Universidade de Aveiro Departamento de Electrónica e Telecomunicações,
2010

**Javier
de Alfonso Miñambres**

Reconhecimento de caras com Componentes Principais
Face Recognition using Principal Component Analysis

" I know it seems hard sometimes but remember one thing: through every dark night, there's a bright day after that. So no matter how hard it gets, stick your chest out, keep your head up... and handle it. "

— Tupac Amaru Shakur

Universidade de Aveiro Departamento de Electrónica e Telecomunicações,
2010

**Javier
de Alfonso Miñambres**

Reconhecimento de caras com Componentes Principais

Face Recognition using Principal Component Analysis

Dissertation developed at the University of Aveiro (Portugal), to partially fulfill the requirements to obtain the degree of Technical Telecommunications Engineer, specialty on Telecommunications Systems at Universidad Carlos III de Madrid. The investigation was carried out under the supervision of Dr. Ana María Perfeito Tomé, professor at the Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA) at the Universidade de Aveiro.

the jury

president	**Doctor Alexandre Manuel Moutela Nunes da Mota** Associate Professor of Universidade de Aveiro
vocals	**Doctor Ana Maria Perfeito Tomé** (Dissertation Advisor) Associate Professor of Universidade de Aveiro
	Doctor Gladys Castillo Jordan Assistant Professor of Universidade de Aveiro

acknowledgements

The dissertation that follows these lines could have never been completed without those little pieces that build up the puzzle of my life, which have somehow contributed to pushing me forward and finally getting me to a point where I am able to close a chapter and open a new one.

The puzzle's big pieces are, as it couldn't be any other way, my parents, Fernando and Carmen. I would like to effusively thank them for all the encouragement they tirelessly gave me. This success I am achieving is a toast to their perseverance to pushing me forward for my own good, even when I couldn't see it clearly.

My brothers, Carlos and Jorge, have a special place in the puzzle too. Our exceptional friendship and endless midnight conversations helped mitigate the occasional frustration that engineering studies may lead to. Thanks a lot for always being there for me when I needed you and for being my best friends.

The next piece has become extremely important to me in a very short time, my girlfriend Bea. Thank you for filling in the blanks of the puzzle when I most needed it. Your support and blind faith in my possibilities have been a true blessing for me. Thank you for your support and understanding, specially in those moments when I faltered. You always stayed by my side and I truly thank you for that.

My school and university friends, many small yet important pieces of the puzzle which together make a great background picture. Much inspiration and eagerness of improvement came from them and I didn't want to miss the opportunity to thank them for that. Great parties and trips helped shake off the pressure of our studies and start back with a fresh air too.

My erasmus friends, present during the development of this dissertation, great support and a family to me (you know who you are) made this whole year so worth it. They are the newest additions to my life puzzle yet have been as important as any other piece.

And of course, I would like to thank my dissertation advisor, Dr. Ana María Tomé, who helped me find the means to write the present text. Thank you for all I learnt from you, for finding so much documentation for me and for switching to English when my Portuguese skills were not that good!

There are many other pieces that form the puzzle, yet since it's bigger than this page, I would like to end by thanking everyone who in some way feels they should be mentioned here but my sometimes forgetting mind didn't include. Thank you all from the bottom of my heart.

abstract

The purpose of this dissertation was to analyze the image processing method known as Principal Component Analysis (PCA) and its performance when applied to face recognition. This algorithm spans a subspace (called *facespace*) where the faces in a database are represented with a reduced number of features (called *feature vectors*).

The study focused on performing various exhaustive tests to analyze in what conditions it is best to apply PCA. First, a facespace was spanned using the images of all the people in the database. We obtained then a new representation of each image by projecting them onto this facespace. We measured the distance between the projected test image with the other projections and determined that the closest test-train couple (k-Nearest Neighbour) was the recognized subject. This first way of applying PCA was tested with the Leave–One–Out test. This test takes an image in the database for test and the rest to build the facespace, and repeats the process until all the images have been used as test image once, adding up the successful recognitions as a result. The second test was to perform an 8–Fold Cross–Validation, which takes ten images as eligible test images (there are 10 persons in the database with eight images each) and uses the rest to build the facespace. All test images are tested for recognition in this *fold*, and the next fold is carried out, until all eight folds are complete, showing a different set of results.

The other way to use PCA we used was to span what we call Single Person Facespaces (SPFs, a group of subspaces, each spanned with images of a single person) and measure subspace distance using the theory of principal angles. Since the database is small, a way to synthesize images from the existing ones was explored as a way to overcoming low successful recognition rates.

All of these tests were performed for a series of *thresholds* (a variable which selected the number of feature vectors the facespaces were built with, i.e. the facespaces' dimension), and for the database after being preprocessed in two different ways in order to reduce statistically redundant information.

The results obtained throughout the tests were within what expected from what can be read in literature: success rates of around 85% in some cases. Special mention needs to be made on the great result improvement between SPFs before and after extending the database with synthetic images.

The results revealed that using PCA to project the images in the group facespace is very accurate for face recognition, even when having a small number of samples per subject. Comparing personal facespaces is more effective when we can synthesize images or have a natural way of acquiring new images of the subject, like for example using video footage.

The tests and results were obtained with a custom software with user interface, designed and programmed by the author of this dissertation.

resumo

O propósito desta Dissertaçao foi a aplicação da Análise em Componentes Principais (PCA, de acordo com as siglas em inglês), em sistemas para reconhecimento de faces. Esta técnica permite calcular um sub-espaço (chamado *facespace*, onde as imagens de uma base de dados são representadas por um número reduzido de características (chamadas *feature vectors*).

O estudo realizado centrou-se em vários testes para analisar quais são as condições óptimas para aplicar o PCA. Para começar, gerou-se um *facespace* utilizando todas as imagens da base de dados. Obtivemos uma nova representação de cada imagem, após a projecção neste espaço, e foram medidas as distâncias entre as projecções da imagem de teste e as de treino. A dupla de imagens de teste-treino mais próximas determina o sujeito reconhecido (classificador vizinhos mais próximos). Esta primeira forma de aplicar o PCA, e o respectivo classificador, foi avaliada com as estratégias *Leave–One–Out* e *8–Fold Cross–Validation*.

A outra forma de utilizar o PCA foi gerando sub-espaços individuais (designada por SPF, *Single Person Facespace*), onde cada subespaço era gerado com imagens de apenas uma pessoa, para a seguir medir a distância entre estes espaços utilizando o conceito de ângulos principais. Como a base de dados era pequena, foi explorada uma forma de sintetizar novas imagens a partir das já existentes.

Todos estes teste foram feitos para uma série de limiares (uma variável *threshold* que determinam o número de *feature vectors* com os que o *facespace* é construído) e diferentes formas de pre-processamento.

Os resultados obtidos estavam dentro do esperado: taxas de acerto aproximadamente iguais a 85% em alguns casos. Pode destacar-se uma grande melhoria na taxa de reconhecimento após a inclusão de imagens sintéticas na base de dados. Os resultados revelaram que o uso do PCA para projectar imagens no sub-espaço da base de dados é viável em sistemas de reconhecimento de faces, principalmente se comparar sub-espaços individuais no caso de base de dados com poucos exemplares em que é possível sintetizar imagens ou em sistemas com captura de vídeo.

resumen

El propósito de este Proyecto de Fin de Carrera era analizar el método de procesado de imágenes conocido como Análisis de Componentes Principales (PCA por sus siglas en inglés), así como su rendimiento cuando es utilizado en reconocimiento facial. Este algoritmo genera un subespacio vectorial (llamado *facespace*), donde las imágenes de una base de datos son representadas con un número reducido de caracaterísticas (llamadas *feature vectors*).

El estudio realizado se centró en diversos tests exhaustivos para analizar en qué condiciones es mejor aplicar PCA. Para empezar, se generó un *facespace* usando todas las imágenes de la base de datos. Obtuvimos una nueva representación de cada imagen tras proyectarlas en dicho espacio, y se midió la distancia entre las proyecciones de la imagen de test y las de entrenamiento. La pareja de imágenes de test-entrenamiento más próxima determinaría el sujeto reconocido. Esta primera forma de aplicar PCA fue probada llevando a cabo el test *Leave–One–Out*, donde se elegía iterativamente una imagen de la base de datos para usarla como test y todas las demas de entrenamiento para generar el *facespace*. El segundo test llevado a cabo el el *8–Fold Cross–Validation*, que seleccionaba 10 imágenes para test (una imagen de cada persona, pues la base de datos tiene 10 personas con 8 fotos por persona) y usaba las restantes para generar el subespacio. En la siguiente iteración se seleccionaban otras 10 para test y las anteriores se incluían de vuelta al grupo de entremaniento.

La otra forma de utilizar PCA era generando subespacios personalizados (SPF, *Single Person Facespace*), donde cada subespacio estaba generado con imágenes de una sola persona, para después medir la distancia entre dichos subespacios mediante la teoría de ángulos principales. Dado que la base de datos es pequeña, se exploró una forma de sintetizar imágenes a partir de las existentes para sobreponerse a las bajas tasas de aciertos.

Todos estos tests fueron llevados a cabo para una serie de umbrales (una variable que selecciona el número de *feature vectors* con los que se construye el *facespace*, es decir, determina su dimensión) y para la base de datos antes y después de haber sido procesada para eliminar información estadísticamente redundante.

Los resultados obtenidos estaban dentro de lo esperado: tasas de acierto de en torno al 85% en algunos casos. Cabe destacar la gran mejoría en la tasa tras haber expandido la base de datos con imágenes sintéticas con respecto a usando sólamente las disponibles para generar SPFs. Los resultados revelaron que el uso de PCA para proyectar imágenes en el subepacio de la base de datos es muy precisa para reconocimiento facial, incluso teniendo pocas muestras por sujeto. Comparar subespacios personales es más efectivo cuando se puede sintetizar imágenes o si hubiese una fuente natural de ellas, como por ejemplo a través de vídeo.

Contents

List of Tables

List of Figures

Chapter 1

Introduction to face recognition

In the modern days, we can see how humanity has tended to delegate as much work as it can to computers, making our lives as easy as we can. It is not surprising that we get to a point where we can rely on computers to do for us one of the most primitive things in human nature: recognize someone when we look at their face.

1.1 Background

In an attempt of creating human-computer interaction, immense progress has been carried out over the past ten or fifteen years. Probably everyone has encountered an example of human-computer interaction software at some point of their daily routines. Situations that before these innovations undoubtedly would have required two people, now they can be performed with a machine or software program on one end. For simple things like calling to order a pizza or to ask for technical support for your home internet network, it used to be necessary to have a receptionist or the technician at the other end of the phone line to take your command or register your inquiry. Now, an automated voice recognition software with pre-recorder sentences can be a suitable substitute for those people.

Other areas with successful development and results on human-computer interaction and, in the end, artificial intelligence, are currently widely used, like text-to-speech software (for example for GPS in-car navigation systems [1] or 'reading out loud' bus stop names in big cities), and speech processing (for example for real-time language translators).

However, other aspects of artificial intelligence are yet not too well developed, or, at least, there hasn't been a proposal that has a high percentage of effectiveness in order to guarantee a successful launching to the market. One of these aspects is computer vision, and, keeping to the subject of this dissertation, *automated facial recognition*.

1.2 Motivations

One can imagine how convenient it can be for everyday tasks to achieve to produce a reliable software that can accurately recognize a person's face. If this technology were to be successfully implemented, an immense range of possibilities opens up to society. Automated facial recognition can lead to great improvement in the law enforcement and commercial fields. Applied to surveillance cameras in secured facilities, police cars, or passport checking can ease personal identification even when the subject is lying. Spotting a person with a criminal record in a shopping center, or even recognizing the identity of a person who disappeared years ago and now pretends to be someone else... application possibilities are as big as one's imagination.

Unlike other forms of personal identification such as fingerprint analysis or iris scanning, facial recognition is non-intrusive and can be performed even when the subjects rejects to, which is indeed a great advantage, specially for law-enforcement purposes.

1.3 Recent Applications

In Tampa and Pinellas County, Florida, police deputies have had contact with this technology already. In 2001, the Tampa Police Department installed surveillance cameras in a nightlife neighborhood in order to reduce crime in the area. Apparently, the cameras, which were equipped with face recognition systems, had an extremely low rate of effectiveness at attempting to recognize criminals and the system was trashed soon despite it cost $ 3.5 million [2, 4]. Nevertheless, in May 2008 the Pinellas County Sheriff's Office decided to get this technology in mobile units to be used by deputies in their routine car patrolling. The results are encouraging as they are registering great success in recognizing drivers' faces when they attempt to lie about their identity [17].

Airports are currently starting to install electronic passport self-check booths, where there is no police officer manually checking your identity, but a machine that recognizes your face and compares it with the information held in the electronic passport's chip.

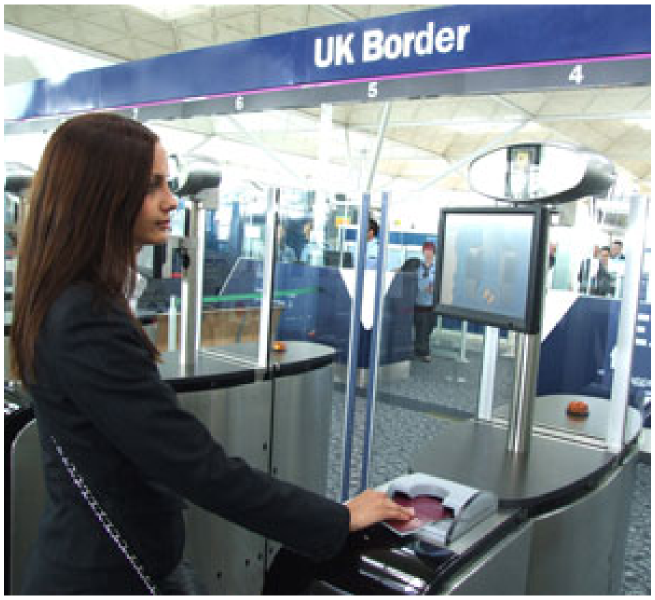


Figure 1.1: Example of airport facial recognition passport checking at Stansted Airport [16]

1.4 Science-Fiction is Science-Fiction

Some more eye-catching applications for facial recognition technology can be found in TV series and Hollywood cinema. In the *Las Vegas* show, they recognize a card-counter using the security cameras to match his face to his picture on a database of the casino. Other examples of TV series where they occasionally use facial recognition are *Chuck*, *Prison Break*, or *24*. Many movies with espionage or FBI topic may also show a very advanced, sophisticated and 100% effective face recognition software, often used for National Security purposes. This, although it may well be the short term future for this technology, it's not its current status and it is still a little premature to rely fully on it as they show in the movies.

Figure 1.2: Example of facial recognition in a film [13]

1.5 Complications

Computer facial recognition is not a simple task at all. Whereas for humans it is an instinctive and innate ability, for computers (and therefore, computer programmers) it represents a very complicated problem. For a human being, it is easy to determine where an object ends and when the next starts, but to a computer a digital image is just a matrix of pixels.

We learn since childhood to discriminate figures and objects from the background. Figure 1.3 is an example of an image where despite the background and the object being very similar, a human being would recognize both. This precise example can be tricky but that is its purpose: to show that if looking at this image it can sometimes be hard for a human to see the polar bear, a computer would see nothing unless specifically trained to do so. Specially, this image I've chosen blends the bear into the ice, both having similar colors. For the computer, this would result in pixels with almost the same value which would be even harder to distinguish.

Humans don't process the image they see as pixels of information, we see the whole image, the big concepts, which is why subjectivity exists. A computer can't have subjectivity, at least not for now or in the near future: it processes data, despite its nature. And data processing can be made in many different ways.

Figure 1.3: Polar bear in the Arctic Ocean [15]

1.6 Approaches

There has been proposed many different ways to perform face recognition [9]. From 2D image analysis like Principal Component Analysis, which is the method studied in this dissertation, to more sophisticated methods like Elastic Bunch Graph Matching (EBGM) [21], and three-dimensional analysis like 3-D Morphable Models [10], amongst many others. Principal Component Analysis is one of the simplest yet most effective ways of performing facial recognition, so a deep study will be developed in this dissertation, testing the algorithm in many different ways.

