplicationRevision>0</ApplicationRevision>
    <ApplicationVersion>1.0.0.%2a</ApplicationVersion>
    <IsWebBootstrapper>false</IsWebBootstrapper>
    <UseApplicationTrust>false</UseApplicationTrust>
    <BootstrapperEnabled>true</BootstrapperEnabled>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU'
">
    <PlatformTarget>x86</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
    <CodeAnalysisIgnoreGeneratedCode>false</CodeAnalysisIgnoreGeneratedCode>
    <CodeAnalysisRuleSet>MinimumRecommendedRules.ruleset</CodeAnalysisRuleSet>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' ==
'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="ILNumerics.Net">
      <HintPath>ILNumerics.Core_runtime\ILNumerics.Net.dll</HintPath>
    </Reference>
    <Reference Include="ShaniSoft.Drawing.PNM">
      <HintPath>.\ShaniSoft.Drawing.PNM.dll</HintPath>
    </Reference>
    <Reference Include="System">
      <Private>True</Private>
    </Reference>
    <Reference Include="System.Core">
      <Private>True</Private>
    </Reference>
    <Reference Include="System.Xml.Linq">
      <Private>True</Private>
    </Reference>
    <Reference Include="System.Data.DataSetExtensions">
      <Private>True</Private>
    </Reference>
    <Reference Include="Microsoft.CSharp">
      <Private>True</Private>
    </Reference>
    <Reference Include="System.Data">
      <Private>True</Private>
    </Reference>
    <Reference Include="System.Deployment">
      <Private>True</Private>
    </Reference>
    <Reference Include="System.Drawing">
      <Private>True</Private>
    </Reference>
    <Reference Include="System.Windows.Forms">
      <Private>True</Private>
    </Reference>
    <Reference Include="System.Xml">
      <Private>True</Private>
    </Reference>
  </ItemGroup>
  <ItemGroup>
    <Compile Include="Form1.cs">
      <SubType>Form</SubType>
    </Compile>
    <Compile Include="Form1.Designer.cs">
      <DependentUpon>Form1.cs</DependentUpon>
    </Compile>
    <Compile Include="MyPGM.cs" />
    <Compile Include="Program.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
    <EmbeddedResource Include="Form1.resx">
      <DependentUpon>Form1.cs</DependentUpon>
    </EmbeddedResource>
    <EmbeddedResource Include="Properties\Resources.resx">
      <Generator>ResXFileCodeGenerator</Generator>
      <LastGenOutput>Resources.Designer.cs</LastGenOutput>
      <SubType>Designer</SubType>
    </EmbeddedResource>
    <Compile Include="Properties\Resources.Designer.cs">
      <AutoGen>True</AutoGen>
      <DependentUpon>Resources.resx</DependentUpon>
      <DesignTime>True</DesignTime>
    </Compile>
    <None Include="Properties\Settings.settings">
      <Generator>SettingsSingleFileGenerator</Generator>
      <LastGenOutput>Settings.Designer.cs</LastGenOutput>
    </None>
    <Compile Include="Properties\Settings.Designer.cs">
      <AutoGen>True</AutoGen>
      <DependentUpon>Settings.settings</DependentUpon>
      <DesignTimeSharedInput>True</DesignTimeSharedInput>
    </Compile>
  </ItemGroup>
  <ItemGroup>
    <None Include="App.config" />
  </ItemGroup>
  <ItemGroup>
    <BootstrapperPackage Include=".NETFramework,Version=v4.0">
      <Visible>False</Visible>
      <ProductName>Microsoft .NET Framework 4 %28x86 and x64%29</ProductName>
      <Install>true</Install>
    </BootstrapperPackage>
    <BootstrapperPackage Include="Microsoft.Net.Client.3.5">
      <Visible>False</Visible>
      <ProductName>.NET Framework 3.5 SP1 Client Profile</ProductName>
      <Install>false</Install>
    </BootstrapperPackage>
    <BootstrapperPackage Include="Microsoft.Net.Framework.3.5.SP1">
      <Visible>False</Visible>
      <ProductName>.NET Framework 3.5 SP1</ProductName>
      <Install>false</Install>
    </BootstrapperPackage>
    <BootstrapperPackage Include="Microsoft.Windows.Installer.3.1">
      <Visible>False</Visible>
      <ProductName>Windows Installer 3.1</ProductName>
      <Install>true</Install>
    </BootstrapperPackage>
  </ItemGroup>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
  <!-- To modify your build process, add your task inside one of the targets
below and uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```