1.7 Dissertation objectives

The objective of this dissertation is to develop my own face recognition software based on Principal Component Analysis (from now on PCA) and evaluate the effectiveness of the algorithm. Its effectiveness is measured through several tests which exhaustively run the model and record whether it succeeds in recognizing the subject or not. These results are classified depending on what circumstances and options available they were ran under.

These options help tune up the values of variables to achieve a reasonable success rate while keeping computational cost as low as possible. To accomplish this, great importance is given to the *threshold* variable, which determines the dimension of the resulting subspace after applying PCA, in order to have more or less information to perform the recognition. Moreover, we are going to be able to choose what kind of preprocessing method we apply to our database to minimize statistical redundancy.

We will also test two different ways to compare the PCA models span by the database. In the first experiment we will span a facespace using all the images available and project the images there, comparing the distance between the projected test image and all the projected training images to motivate a classification decision. We will draw conclusions on effectiveness after measuring projection distance using the kNearest Neighbour rule, which will be explained in future chapters in detail.

In the second experiment, we will span a facespace for each person and compare their the principal angles to measure the distance between these subspace. We will explore different ways of constructing the databases to span these personal facespaces and interpret the results obtained.

1.8 Dissertation Outline

As we have already seen, chapter 1 introduces the reader into the world of facial recognition, giving him a hint of why this area of computer intelligence is so important and why it has such a brilliant future. Chapter 2 deals with the mathematical basis of the PCA algorithm. Here, the procedure to perform PCA is shown as well as other mathematical operations carried out in the experiments. Following, a full explanation of all the software involved in my work is given in chapter 3, where the insides of my script, the database and other tools are presented. In chapter 4 all the experimental results are shown, along with explanatory figures and tables. Demonstrations of results and software printouts are shown to corroborate explanations. Finally, chapter 5 contains the conclusions obtained after studying and analyzing the performance of the PCA algorithm.

Chapter 2

Principal Component Analysis

2.1 Principles of Principal Component Analysis

The Principal Component Analysis is one of the first yet most successful techniques that have been applied to facial recognition. Its success lies on the easiness of the mathematical theory leading an equivalent representation of the data using less information.

2.1.1 Introduction

The objective of PCA is to reduce the amount of information needed to define an image [20], by means of the extraction of features of the faces. These features, as Turk and Pentland note [19], may or may not be related to our intuitive notion of facial features, like eyes, nose, mouth, etc. PCA has its best application in the case that there exists a strong correlation between the observed variables (as in the case of face images in the database, which are similar in overall configuration).

We know that an image is defined by all of its pixels. This, for an image $\mathbf{\Psi}$ let's say of dimension 64×64, implies it having 4096 units of information to define it. We can represent this 2–D image as a 1–D vector of dimension 4096. This vector is now representable as one point in a 4096-dimension space. Consequently, a database of images is represented as a cloud of points in this huge space. The aim of PCA is to reduce the dimension of this space where the images are represented.

PCA's purpose is to encode the relevant information in a face image as efficiently as possible to reduce the data we handle to define it. When attempting to perform facial recognition, the image is compared to a database of images encoded in the same way [19]. This way, we reduce the large dimensionality of the data space produced by so many units of information by describing the image with the encoded data. This encoded data will be the *features* of the collection of faces. The aim is to extract these features, which are the variation in the face images, to be able to compare them.

By performing an eigendecomposition on the covariance matrix of the data we obtain a set of eigenvectors and eigenvalues. We call *Facespace* to the set of eigenvectors corresponding to non-zero eigenvalues. This facespace is a model of the database and by projecting the images onto the facespace we get the new representation. The features, which are called *feature vectors* and account for the variation in the images of the data set, are the result of the projection of the image into the facespace [11]. These eigenface vectors are orthogonal amongst each other (therefore the eigenfaces are uncorrelated). With this decomposition into a feature space, the amount of information is considerably reduced [20].

Therefore, the database can be represented by means of the *feature vector*s, which are the projection of the database onto each of the *eigenfaces* which construct the *facespace*.

2.1.2 Mathematical basis of PCA

The eigendecomposition referred to in the previous section is performed on the covariance matrix of the database's matrix $\mathbf{X}$. This matrix is built by representing each 2–D image as a 1–D column vector and placing them side to side. This matrix may or may not be preprocessed prior to extracting its features depending on the user's needs. The result of the preprocessing step is called matrix $\mathbf{A}$ in this document. For the following mathematical explanations, $\mathbf{A}$ will be the data matrix, regardless whether it has been preprocessed or not (for this, please refer to section 3.1.2).

Having $\mathbf{A}$ a dimension of $P \times Q$, and usually $P \gg Q$, the covariance matrix

$$\mathbf{C} = \mathbf{A}\mathbf{A}^T \tag{2.1}$$

would have a dimension of $P \times P$ which would be too large to calculate eigenvalues in a decent time and computational expense. Nonetheless, it is mathematically equivalent to calculate the eigenvalues of a surrogate of the covariance matrix $\mathbf{C}$, let's call it $\mathbf{L}$:

$$\mathbf{L} = \mathbf{A}^T\mathbf{A} \tag{2.2}$$

Matrix $\mathbf{L}$ would have a dimension of $Q \times Q$, which is considerably smaller than $\mathbf{C}$. In our case, our $Q = 80$ images, which have $P = M \times N = 64 \times 64 = 4096$ pixels, would create a covariance matrix $\mathbf{C}$ of 4096×4096, which is way too excessive, while $\mathbf{L}$ would have a dimension of 80×80. The eigendecomposition for $\mathbf{C}$ and $\mathbf{L}$ are

$$\begin{aligned} \mathbf{C} &= \mathbf{U}\mathbf{B}\mathbf{U}^T \\ \mathbf{L} &= \mathbf{V}\mathbf{D}\mathbf{V}^T \end{aligned} \tag{2.3}$$

where we know that $\mathbf{U}$ is what we call *Facespace*: the subspace model of the database (matrix of eigenvectors) with a dimension of $P \times R$, being R the number of eigenvectors produced. $\mathbf{V}$ is the surrogate facespace we calculate to take the mathematical detour. $\mathbf{B}$ and $\mathbf{D}$ are diagonal matrices of eigenvalues. $\mathbf{B}$ has a dimension or $P \times P$ and $\mathbf{D}$ of $Q \times Q$. Each matrix has the same $\min(P, Q) = Q$ non-zero eigenvalues in the diagonal. $\mathbf{B}$ has $P - Q$ zero values in the diagonal.

The way to get to $\mathbf{U}$ using the alternative eigendecomposition of $\mathbf{L}$ is using the Singular Value Decomposition (SVD) of the data matrix, who's equation is

$$\mathbf{A} = \mathbf{U}\mathbf{D}^{\frac{1}{2}}\mathbf{V}^T \tag{2.4}$$

multiplying by $\mathbf{V}$ in both sides we get:

$$\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{D}^{\frac{1}{2}}\mathbf{V}^T\mathbf{V} \tag{2.5}$$

where we know that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. Therefore

$$\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{D}^{\frac{1}{2}} \tag{2.6}$$

By isolating variable $\mathbf{U}$ in one side of the equation we obtain

$$\mathbf{A}\mathbf{V}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U} \tag{2.7}$$

where if we reorder the equation we get the so-called dual form of the PCA model, $\mathbf{U}$:

$$\mathbf{U} = \mathbf{A}\mathbf{V}\mathbf{D}^{-\frac{1}{2}} \tag{2.8}$$

$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_Q]$ is the facespace built from the database matrix $\mathbf{A}$. Its columns $\mathbf{u}_q$, $r = 1, 2, \ldots, Q$, are the eigenfaces that describe the facespace.

Having Q eigenvalues at the most, not all of them contain relevant information. If we sort them in descending order, we can define a threshold that will select just the number of eigenvalues who provide the information specified by it. If we do

$$i_t = \frac{\sum\limits_{q=1}^{t} d_q}{\sum\limits_{q=1}^{Q} d_q}, \text{ for } t = 1, \ldots, Q \tag{2.9}$$

where $\mathbf{i} = [i_1 \; i_2 \; \ldots \; i_Q]$ is the information array where we store the cumulative sum, and d_q are the elements of an array $\mathbf{d}$ which holds the eigenvalues that correspond with the eigenvectors ordered from bigger to smaller, and we define a threshold between 0 and 1, we can keep just the eigenvectors whose eigenvalues' normalized information add to that threshold. The resulting number of eigenvectors is R, where $R < Q < P$.

2.1.3 Eigenvectors

As we have already seen, the eigenvectors extracted from the covariance matrix of the data set $\mathbf{A}$ by means of an eigendecomposition (equation (2.3)) represent the variation of the images. The first eigenvector corresponds to the largest variation, which is the principal direction of the data, the second eigenvector to the second largest and so on.

The images are now represented not as a point in an P-dimension space, but as a point in a facespace of dimension R. The eigenvectors characterize the variation between face images, so the distance between points in this facespace represents the difference between one image and another. Each image location contributes in a greater or lesser extent to each eigenvector, so when we want to compare a test image with the database we just represent it as a dot in the facespace. Then, the distance between points is measured to determine which face image in the database is closer to the test image.

2.1.4 New Representation of the Database

The eigenfaces construct an orthogonal basis subspace which can be used to represent the database with fewer coordinates. To do so, all images have to be projected into this new subspace (facespace). Its projections are called $\mathbf{P}$, which are the result of multiplying the eigenfaces with the data matrix.

$$\mathbf{P} = \mathbf{U}^T\mathbf{A} \tag{2.10}$$

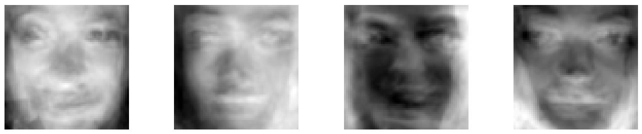
$\mathbf{P}$ results in a matrix of dimension of $R \times Q$. This means that the matrix describing the database has a much smaller dimension ($R \ll P$). Now, each image has its feature vectors placed in the corresponding columns of $\mathbf{P}$. With this, the projected vector of each face will be its corresponding feature vector.

2.2 PCA Models and Face Recognition

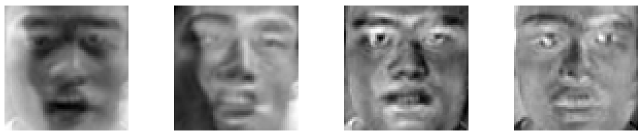
We have seen that applying PCA to a set of images results in a subpsace model called facespace. This facespace can be span with the images we desire. Therefore, we are going to apply two different methods to perform face recognition using PCA.

The first method is to span a single facespace using all the training images. This means that there will be a single model representing the whole database, thus, all the subjects in in. The second method is to span a facespace per subject. We separate the images in the database into subject image groups, and apply PCA to each small group to span, what we call, a Single Person Facespace (SPF).

To perform recognition, we measure similarity among representations of the images. We will differentiate two methods of comparing samples. In the case we span a single facespace for the whole database, we will use a k-Nearest Neighbour Classifier, to measure the distance between projections of the images in the facespace (measure distance between feature vectors). In the case of having multiple PCA models, we will use the theory of subspace principal angles to measure the similarities between a subject's test SPF and all the training SPFs.



(a) Group Database



(b) Single Person Facespace

Figure 2.1: Example of the first four eigenvectors generated with the group database and with a Single Person Facespace (SPF)

Figure 2.1 shows an example of the four largest eigenvectors generated with these two different databases. Ghostly images of the people in the full database are appreciable, while in the second row you can notice clearly a single person's eigenvectors (generated by different images of the same person).

2.2.1 *k*-Nearest Neighbour Classifier

The way to calculate the distance between the test image and the training database after projecting them into the facespace is by means of the k-Nearest Neighbour [6]. The k-Nearest Neighbour classification (kNN) is one of the most fundamental and simple classification methods. It is commonly based on the Euclidean distance between a test sample and the specified training samples.

kNN implements the euclidean distance, which is defined by

$$d = \left\| \mathbf{p}_{train} - \mathbf{p}_{test} \right\|^2 \tag{2.11}$$

where d stands for the distance value and $\mathbf{p}$ is the test or train projected images. The distance between the test image and all the training images is calculated, and the smallest distance will correspond to the closest test-train images couple. This will determine which training image is the most similar (i.e. closer in the facespace) to the test image and therefore encourage a decision. An example of this is shown in figure 2.2. To make it visually representable, the images are defined by only two eigenvectors, in order to plot a 2–D graph. Figure 2.3 shows a zoomed-in area around the plotted eigenfaces. This way the distance between the closest train images and the test image is easier to see.

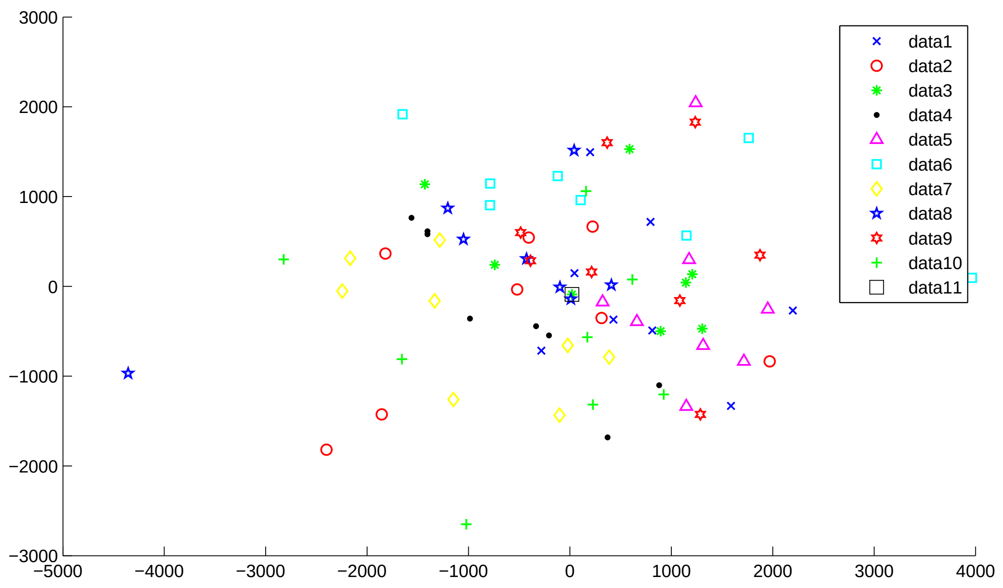


Figure 2.2: Example of distance measuring in a 2-dimension facespace

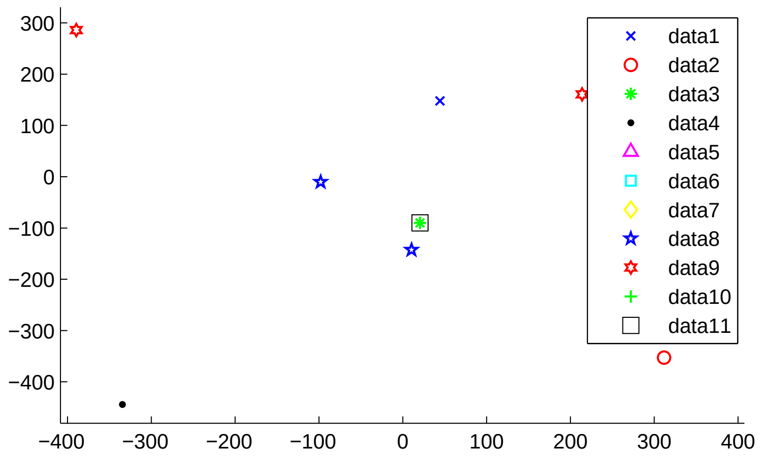


Figure 2.3: Zoom into the example of distance measuring in a 2-dimension facespace

Data 1 to 10 in the legend represent the image location of the projected training images, while data 11 represents the projection of the test image on this facespace. As you can see in the zoomed-in image, the test image projects exactly onto data No. 3. The distance between the projected test image and the rest of the images in the database is calculated. Visually we can see that the closest image to the test image will be recognized as person No. 3.

2.2.2 Multiple PCA Models

A different approach to comparing similarity between people is to span a subspace for each person and compare them. When generating what we defined earlier as Single Person Facespaces (SPFs), we obtain a model of each person represented by a set of orthogonal eigenvectors, span using the available images for each subject. This time, the similarity between subject representations is measured as a *subspace distance*, which is a comparison of the principal angles between facespaces, to determine the grade of similarity they have [5].

The small amount of images available will result in very small facespaces (as k images can result in, at the most, k non-zero eigenvalues). To try to overcome this inconvenient, synthesized images can be created from the available, in an attempt to use more images from each person to span the facespaces. The synthesized images are a result of shifting the original image. This will result in Extended Databases, which we call SIED (Single Image Extended Database) and MIED (Multiple Image Extended Database). To shift the images we will use two variables, m and n, which in the end we obtain a database with mn shifted images from a single original one. This whole process will be deeply explained in section 4.4.3.

Principal Angles

Having two facespaces (orthogonal basis eigenvector subspaces), a set of angles between their dimension basis could be defined. These angles are called Principal Angles (or Canonical Angles), and are denoted by θ_k. Let's consider data matrices $\mathbf{A}$ and $\mathbf{B}$, which contain an image in each column as already seen. After applying PCA to them, we obtain their respective facespaces $\mathbf{U}_A$ and $\mathbf{U}_B$.

In the case where $\mathbf{U}_A$ and $\mathbf{U}_B$ have orthogonal columns[1], the principal angles, which are noted as

$$\theta_1 > \theta_2 > \ldots > \theta_R \in [0, \frac{\pi}{2}]$$

are related to the singular values of $\mathbf{U}_A^T\mathbf{U}_B$. This means that if we perform the SVD (as in equation (2.4)) of the product $\mathbf{U}_A^T\mathbf{U}_B$, as follows:

$$\mathbf{U}_A^T\mathbf{U}_B = \mathbf{SEV}^T \tag{2.12}$$

the values of the diagonal matrix $\mathbf{E}$ are the cosine of each principal angle θ_r.

$$\mathbf{E} = \begin{pmatrix} \cos(\theta_1) & 0 & \ldots & 0 \\ 0 & \cos(\theta_2) & \ldots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \ldots & \cos(\theta_R) \end{pmatrix} \tag{2.13}$$

[1]For the complete theory of principal angles between whichever subspaces, please refer to [8]

The way to calculate subspace distance we implement does this in an equivalent way [12]. We calculate the distance between **A** and **B** using the Frobenious norm and their corresponding facespaces as follows:

$$dist(\mathbf{A}, \mathbf{B}) = \left\| \mathbf{U}_A \mathbf{U}_A^T - \mathbf{U}_B \mathbf{U}_B^T \right\|_F \tag{2.14}$$

which mathematically is equivalent to

$$dist(\mathbf{A}, \mathbf{B}) = \sqrt{r_1 + r_2 - 2\left\| \mathbf{U}_A^T \mathbf{U}_B \right\|_F^2} \tag{2.15}$$

where r_1 and r_2 are $\mathbf{U}_A$ and $\mathbf{U}_B$'s dimensions respectively, and the Frobenious norm $\| \cdot \|_F$ is defined as

$$\left\| \mathbf{X} \right\|_F = \sqrt{trace(\mathbf{X}^T \mathbf{X})}$$

Knowing that the cosine of the principal angles are the singular values of the product of the two facespaces, it's straight forward to see that in order to compute all of the principal angles and calculate their distance we have

$$\left\| \mathbf{U}_A^T \mathbf{U}_B \right\|_F^2 = \sum_k \cos^2(\theta_k)$$

hence leading to

$$dist(\mathbf{A}, \mathbf{B}) = \sqrt{r_1 + r_2 - 2\sum_k \cos^2(\theta_k)} \tag{2.16}$$

which calculates the subspace distance as a single value considering all of the eigenvectors in both spaces. In our software we will apply equation (2.15) because it's the shortest way to calculate the distance (it just takes one code line to implement the formula). We just need to multiply the facespaces. With the other equivalent formulas we would need to perform SVD and make the code longer, which it is not needed. Nonetheless we have seen that the principle of the equation we use relies on principal angles.

Example of subspace distances

Figure 2.4 is an example of the distance measures when performing recognition on subjects 3 and 6. We can appreciate how the distance between the subject's test SPF and the rest remains somewhat constant, while its distance with the same person's train model is clearly smaller. This gives us and idea of how subspace distances look like, and how the decision for recognition is taken.

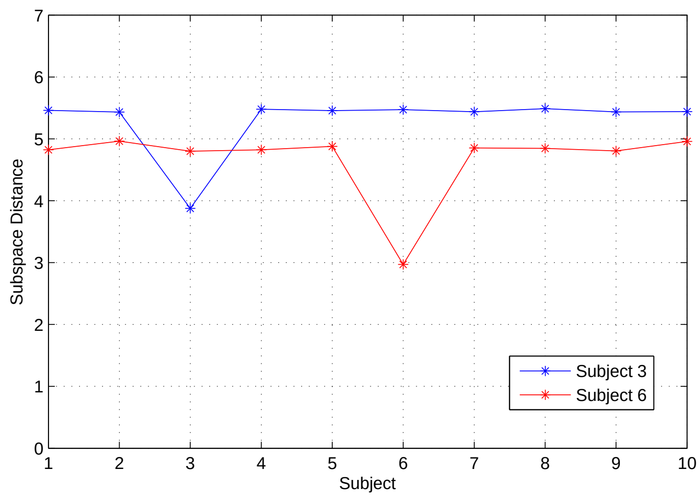


Figure 2.4: Distance between SPF of Subjects 3 and 6, with a fixed database

Figure 2.5 represents the subspace distance calculated between the test space of subject 1 and all the train spaces for all ten subjects in the database. The equation implemented to calculate this distance is eqn. (2.15).

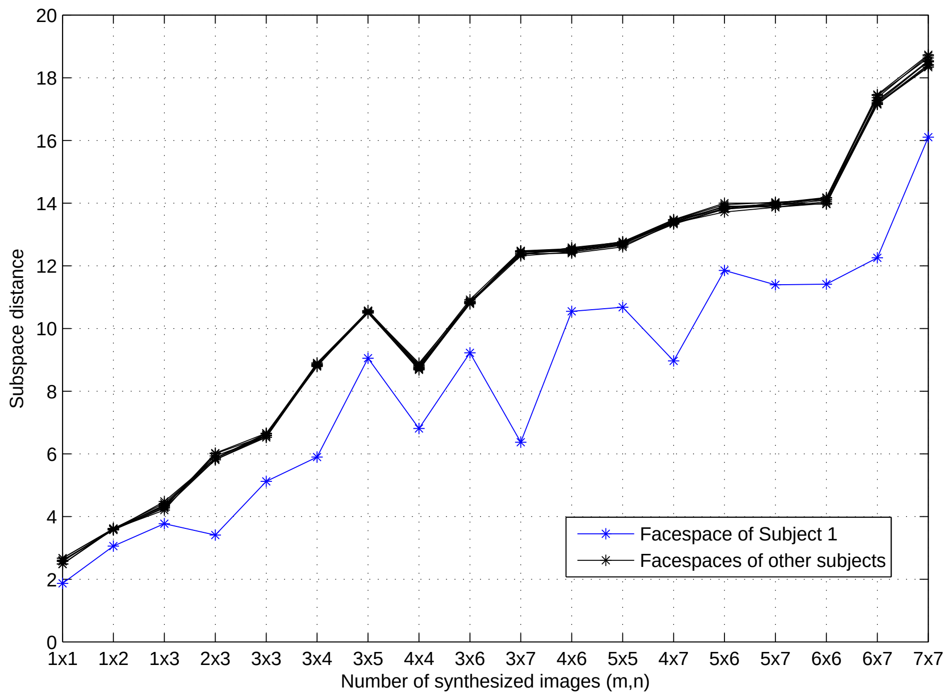


Figure 2.5: Distance between SPF of Subject 1 and the rest while augmenting the number of synthesized images

Each distance measure has been taken for a different number of images to span the faces-pace, as described in section 4.4.4. An increase in the subspace distance can be noted as we use bigger databases. This is due to r_1 and r_2 increasing their value, as they are the subspace size and are added in eqn. (2.15). Despite this, it's clear that the distance between the test and train SPFs of subject 1 is always smaller than between subject 1 and the rest.

2.3 The Mean

Having matrix $\mathbf{X}$ each image placed as its columns, we are going to process these images to get to matrix $\mathbf{A}$. There is a case where we don't process the images to keep control of the influence of the mean in classification. Nonetheless, $\mathbf{A}$ will be the matrix where we will apply PCA to create a facespace from, despite the images in $\mathbf{X}$ being preprocessed or not.

In order to remove non-relevant information from the database and reduce common information that doesn't help classify the subjects, centering or normalizing it can be very helpful. To center a face database we have to calculate the average face and then subtract it to all images. To do so, each pixel value of an image is added up to its corresponding in the rest of the images. This value is then divided by the number of faces in the database.

For instance, let's say that our face database contains Q images named $\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2, \ldots, \boldsymbol{\Psi}_Q$. These images are matrices of pixels. By adding them all up and then dividing the resulting matrix by Q, we obtain the average or mean face, $\bar{\boldsymbol{\Psi}}$:

$$\bar{\boldsymbol{\Psi}} = \frac{\boldsymbol{\Psi}_1 + \boldsymbol{\Psi}_2 + \cdots + \boldsymbol{\Psi}_Q}{Q} \tag{2.17}$$

Then, by subtracting the mean face $\bar{\boldsymbol{\Psi}}$ to each image in the database we obtain a database where all images have their values centered (i.e. around zero). This is very useful since the mean contains a lot of energy and can induce errors when classifying.

$$\begin{gathered} \hat{\boldsymbol{\Psi}}_q = \boldsymbol{\Psi}_q - \bar{\boldsymbol{\Psi}} \\ q = 1, 2, \ldots, Q \end{gathered} \tag{2.18}$$

On the other hand, extracting the mean face can be quite inconvenient. The main handicap it presents if that if the database is extended with new images, the mean face has to be calculated again and subtracted to all the images once again. When image databases are big and keep growing, this can be a tedious process and very resource-consuming. Also, it's another variable to store and use as parameter, having to consider it when programming face recognition software, specially when its sole function is to center the database. It is of no help when classifying.

Nevertheless, there is another way to center the database which solves the handicaps of calculating and subtracting the mean. Considering the database is built by 8-bit images, each pixels has a value between 0 and 255. If we subtract the median value (i.e. 128) and divide by it each image, we achieve normalize the image between -1 and +1. The values are located now around zero too. The normalization also leads to smaller eigenvalues than when subtracting the mean.

$$\begin{aligned} \hat{\boldsymbol{\Psi}}_q &= \frac{\boldsymbol{\Psi}_q - 128}{128} \\ q &= 1, 2, \ldots, Q \end{aligned} \tag{2.19}$$

Normalizing the database doesn't statistically center it around zero, because it doesn't subtract the mean, but it subtracts an approximation which might be good enough. The benefit of normalizing the database is that we don't need to calculate a mean every time we add images to our database, or store it for further use.

When we expand our database with new images, we just need to subtract 128 to it and divide it by 128. No other data is needed to process new images to match the processing in the existing ones.

The processed images $\hat{\boldsymbol{\Psi}}$ from either methods will shape the database prior to applying PCA. With these images we will construct matrix $\mathbf{A}$ and we will apply PCA to it. For the control database, where we don't process the images (i.e. leave the data raw), $\mathbf{X} = \mathbf{A}$, and receives the same treatment as if it were processed by any of the methods seen.

Chapter 3

Source Codes and Software Tools

In this chapter we will get to know how PCA was approached, what mathematical processes were carried out and how. We will leave out mere programming explanations like how accessing the database was done, how the correct names are fetched after recognition, etc.; because although it is also an important part of the code it is not directly related to PCA and the mathematical procedures of recognition. Flowcharts will be included to help understand the main ideas of some of the scripts.

3.1 The Source Code

The software presented in this dissertation was entirely developed by me. A basic PCA script was followed [14] as a small guidance throughout the main steps of applying the algorithm. It is separated into functional modules to ease the comprehension of the functions and the extraction of figures and variables at any point. As an external aid, the MatLab® toolbox STPR Tools is also used in some aspects, but we will get to that later.

3.1.1 Preparing the Variables: SetUpVariables.m and StructureData.m

To access the images a log file is included. In it, the names of each image is included in order. Each image has a name structure of `name_surname_000X.pgm`, and by each person's name we include a number. Each person has a corresponding number, which will then identify it into a class. The log file is arranged with the first image of each person first, then the second and so on. This way, the first 10 lines in the log file have the image number 1 of each person in the database, as shown in listing 3.1. Each person is identified by the number by its side. This helps classify the recognized subject with just looking at its label, instead of having to read the log file again.

The first step is to generate the matrix $\mathbf{X}$ which will hold all the images. Each image in the database is turned into a column and placed one by the other to build matrix $\mathbf{X}$. A simple sketch is shown in figure 3.1 to illustrate how an image is broken into its columns, converted to a column vector and used to construct $\mathbf{X}$.

Listing 3.1: Extract of the log file

```
Angelina_Jolie_0001.pgm 1
Arnold_Schwarzenegger_0001.pgm 2
Britney_Spears_0001.pgm 3
David_Beckham_0001.pgm 4
Leonardo_DiCaprio_0001.pgm 5
Michael_Jackson_0001.pgm 6
Michael_Schumacher_0001.pgm 7
Muhammad_Ali_0001.pgm 8
Winona_Ryder_0001.pgm 9
Yao_Ming_0001.pgm 10
Angelina_Jolie_0002.pgm 1
Arnold_Schwarzenegger_0002.pgm 2
Britney_Spears_0002.pgm 3
David_Beckham_0002.pgm 4
Leonardo_DiCaprio_0002.pgm 5
Michael_Jackson_0002.pgm 6
Michael_Schumacher_0002.pgm 7
Muhammad_Ali_0002.pgm 8
Winona_Ryder_0002.pgm 9
Yao_Ming_0002.pgm 10
Angelina_Jolie_0003.pgm 1
Arnold_Schwarzenegger_0003.pgm 2
Britney_Spears_0003.pgm 3
...
```

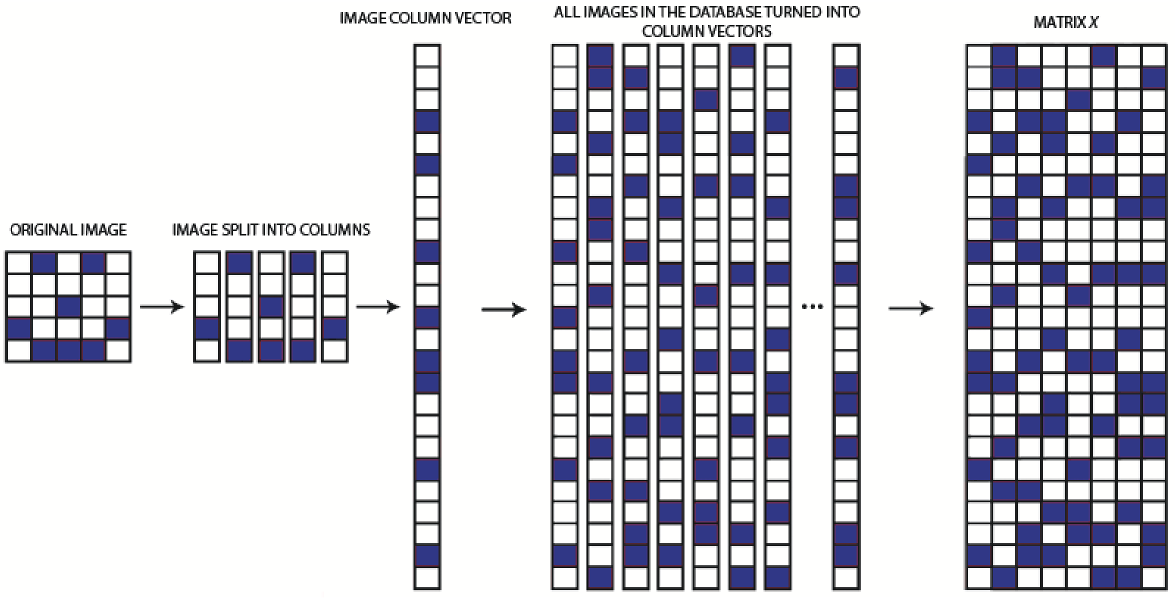


Figure 3.1: Sketch of how matrix **X** is built

Matrix **X** is then constructed by putting together all the columns. Please note that if the image has size $M \times N$, and let's call $P = MN$ (number of pixels per image), the resulting vector will have size $P \times 1$. Matrix **X** will have, therefore, size $P \times Q$, being Q the number of images in the database.

Figure 3.2 graphically shows how the construction of the structure variable is built. Depending on the test performed the code selects the proper images to include in the train database to build **X**.

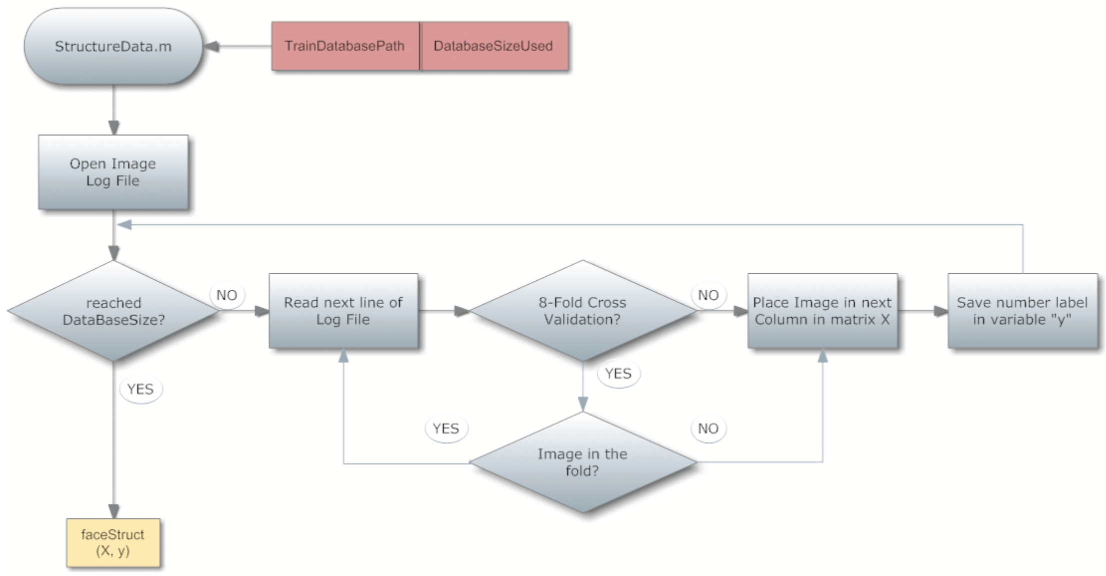


Figure 3.2: Flow Chart of how the access to the database is performed

3.1.2 Data Preprocessing: CenterData.m

Data preprocessing is proved to be an essential step in PCA. This process consists on preparing the data set prior to using it. In order to consider an experiment correctly carried out, the images have to be *evened out* somehow. Two methods have been considered for this purpose: subtracting the mean (or average) face to the data set, and normalizing it between -1 and $+1$. Also, no preprocessing is made to the data set as a control variable in order to compare not only both preprocessing methods among them but also with no preprocessing. We use this ”control variable” to compare the performance between raw and preprocessed data.

Centering

To center the data set, the average face is extracted from the matrix **X**. To do so, the mean is calculated row by row and saved as variable vector **m**, represented in figure 3.3. This average face is subtracted to all of the images in the data set. With this preprocessing, we make sure that all images have statistically average zero, i.e. the values are centered around zero.

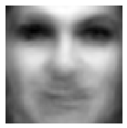


Figure 3.3: Average face

Mathematically, the steps are as follows:

$$m_p = \frac{\sum_{q=1}^{Q} a_{p,q}}{Q} \;;\; \forall\, p \in [1, P] \tag{3.1}$$

$$\mathbf{A} = \mathbf{X} - \mathbf{mj} \tag{3.2}$$

Where matrices $\mathbf{A},\ \mathbf{X} \in \mathbb{Z}^{P\times Q}$, column vector $\mathbf{m} \in \mathbb{Z}^{P\times 1}$, and we define $\mathbf{j} = [1, 1, \ldots, 1]$ as an unitary row vector size $1 \times Q$, elements $m_p \in \mathbf{m}$, elements $a_{p,q} \in \mathbf{A}$, and matrix $\mathbf{A}$ is, as a result, the centered matrix $\mathbf{X}$. Following, a numerical example of this is shown:

$$\mathbf{X} = \begin{pmatrix} 1 & 5 & 3 \\ 4 & 3 & 5 \\ 2 & 0 & 4 \end{pmatrix} \longrightarrow \mathbf{m} = \begin{pmatrix} 3 \\ 4 \\ 2 \end{pmatrix} \longrightarrow \mathbf{A} = \begin{pmatrix} -2 & 2 & 0 \\ 0 & -1 & 1 \\ 0 & -2 & 2 \end{pmatrix}$$

To illustrate this effect on the images, figure 3.5 shows the images involved. Figure 3.4 compares the image before and after it's been centered, and, to prove the purpose of subtracting the mean, histograms are also shown. The images on the left column are the original picture of a subject and its histogram. Subtracting the mean face to this image results on the image in the right column of the figure. Looking at its histogram it's noticeable that the values of the information in this picture are grouped around zero, which serves the point of the preprocessing of the database.

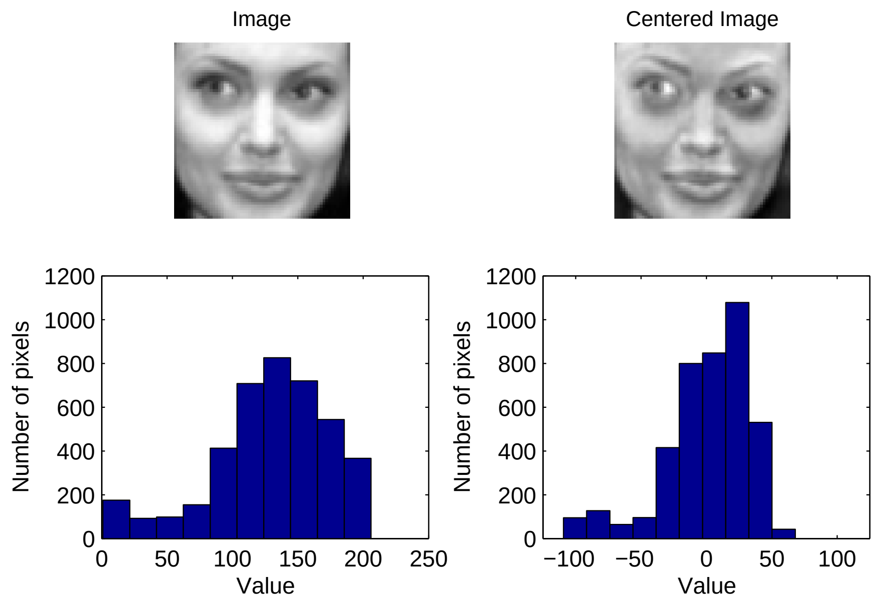


Figure 3.4: Histogram example of the values of an image before and after being centered

Looking at the images, it can be appreciated that the lighting has been evened out, thus all the people's pictures will not have a difference lighting-wise and the mean value of their pixels will be zero.

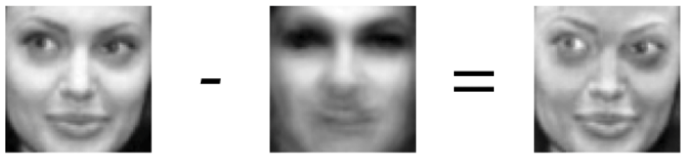


Figure 3.5: Illustration of the centering of the images

Normalizing

Normalizing the database is the other preprocessing method contemplated. It consists in normalizing the values of the images so that they are delimited by -1 and $+1$. The way to do it is to subtract and divide by the central number of the range of values available, in this case 128. To normalize, we do

$$\begin{aligned} &\forall x \in \mathbf{X} \wedge \forall a \in \mathbf{A} \\ &a = \frac{x-128}{128}; \implies a \in [-1,1] \end{aligned} \tag{3.3}$$

where $\mathbf{X} \in \mathbb{Z}^{P\times Q}$ is the database matrix of size $P \times Q$, $\mathbf{A} \in \mathbb{Z}^{P\times Q}$ is the matrix $\mathbf{X}$ after being normalized, and a and x represent each element in matrices $\mathbf{A}$ and $\mathbf{X}$ respectively. As before, the result of this action on the database[1] is shown in figure 3.6.

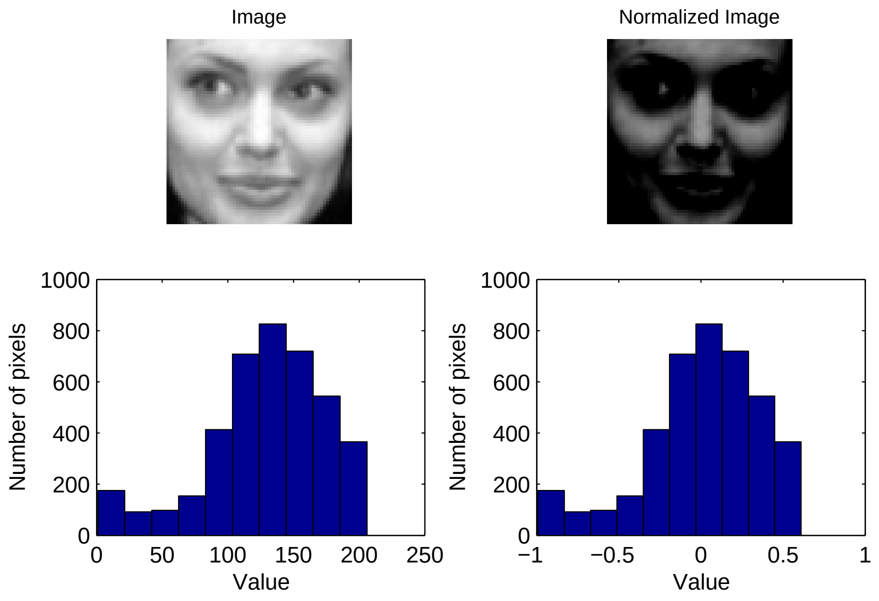


Figure 3.6: Histogram example of the values of an image before and after being normalized

[1]Please note that both the Centered Image and the Normalized Image shown are just for illustration purposes. MatLab interprets the range of values of the image and plots it accordingly, despite them being negative. Therefore, the normalized image should have been shown exactly the same as the original one since the ratio of the pixels remains the same.

Figure 3.7 shows a flowchart of how the script `CenterData.m` is structured. This script is in charge of centering, normalizing or leaving the data raw.

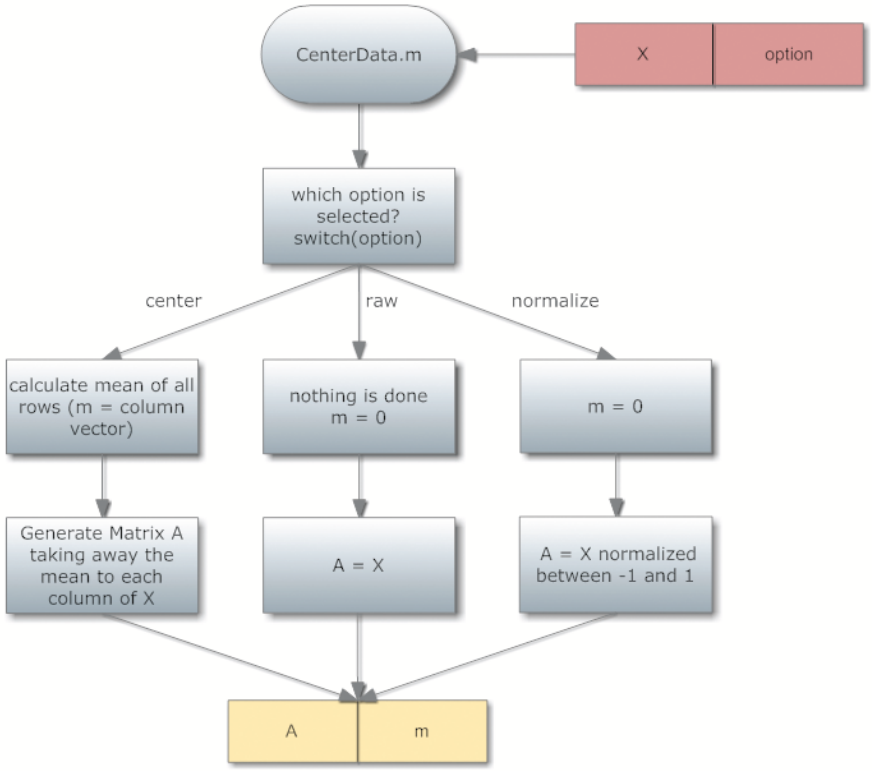

CenterData.m
X
option
which option is selected? switch(option)
center
raw
normalize
calculate mean of all rows (m = column vector)
nothing is done m = 0
m = 0
Generate Matrix A taking away the mean to each column of X
A = X
A = X normalized between -1 and 1
A
m

Figure 3.7: Flow Chart of the preprocessing methods

3.1.3 Eigenface Extraction: ComputeEigenfaces.m

As seen in section 2.1.2, although the way to calculate the eigenvalues and eigenvectors for this purpose is directly from the covariance matrix of **A** (from now on **A** will be the preprocessed matrix, despite the method used), it is computationally more efficient and mathematically equivalent to calculate the eigenvectors from a surrogate matrix of this covariance.

Therefore, the eigenvectors are extracted from **L** as described in equation (2.3). The eigenvectors are then ordered in decreasing order to make sure the principal directions are placed first. The number of eigenvectors is optionally truncated using the variable *threshold*, which selects the necessary eigenvectors (in order from biggest to smallest) to match a desired percentage of information supplied by them. The meaning of this threshold is more deeply studied in section 3.2.

In the case of the single person facespaces, we separate each person's pictures in groups. Depending on the value of the parameter selected, a specific number of images of a single person will be used to generate its personal facespace for training. The selected person for recognition will use the remaining images to construct its test facespace. Having the images separated, the eigendecomposition is carried out as before, as if it were the global facespace, but repeteadly to generate each person's facespace. By doing this we obtain a set of training facespaces and the test facespace. To illustrate how the script works, figure 3.8 shows a flowchart of how the eigenface extraction is approached.

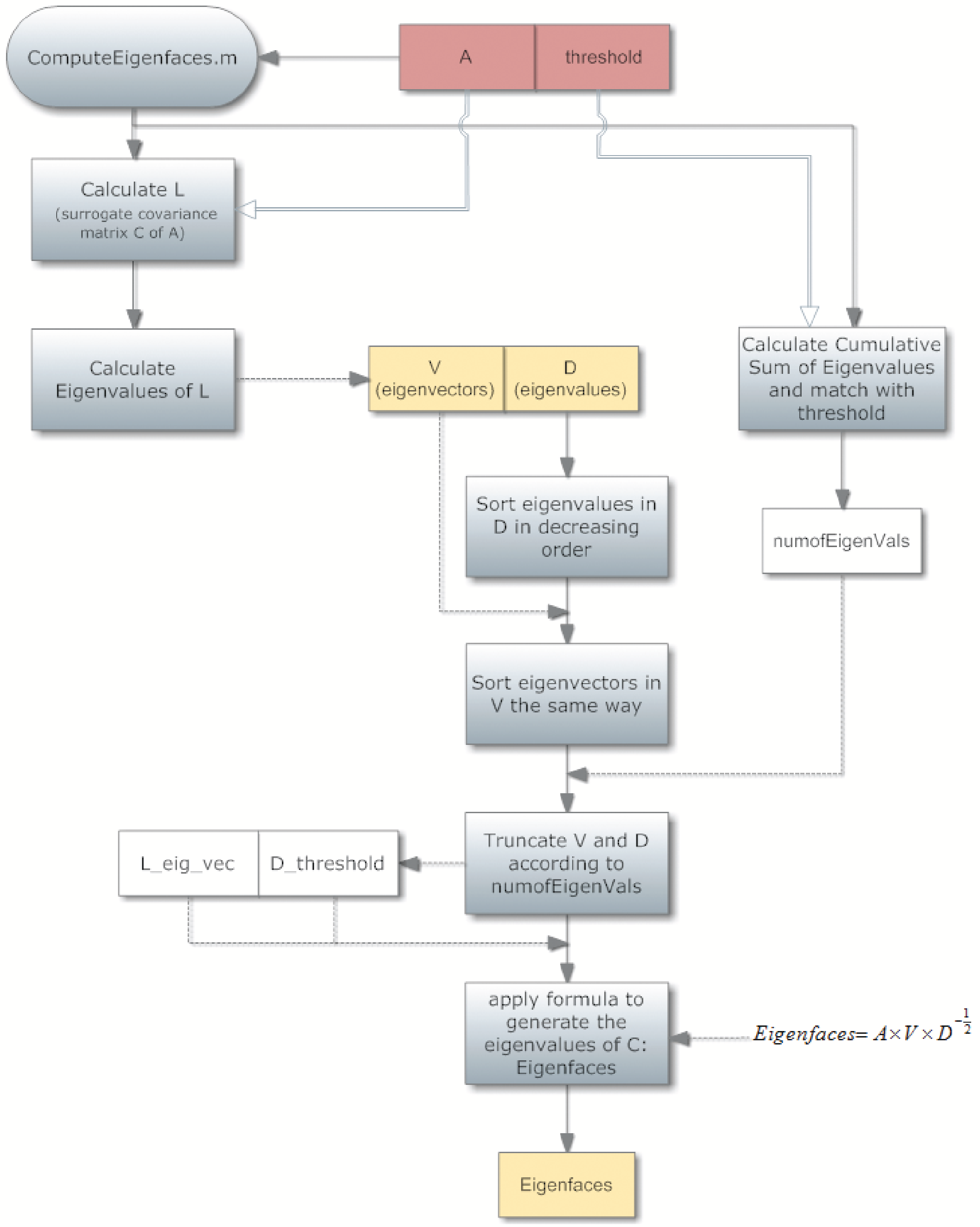

ComputeEigenfaces.m
A
threshold
Calculate L
(surrogate covariance matrix C of A)
Calculate Eigenvalues of L
V (eigenvectors)
D (eigenvalues)
Calculate Cumulative Sum of Eigenvalues and match with threshold
Sort eigenvalues in D in decreasing order
numofEigenVals
Sort eigenvectors in V the same way
L_eig_vec
D_threshold
Truncate V and D according to numofEigenVals
apply formula to generate the eigenvalues of C: Eigenfaces
$Eigenfaces = A \times V \times D^{-\frac{1}{2}}$
Eigenfaces

Figure 3.8: Flow Chart of the Eigenface extraction implementation

3.1.4 Recognition: Recognition.m

The recognition script is divided in two parts. One, it calculates the distance between projected images when using a single database facespace. The other, compares principal angles when using SPFs to calculate subspace distances.

Images projected onto the database model

In the case that we span a model of the database, all the images are then projected onto the facespace as defined in the following equation, which has already been talked about in section 2.1.4:

$$\mathbf{P} = \mathbf{U}^T\mathbf{A} \tag{3.4}$$

Now, $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ldots, \mathbf{p}_R]$, where $\mathbf{p}_r$ are column vectors with the projected coordinates of each image, i.e. the eigenfaces. Then, the test image is converted into a column vector $\mathbf{t}_{test}$ and centered like the rest of the database. In the same way as before, the centered test image is projected onto the facespace as vector $\mathbf{t}$

$$\mathbf{t} = \mathbf{U}^T\mathbf{t}_{test} \tag{3.5}$$

and the euclidean distance is measured. The distances between the test and all the train images is calculated by means of the functions `knnrule` and `knnclass` from the Statistical Pattern Recognition Toolbox [7] for MatLab®, which receives a structure variable (with the training matrix and a vector of labels identifying which images correspond to each person) and the test image and performs kNN distance classifier and outputs the label of the closest person to the test image. Nevertheless, the basics of how it works are as follows: vector distance $\mathbf{d}$ is defined by

$$\begin{gathered} \mathbf{d} = [d_1, d_2, d_3, \ldots, d_R] \\ d_{min} = \min_r \left\|\mathbf{t} - \mathbf{p}_r\right\|^2 , \forall\, r \in [1, R] \end{gathered} \tag{3.6}$$

Since it's easy to find out in which position that minimum distance is and we know which position corresponds to each person (thanks to the labels or because we inserted them in a specific order we know) it is pretty straight forward to find out the identity of the person.

Comparing Models

To compare personal models, we span a facespace for each person in the database. To do this, we select a number of images for test and the rest for training. This selection is random among the pictures. This way, the results are not restricted by the facespace span by the same images all the time. Each time different images construct the SPFs. Once we have selected the images, PCA is performed as usually to obtain the eigenfaces. This time, we generate the test subject SPF and the training subjects SPFs, and we compare their similarity using Principal Angles. The mathematical expression that we implemented to calculate subspace distances is [12]:

$$dist(\mathbf{A}, \mathbf{B}) = \sqrt{k_1 + k_2 - 2\left\|\mathbf{U}_A^T \mathbf{U}_B\right\|_F^2} \tag{3.7}$$

as already talked about in section 2.2.2. Following these calculations, the smallest distance of all will determine the subspace that is more similar to the test subspace, and, therefore, who will the software decide it is.

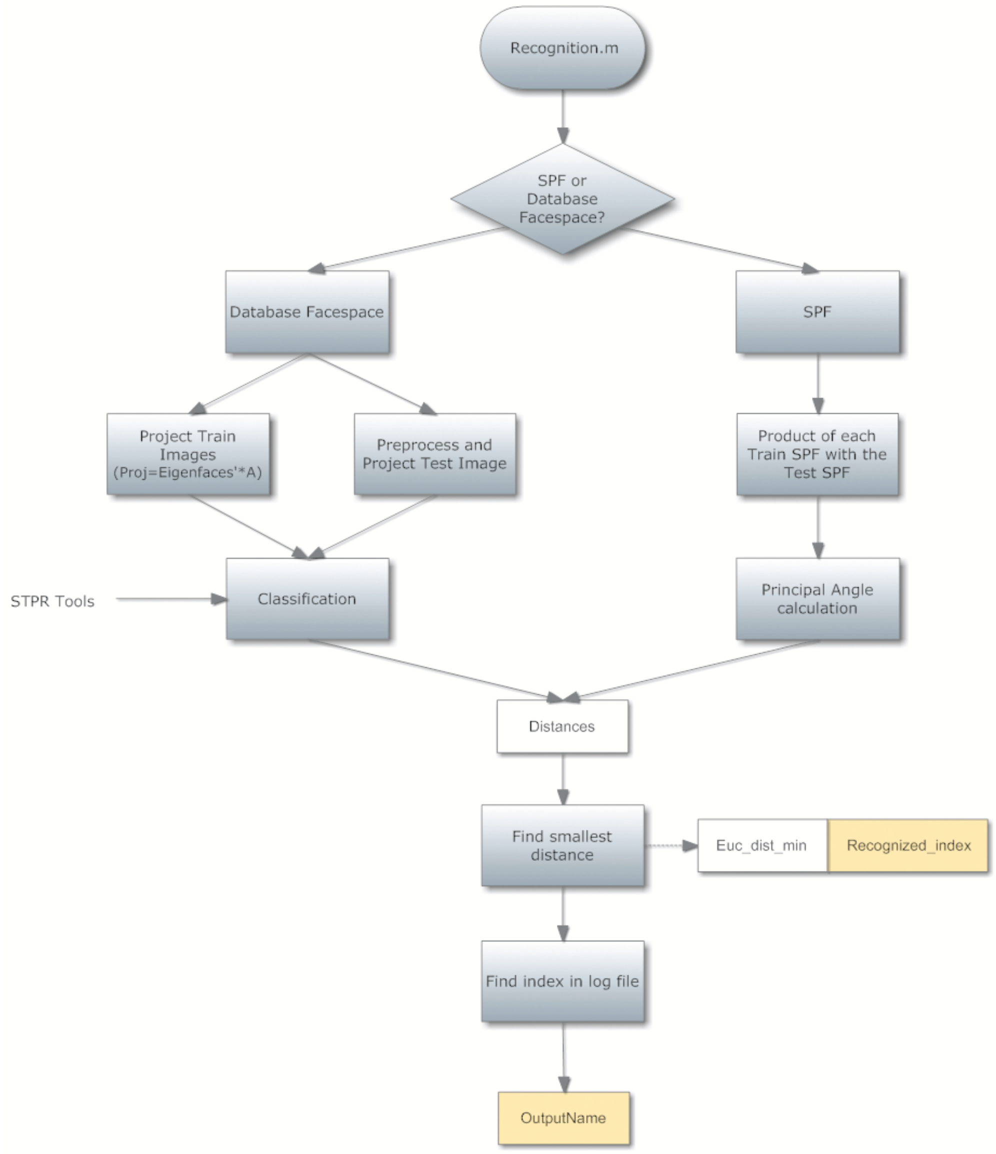

Recognition.m
SPF or Database Facespace?
Database Facespace
SPF
Project Train Images (Proj=Eigenfaces'*A)
Preprocess and Project Test Image
Product of each Train SPF with the Test SPF
STPR Tools
Classification
Principal Angle calculation
Distances
Find smallest distance
Euc_dist_min
Recognized_index
Find index in log file
OutputName

Figure 3.9: Flowchart of the implementation of the recognition script

3.1.5 Overview of the source code: StartDetection.m

Having seen all the small processes, let's take a look now at how they integrate together to construct the full software for face recognition. The launcher script calls every function in the necessary order depending on how we want to perform the recognition. The actual code is a bit different in structure to what it's shown in figure 3.10. The actual `StartDetection.m` file, as you can see in appendix A.1, contains a section for each test where all of the other scripts are called accordingly to that test's needs.

Retrieving information from the user

The program starts by asking the user to choose the test to perform. It also asks about other details to perform the tests such as subject to recognize, threshold (or thresholds) to use, and several others depending on which test you want to carry out. It has also a debugging mode, which is meant to be used to run the code quickly without having to input the options every time. Using this mode, the user needs to open the source code and change the variables manually. This debugging mode is helpful in case you want to run the software repeated times with the same options. With it, you don't need to input them each time, just run the code and it will not ask questions.

Function Calling

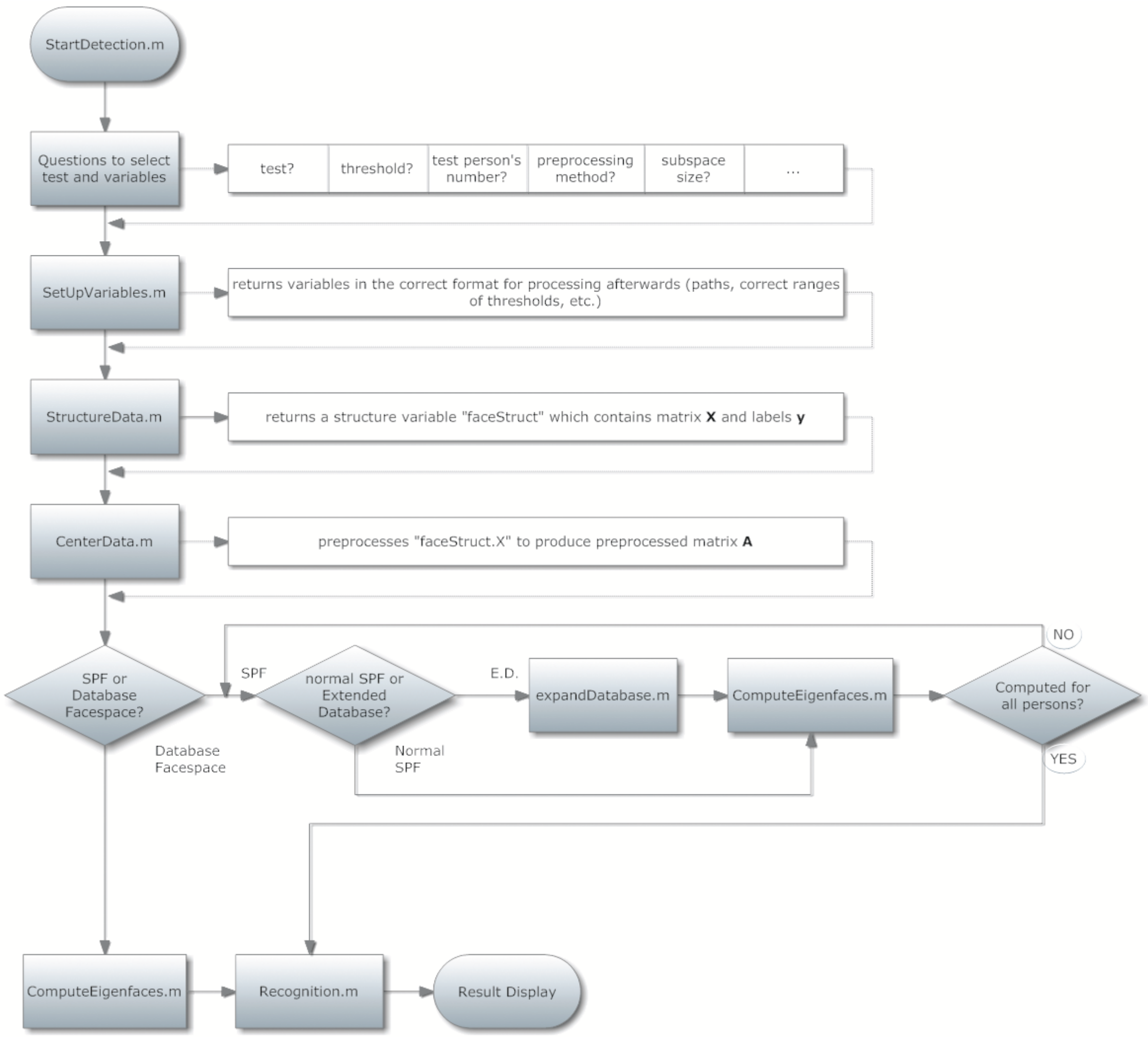
After all the options have been selected, the programs goes to the selected test's section by means of an `if` instruction. The other functions are called with the appropriate input variables in order. The figure shows the basic steps that each test method takes.

First, all of them call `SetUpVariables.m`, which makes sure that threshold percentages are between 1 and 100, and initiates other variables as image paths and database size. Following, `StructureData.m` builds a structure variable with matrix $\mathbf{X}$ and its corresponding labels as seen in figure 3.2. The next step is to preprocess this data, by means of `CenterData.m`, to obtain matrix `A`. Eigenfaces are then extracted from $\mathbf{A}$ with `ComputeEigenfaces.m`. To end, `Recognition.m` performs the convenient recognition method (projection distance or SPF comparison) and `ResultDisplay` shows useful information about the recognition.

In the case of spanning SPFs, the script performs an intermediate step after centering the data. It separates the test and train images for each person, selects the images and shifts them to generate a SIED or a MIED if necessary using `expandDatabase.m`. Then, it systematically calls `ComputeEigenfaces.m` to span the facespace for each person. After obtaining all the SPFs needed, it proceeds with the recognition.

Flowchart

The figure shows a more concise and efficient way to sort the code than what is really implemented. The reason of having repeated chunks of code is to ease debugging and further additions of new methods. Having so many different methods and options (various projection distance tests, and three different ways of spanning SPFs) it would have been tedious and more difficult to understand if it had been implemented as the correct theoretical way. Nevertheless, the effect is the same and for explanatory purposes the flowchart shown is easier to understand.



StartDetection.m
Questions to select test and variables
test?
threshold?
test person's number?
preprocessing method?
subspace size?
...
SetUpVariables.m
returns variables in the correct format for processing afterwards (paths, correct ranges of thresholds, etc.)
StructureData.m
returns a structure variable "faceStruct" which contains matrix X and labels y
CenterData.m
preprocesses "faceStruct.X" to produce preprocessed matrix A
SPF or Database Facespace?
SPF
normal SPF or Extended Database?
E.D.
expandDatabase.m
ComputeEigenfaces.m
Computed for all persons?
NO
YES
Normal SPF
Database Facespace
ComputeEigenfaces.m
Recognition.m
Result Display

Figure 3.10: Flowchart of the main script

3.2 Threshold

The threshold is a variable that we use to truncate the number of eigenvectors available to span the eigenface. The eigenvalues are ordered in decreasing order, so that the first ones are the largest: the ones that provide more information. As we go down the list each eigenvalue gets smaller and smaller, meaning that they contain less important information or features. Having this, to span the facespace the threshold selects the adequate number of eigenvectors depending on the information provided by their associated eigenvalues.

The threshold calculates the cumulative sum of the eigenvalues in that order, and sets a percentage value which will select only the number of eigenvectors whose associated eigenvalues' information sums to that percentage. The threshold is usually set between 80% and 95%. If the threshold is risen higher than 95%, the number of eigenvectors (and therefore the facespace dimension) will rise exponentially. Thresholds of under 70% will in most cases result in using just a handful of eigenvectors, which might be insufficient to perform recognition.

Figure 3.12 shows the normalized cumulative sum of the eigenvectors plotted against the number of eigenvectors needed. This is calculated in MatLab by means of this expression: `i = cumsum(D)./sum(D)`, although the actual mathematical formula is the one explained in section 2.1.2.

It can be seen that the curves in the figure have a great slope at the beginning and are more horizontal at the end, which denotes that the first few eigenvectors provide a great amount of information while the last don't. This implies that changing a threshold from 30 to 40% may not increment the number of eigenvectors, or even up to 80% around just 10 eigenvectors are needed. On the other side of the curve, rising from 80 to 100% implies using around 70 eigenvectors.

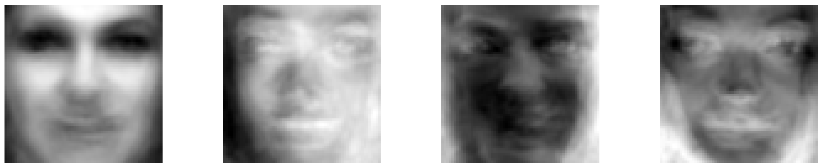


Figure 3.11: First four eigenvectors generated with no preprocessing of the database

Special mention has to be made about the curve representing the eigenvectors of the database without centering or normalizing (red line: control or raw data). Since the mean hasn't been subtracted and it hasn't been normalized, we can see that the first eigenvectors provide pretty much all the information. In order to be able to generate a facespace with more than one dimension it is necessary to rise the threshold more than 95% which is a lot. This is because the mean face contains a lot of information, because it stores statistical data of the images. Actually, figure 3.11 shows these first four eigenvectors, and it can be seen that the first one is the mean face showed in earlier sections.

Also, figure 3.12 shows that centering the data by subtracting the mean and normalizing the data between -1 and +1 give a similar eigenvector information distribution. This could lead to think that it is more worthwhile to normalize the database rather than to center it. This is because computationally it is more expensive to calculate the mean face of the database and store it to subtract it to every image. On the other hand, normalizing is a numerical process standard to any image, it doesn't require any extra data stored to perform it. This cheaper computational expense might be worth getting a higher curve in the eigenvector cumulative sum. This means that both methods have a similar effect on the database but normalizing is faster and less resource-consuming than centering.

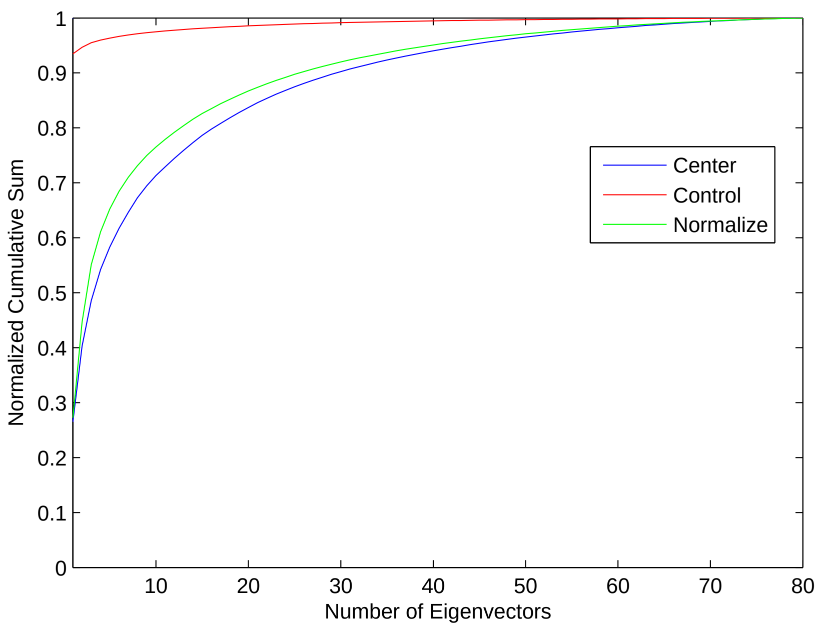


Figure 3.12: Comparison of the information supplied by the eigenvectors depending on the preprocessing method

3.3 STPR Toolbox

The MatLab® toolbox used to perform kNN classification is the STPR toolbox, which stands for Statistical Pattern Recognition Toolbox [7]. This toolbox implements many routines which ease the analysis of PCA, graph plotting and database printouts. It became very handy when printing, for example, the eigenfaces in matrix **U**, using the function `showim`. When classifying by kNN, this toolbox has a couple of functions called `knnrule` and `knnclass`, which create a model of the k-nearest neighbour classifier and compare and classify the test image against that model, respectively.

3.4 Image Database: LFWcrop Face Dataset

The database used is the LFWcrop Face Dataset, which is a modification of the Labeled Faces in the Wild (LFW) dataset. LFWcrop improves the original LFW by removing the background of all the images and keeping just the face, preventing face recognition softwares to exploit the possible correlation between certain backgrounds and faces. This dataset contains grey-scale images of size 64×64 pixels in `pgm` format of famous people photographed under uncontrolled lighting, position or facial expression circumstances.

The origin of the images makes them very uneven in terms of the facial expression of the subject, such as having pictures of the same person smiling and showing the teeth, with and without a moustache, or occasionally wearing sunglasses. Some subjects have their face partially covered by hair, and others have the head pretty turned around. This makes the database very inconsistent unlike other databases like Yale Face Database, which have controlled pictures of each person with specific lighting and configuration.

This inhomogeneity in the images may reduce the success rate of the PCA, since the greater the difference among pictures the less correlation there is between test and train images and therefore decision taking is not that obvious, possibly leading to a greater error rate. Nevertheless, this database is somehow more realistic for real-life applications, having images alike to those that could be obtained by a police officer's camera or by video-footage.

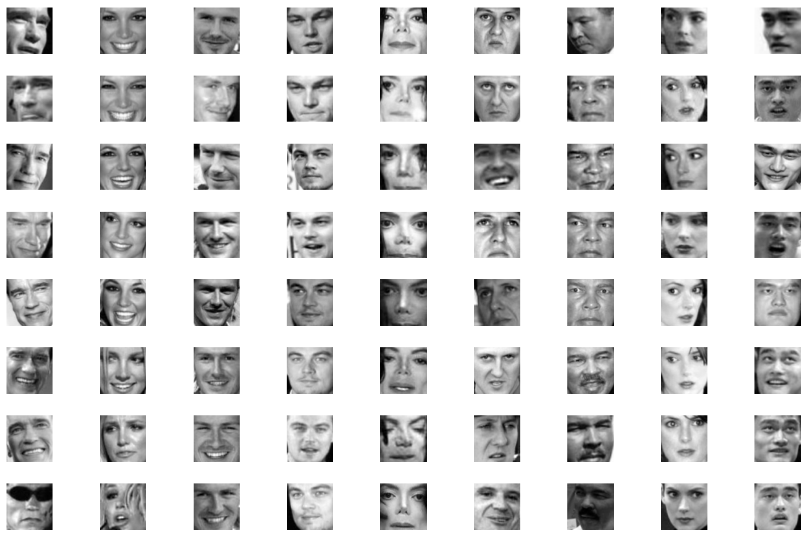


Figure 3.13: Full image database

Database Organization

For this dissertation, out of 13234 contained in the database, only 80 have been used in order to have a list of people easier to handle and because not everybody had the same number of images. These 80 images comprehend ten people with eight pictures each. Each person's images are number from 1 to 8, and each person has been assigned a number label, from 1 to 10 in a log file. With these labels, the data is always related to the person it corresponds to, despite us using one picture or another of that person.

All the images have been placed in an *Images* folder where the code accesses to. There, all the images are together to make preprocessing easier. Then the software will separate test from train images. To select the images, their names have to be included in a log file, called, *ImagesLogFile.txt* where a list of the image names and their label is. This log file is accessed by the software and easily helps the user select which images should be considered for recognition and which don't. This way to include or exclude images they only need to be added or taken away from the log file, instead of having to delete or add the images themselves.

Chapter 4

Experimental Results

4.1 Single Recognition

The first of all steps in order to achieve experimental results and get a good idea of what's going on was to perform single person recognitions at a time. Performing a single test allows us to see whether the recognition was successful or not with specific values in the variables. For this kind of test, where we test just one image, the recognition parameters are printed out.

This is the result of a single recognition performed on a picture of Britney Spears.

RECOGNITION SUCCESSFUL!!!	
Selected person	Britney Spears
Identified person	Britney Spears
Recognition Parameters	
Threshold Percentage	50
Processing Criteria	center
Eigenface Subspace Dimension	4

Table 4.1: Example of data printout in single recognition mode

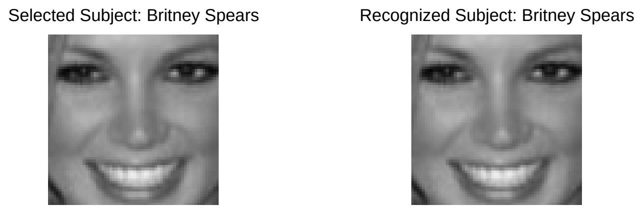

Selected Subject: Britney Spears
Recognized Subject: Britney Spears

Figure 4.1: Single Recognition

Figures 4.2 and 4.3 show the image database used to train the PCA. Figure 4.2 is the original database and 4.3 is the database with its images preprocessed (in this case centered) to each one of them:

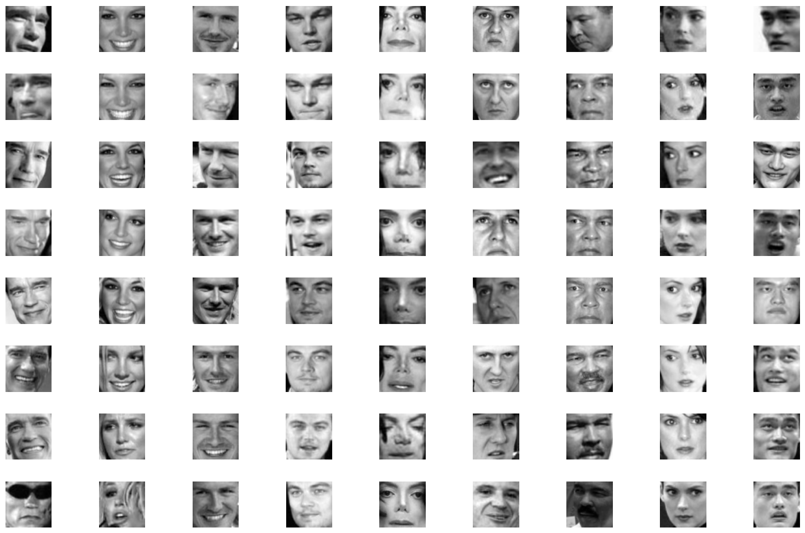


Figure 4.2: Database used for recognition

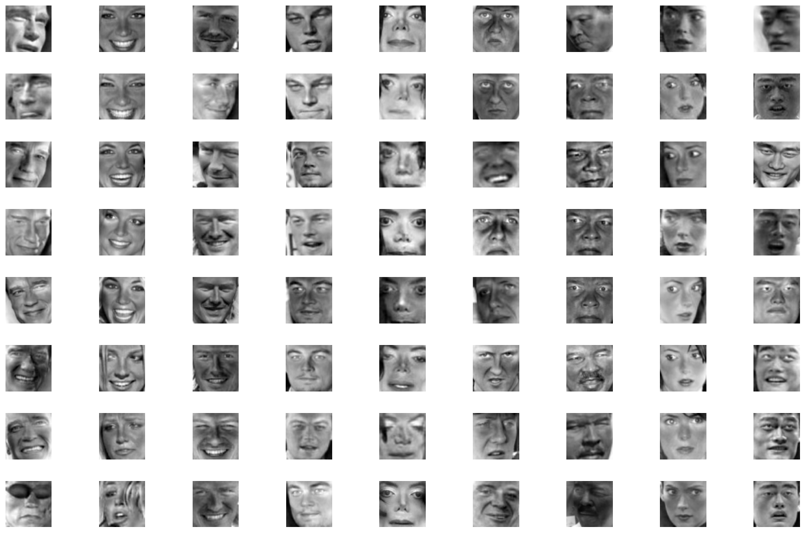


Figure 4.3: Centered images for recognition

Figure 4.4 show the eigenfaces generated by the database available. More eigenfaces were generated but these ones provide the desired amount information selected in the *threshold* variable. For this instance, these four eigenfaces contain 50% of the information that is held in the eigenvectors of the database matrix.

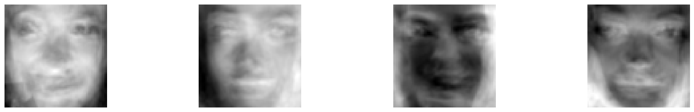


Figure 4.4: Eigenfaces generated with the database and truncated with the threshold

But this information, while it being useful to know how a single recognition was carried out, it doesn't give us an idea of the effectiveness of PCA. Therefore, more exhaustive tests should be performed.

4.2 Leave–One–Out Test

The Leave–One–Out Test carried out is a very popular data mining test [22] which measures the error rate of a learning scheme, like the matter in hand. The idea behind it consists on selecting an image from the database to use as test while training the model with the rest. At the next instance, the test image used will be the next in the database and the previously used as test is included back to the training set. This is a particularization of the n-Fold Cross-Validation, being $n = 1$ in this case.

4.2.1 Methodology

So, for our database of 80 images, the first instance will use image 1 as test and will train the model with images 2 through 80. If the recognition is successful, +1 will be noted. For the next instance, image 2 will be used as test image and images 1 and 3-80 will be used as training set. This will be repeated until all the folds are complete. Also a good amount of thresholds are tested. The number of correct judgements for each threshold is then averaged, which gives us the final error estimate for the model.

4.2.2 Results

The positive recognitions for each threshold, as well as its average is represented on table 4.2. Please note that the "Positive Recs" row shows the number of positive recognitions out of the 80 attempts it performs for each threshold. These values are turned into a percentage in the row "Positive Average", while "Error Estimate" shows the percentage of times it failed to recognize the subject. This table shows results for a centered database.

Threshold (%)	20	30	40	50	55	60	65	70	75	80	85	90	95	100
Subspace Size	1	2	3	4	5	6	8	10	13	17	22	30	44	78
Positive Recs	13	12	13	18	14	22	28	26	25	24	30	28	29	32
Positive Average (%)	16	15	16	22	18	28	35	32	31	30	38	35	36	40
Error Estimate (%)	84	85	84	78	82	72	65	68	69	70	62	65	32	60

Table 4.2: Results of the Leave–One–Out Test

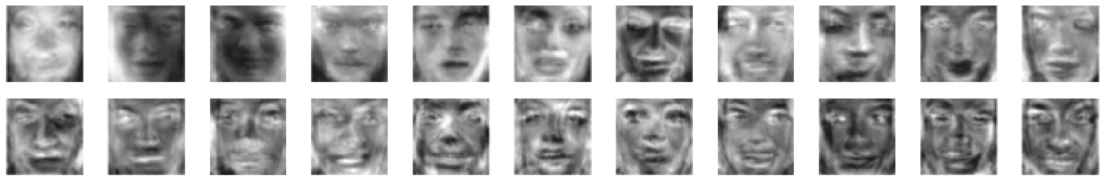


Figure 4.5: Example of Eigenfaces

Figure 4.5 shows an example of the first 22 eigenvalues generated by the database. The threshold selected to generate such facespace is of 85%. Figure 4.6 plots the successful recognitions percentage for each threshold tested and for the three preprocessing methods used. The data can be found in table 4.3.

Threshold	20	30	40	50	55	60	65	70	75	80	85	90	95	100
Centered (%)	16	15	16	22	18	28	35	32	31	30	38	35	36	40
Normalized (%)	8	8	8	13	16	20	25	31	36	33	32	33	36	40
Raw (%)	10	10	10	10	10	10	10	10	10	10	10	10	21	40

Table 4.3: Results for Leave–One–Out test (in percentage)

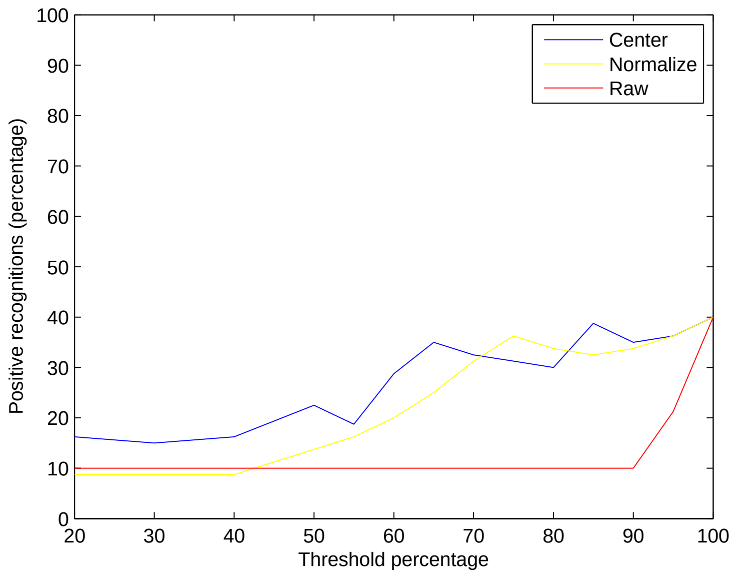


Figure 4.6: Evolution graph for Leave–One–Out test

The same test has been repeated including the tested image in the training group. In this case, the test image was not excluded from the training database. In table 4.4 the detailed results for Leave–One–Out test including for a centered database are shown.

This experiment, although it is not strictly Leave–One–Out test because we don't leave the test image out of the training set, was performed to see if the low recognition rates are due to a possible uneffectiveness of the PCA algorithm or maybe because the image dataset is not optimal for recognition.

Threshold (%)	20	30	40	50	55	60	65	70	75	80	85	90	95	100
Subspace Size	1	2	3	4	5	6	8	10	13	17	22	30	44	78
Positive Recs	67	67	67	55	56	54	55	54	57	56	54	55	55	54
Positive Average (%)	84	84	84	69	70	68	69	68	71	70	68	69	69	68
Error Estimate (%)	16	16	16	31	30	32	31	32	29	30	32	31	31	32

Table 4.4: Results of the Leave–One–Out Test

The results were much better. Without getting into detailed results in tables, the success rates are shown in figure 4.7.

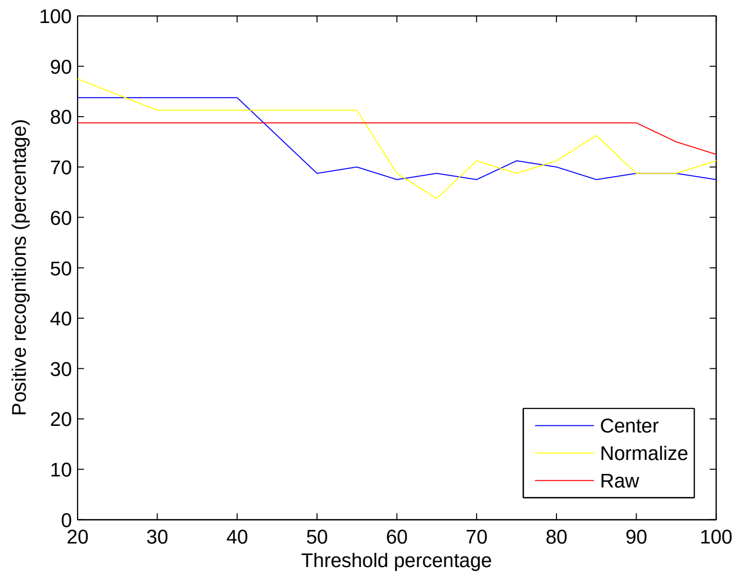


Figure 4.7: Evolution graph for Leave–One–Out test

The results shows a tendency in the success average for both centered and normalized databases: The raw (control) database holds its rate constant which is surprising, specially when rising the threshold, where the preprocessed databases lower their rate. Nevertheless all three recognition rates are satisfactory within what expected. This is because the algorithm finds the similarity between the test image and its copy in the training set more easily than in the first test where it has to find the similarity of a person with other pictures of him. This leads us to think that the problem of poor recognition is derived from the database's face posing.

4.3 8–Fold Cross–Validation Test

The 8–Fold Cross–Validation test is a generalization of the data mining technique in section 4.2. The variation now is that instead of selecting just one image for test and training the model with the rest, a group of ten[1] images (one for each person in the database) is selected as *test images* and the rest as *train images*.

4.3.1 Methodology

To perform this test, image number one of each person in the database is selected for the test group. The model is trained with the remaining 70 images. During this fold, the algorithm is run to test all the test images. When all of them have been tested, the next fold is carried out and image number two for each person is selected for the test group. The same procedure is carried out until all the image groups have been used for test and all the threshold have been tested. The results of the recognition are recorded, both whether the recognition was successful or not and who was the tested person and who was the predicted.

4.3.2 Results

The results of this test are presented in table 4.5. Extra information is added this time in addition to previous tables shown. Here, the numbers by the names of the people in the database represent how many times each person was recognized correctly. As we know, each person is tested 8 times per threshold for this test, so the maximum number in this area of the table is 8. The results shown in this table correspond to the database after being centered.

Threshold (%)	20	30	40	50	55	60	65	70	75	80	85	90	95	100
Subspace Size	1	2	3	4	5	6	8	10	13	16	21	28	40	69
Angelina Jolie	1	0	0	0	0	0	0	1	1	0	0	2	2	3
Arnold Schwarzenegger	4	1	1	1	1	1	1	1	1	1	1	0	0	0
Britney Spears	1	2	2	4	4	4	4	3	3	4	3	3	2	2
David Beckham	0	5	5	4	4	4	4	4	4	4	5	5	4	5
Leonardo DiCaprio	3	0	0	1	2	3	2	3	3	3	4	5	5	5
Michael Jackson	1	2	3	5	4	4	4	4	4	4	3	3	3	3
Michael Schumacher	3	1	1	2	3	3	3	3	3	2	2	2	2	4
Muhammad Ali	0	0	1	3	2	1	1	3	5	6	6	6	6	7
Winona Ryder	0	0	1	0	0	0	2	3	3	3	4	4	3	5
Yao Ming	2	0	0	0	0	0	2	2	2	2	2	2	3	3
Total	15	11	14	20	20	20	23	27	29	29	30	32	30	37
Success Average (%)	19	14	17	25	25	25	29	34	36	36	37	40	37	46

Table 4.5: Results of the 8–Fold Cross–Validation Test

Figure 4.8 represents the evolution of the successful recognitions as we increment the threshold percentage, showing an increase of positive recognitions as we consider more infor-

[1]please note that the database used consists of eight images of ten different people, hence 8–Fold instead of the usual 10–Fold considered in [22]

mation given by the eigenvectors to perform PCA. The numerical data is represented in table 4.6.

Threshold	20	30	40	50	55	60	65	70	75	80	85	90	95	100
Centered (%)	19	14	17	25	25	25	29	34	36	36	37	40	37	46
Normalized (%)	16	15	15	12	19	22	25	35	37	34	36	35	40	46
Raw (%)	12	12	12	12	12	12	12	12	12	12	12	12	20	46

Table 4.6: Average results for 8–Fold Cross–Validation

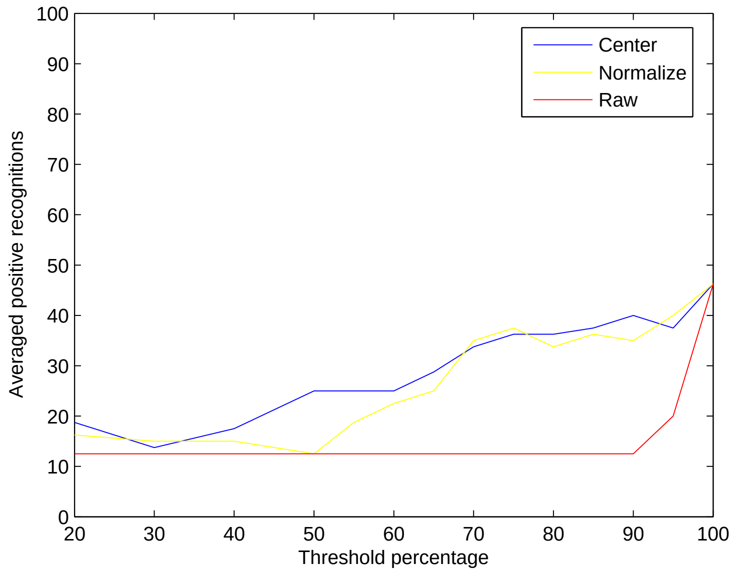


Figure 4.8: Evolution graph for 8–Fold Cross–Validation

We see that the recognition results are very similar to those of the Leave–One–Out test, which is now expected because the tested image is not included here in the training set either.

Further analysis could be made to conclude what images or persons are easier to recognize, based on the images available and their poses. For this purpose, confusion matrices can be generated for a desired threshold. These matrices show with whom each person was confused with. An example of this is table 4.7. Columns represent the person who was tested, where rows represent the predicted person. Therefore, the diagonal values of the table (bold) are the number of correct recognitions for each person, whereas the rest of the values indicate how many times the tested person was confused with the rest.

		Real Person									
		A. Jolie	A. Schwarzenegger	B. Spears	D. Beckham	L. DiCaprio	M. Jackson	M. Schumacher	M. Ali	W. Ryder	Y. Ming
Predicted Person	A. Jolie	**2**	1	0	0	0	1	0	1	1	1
	A. Schwarzenegger	0	**0**	0	0	0	0	1	0	0	0
	B. Spears	1	1	**3**	0	0	1	0	0	0	1
	D. Beckham	1	0	1	**5**	0	0	4	0	0	1
	L. DiCaprio	0	1	0	1	**5**	0	0	0	1	0
	M. Jackson	0	1	0	0	0	**3**	0	0	0	0
	M. Schumacher	2	2	0	1	1	0	**2**	1	1	3
	M. Ali	1	1	3	0	1	2	1	**6**	0	0
	W. Ryder	1	0	1	0	1	1	0	0	**4**	0
	Y. Ming	0	1	0	1	0	0	0	0	1	**2**

Table 4.7: Confusion Matrix for threshold 90%, 8–Fold Cross–Validation

4.4 Single Person Facespace (SPF)

In this section we study how subspace distance performs when comparing personal facespaces. We will span different facespaces for each person in the different sections of this chapter, in an attempt to study various situations depending on the images we have available. In this sense, at a first approach we will span facespaces with the images available. This will result in spaces with a small number of eigenvectors since we don't have big databases. We will explore a new way to use our limited database to span facespaces. We will synthesize images by shifting the existing ones. We will use this technique in two manners: firstly, we will generate a personal database by shifting just an image of that person. We will call this *Single Image Extended Database*, or SIED. Secondly, we will shift images from all the images available for each person, obtaining what we call a *Multiple Image Extended Database* or MIED.

4.4.1 Spanning facespaces with the available images

The first approach to comparing personal facespaces between the test person and all the train people considered only the images we had in the database: 8 pictures per person and 10 persons in total. Since the database used is reduced (the so-called *Small Sample Size* (SSS) problem), there was not much hope on getting many successful detections. Since there were only 8 images available per person, a selectable number of them is destined to generate the training facespaces for each person and the rest will be used to generate the test facespace.

To perform the tests, we decided to use half of the images for training and the other half for testing. This means each group had four images. The dimension of the subspaces generated for each person had a maximum of just 4 eigenvectors, since four images can only result in four or less non-zero eigenvalues, which is not very encouraging. These four eigenfaces would have a lot of information held in just four directions, it would be better to have that information more evened out in ten or twelve eigenfaces.

The number of images used to span each SPF is a parameter explicitly chosen before running the test, and those images selected for that purpose will be randomly picked out from the available for each person. This way each run uses different faces to span each SPF and we can also measure the variance in results depending on which (and not only how many) images are used in each person's facespace.

4.4.2 Expected Results

The results of the recognition attempts turned out as expected: very low rate of success in the recognitions. Different results were obtained since the test and train subspaces were generated each time with different combinations of each person's images.

Due to this random selection of the images that will construct matrix **A** for each person and which PCA is performed on, the test was repeated a large number of times. A statistical average was obtained to show average results after many random selections of images to generate de SPFs. The average results after 50 repetitions and testing for the database being centered, normalized and raw are represented in table 4.8:

Threshold	40	60	80	100
Subspace Size	1	2	3	4
Centered (%)	5.57	13.85	16.35	41.73
Normalized (%)	26.80	23.60	39.40	60.60
Raw (%)	53.20	53.20	53.20	38.00

Table 4.8: Average results in first attempt with Single Person Facespace

Figure 4.9 shows the evolution of the successful recognitions throughout the threshold percentages. It is noticeable that since there are so few eigenvectors, the more we use to construct the SPFs the better.

Despite finding a significant improvement in the results as we rise the threshold (except for the raw database) the results don't reach the expected percentages. So, the next move is to extend the database in order to have more eigenvectors which will give us more information so that we can recognize better. Since images are usually very scarce due to them being hard to obtain, we could generate images from the existing.

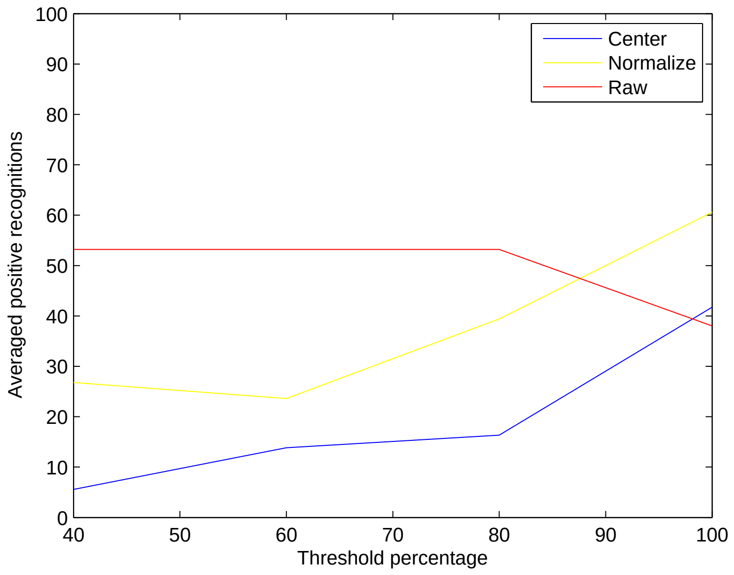
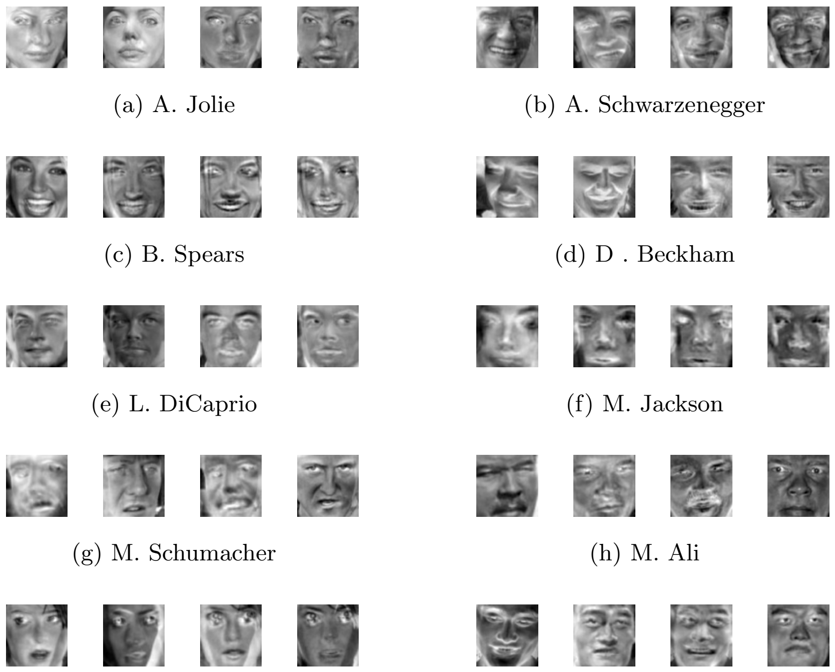


Figure 4.9: Evolution graph for Single Person Subspace averaged after 50 tests

Figure 4.10 shows the eigenfaces which compose each person's facespaces:



(a) A. Jolie

(b) A. Schwarzenegger

(c) B. Spears

(d) D . Beckham

(e) L. DiCaprio

(f) M. Jackson

(g) M. Schumacher

(h) M. Ali

(i) W. Ryder

(j) Y. Ming

Figure 4.10: SPFs' eigenfaces

4.4.3 Enlarging the Subspace Dimension: Single Image Extended Database

By creating new images from the ones we already have we possibly add information that would have been provided by real images. We are going to synthesize new images by shifting the images in the database [12]. This way, we obtain as many new images as we want with just one, being them all very similar but not equal. This will give a feature extraction very different to what we have seen. We will see how the eigenfaces show texture and face contour instead of a mix of ghostly faces like before.

Mathematics for shifting Images

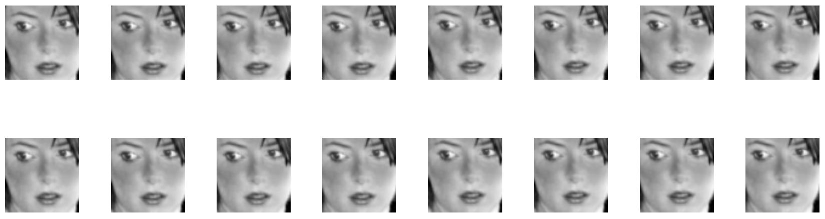
The procedure is fairly simple and it doesn't add computational or storage costs since the images generated are simply shifted versions of the original one. The images are generated from an image represented by matrix $\mathbf{\Psi}$ of dimension $M \times N$, in our case 64×64, getting mn images of dimension $l \times r$ as follows:

$$\mathbf{\Psi}_{i,j} = \mathbf{\Psi}(i : (l + i - 1), j : (r + j - 1)) \tag{4.1}$$

With $1 \le i \le m$ and $1 \le j \le n$, where m and n are parameters that choose the amount of images to synthesize (mn). $l = M - m + 1$ and $r = N - n + 1$.

The resulting database used to generate a Single Person Facespace is represented in figure 4.11. The parameters used to shift the images for this section are $m = 4$ and $n = 4$, which generate a total of 16 synthetic images. The image to be shifted is selected randomly from the available. This figure is an example of the database generate for one person using just an image of him, which we call Single Image Extended Database (SIED). The image is selected randomly out of the available (figure 4.11a) and produces the database in figure 4.11b.

(a) Original image



(b) Shifted images

Figure 4.11: Shifted synthetic images to generate a Single Image Extended Database (SIED)

This Single Image Extended Database results in the following Single Person Facespace (SPF) which composes what we call an SPF–SIED. The figure includes all 16 eigenfaces in the facespace:

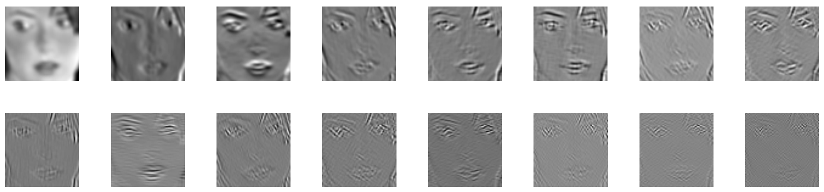


Figure 4.12: SPF–SIED eigenfaces

The thresholds considered have changed and are rather big. This is because the first eigenvector has a lot of energy (it looks pretty much like the original face), so in order to get facespaces of dimension bigger than one we have to raise the threshold quite a lot. Also, each database pre-processing method has an influence on the number of eigenvectors used for each threshold. Table 4.9 shows the different dimensions for each method and threshold.

	Threshold	85	90	91	92	93	94	95	96	97	98	99	100
Size	Center	3.1	2.8	3.8	4.0	3.0	3.9	3.2	4.4	5.2	5.4	8.5	16.0
	Normalize	2.6	2.8	2.9	2.8	3.0	3.9	3.0	2.5	3.70	4.4	7.9	16.0
	Raw	1.0	1.0	1.2	1.1	1.1	1.0	1.50	1.2	1.5	2.3	2.7	16.0

Table 4.9: Comparison of average number of eigenvectors per threshold depending on the pre-processing method for a SIED database (16 images)

It is clearly seen that the database without preprocessing has very big eigenvectors, because it has neither been centered or normalized, which therefore has a lot of energy. With this information, the successful recognition ratio for the database pre-processing methods is displayed on table 4.10 and on figure 4.13.

Threshold	85	90	91	92	93	94	95	96	97	98	99	100
Centered (%)	23	23	22	23	28	26	18	19	24	24	22	33
Normalized (%)	28	30	26	18	28	36	35	28	33	24	26	44
Raw (%)	34	32	34	38	36	32	28	37	42	28	30	42

Table 4.10: Average results for Single Person Facespace using Single Image Extended Database (SPF–SIED)

A more or less constant rate of successful recognitions is achieved here despite rising the threshold up to 100%. This means that it's almost as efficient constructing 2– or 3–dimensional SPFs than using all of the eigenvectors available. The first three or four eigenfaces contain all the relevant features.

Surprisingly, the database that has a higher percentage of success is the raw database, with an average success rate throughout the thresholds of 34.75%, while centered and normalized SIEDs have 23.88% and 29.70% respectively (more in table 4.12).

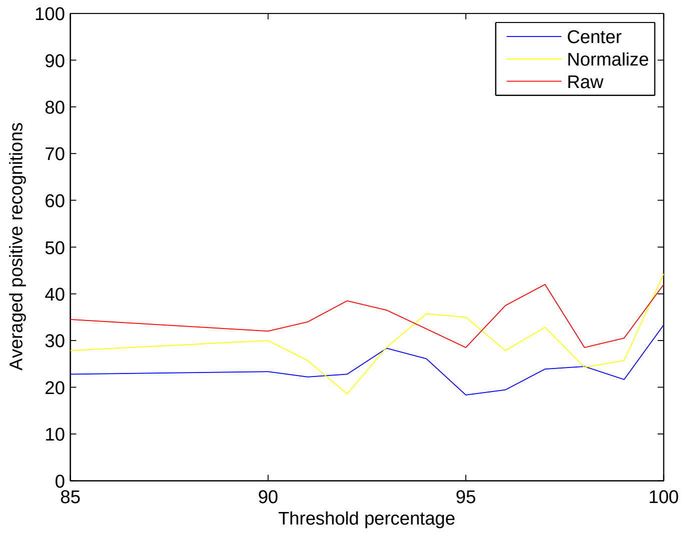


Figure 4.13: Evolution graph for SPF–SIED similarity comparisons

4.4.4 Enlarging the Subspace Dimension: Multiple Image Extended Database

Since we have eight images per person, and we use half of them to generate their facespace and the rest are left out in case we want to test that person, it's reasonable to think that it might be better to expand the database using shifted images from those four originals instead of just from one of them.

This way, each image generates four shifted images, making a total of 16 images to construct the facespace. More shifted images could be generated from each one if desired. This extends the idea of generating synthesized images from the available as in [12], but with the improvement that having a few images in the training set to synthesize and generate the personal facespace will include more information in the feature vectors.

From only four images available we have obtained sixteen, yet preserving as much information as we had with those original images. This is what we call a Multiple Image Extended Database (MIED), which we will use to create a Single Person Facespace (SPF–MIED), and its represented in figure 4.14.

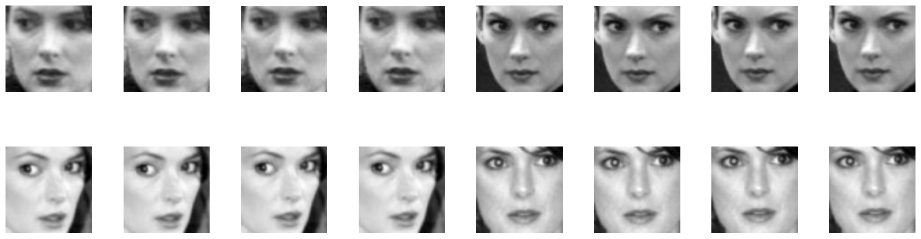


Figure 4.14: Multiple Image Extended Database (MIED)

The SPF's eigenfaces that result from this MIED database are shown in figure 4.15. Please compare them to the SPF's eigenfaces generated for the same person in the previous section using a SIED.

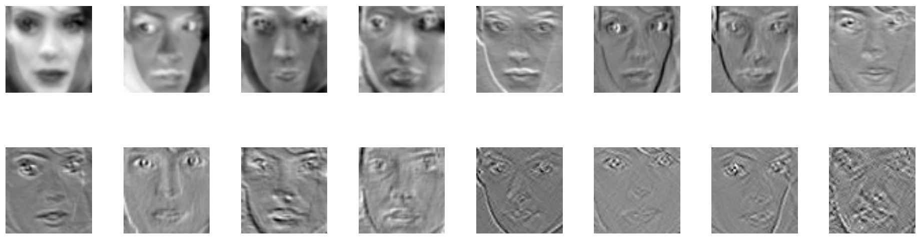


Figure 4.15: SPF–MIED's eigenfaces

This new way of synthesizing the database gives out much better results than the previous, at is is shown in table 4.11 and figure 4.16.

Threshold	85	90	91	92	93	94	95	96	97	98	99	100
Centered (%)	75	79	84	80	84	80	84	74	74	68	82	88
Normalized (%)	74	84	86	89	83	85	86	86	78	67	78	87
Raw (%)	77	72	71	73	70	68	67	58	43	47	65	83

Table 4.11: Average results for Single Person Subspace with MIED

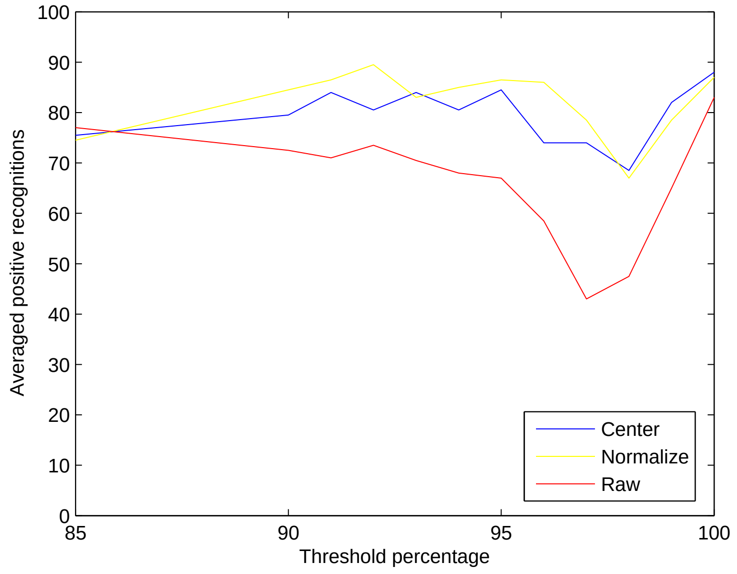
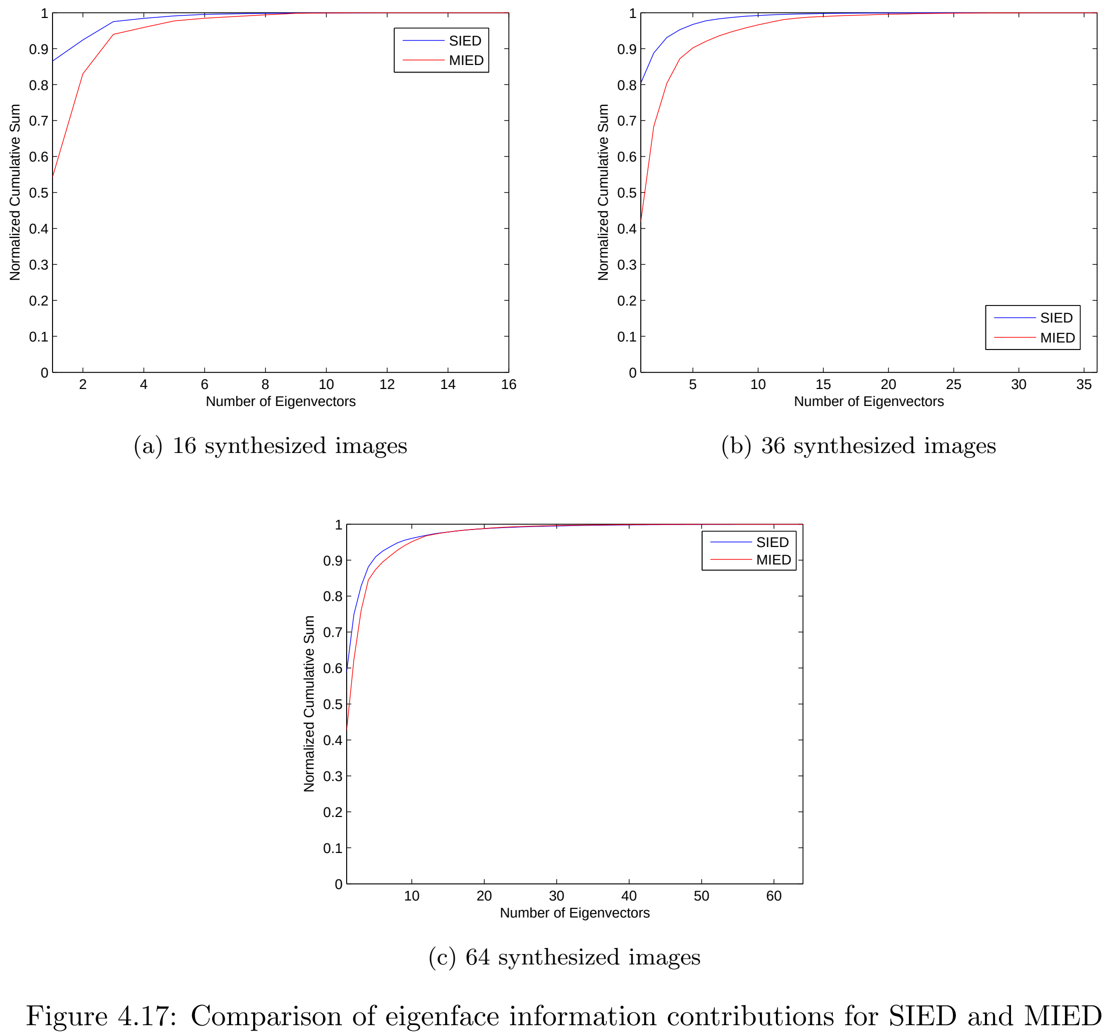


Figure 4.16: Evolution graph for Single Person Subspace with MIED – 16 images

Now we can see that a normalized database gives the best results, whereas the raw database performs decently until the threshold includes another eigenface. Then, the performance drops considerably while the centered and normalized databases (who also suffer a loss of performance) manage to hold the rates higher.

4.4.5 Comparison between SIED and MIED

It is noticeable that MIED's eigenfaces contain the information more evened out than SIED's. It can be seen that in a SPF–MIED more eigenfaces contain relevant information than in the single image extended database. This can be clearly seen in figure 4.17, where the cumulative sum of the information supplied by each eigenvector after the eigendecomposition is represented for both ways to expand the database. For an SPF–SIED of 16 synthesized images (fig. 4.11), its eigenfaces apparently stop having relevant information from the 5th or 6th onwards, while for the SPF–MIED (fig. 4.15) you could say that they stop providing good information from the 13th or 14th onwards.



(a) 16 synthesized images

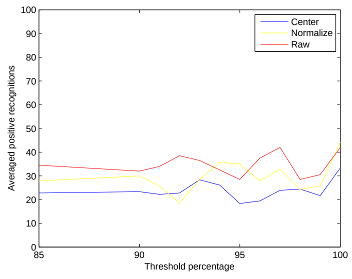
(b) 36 synthesized images

(c) 64 synthesized images

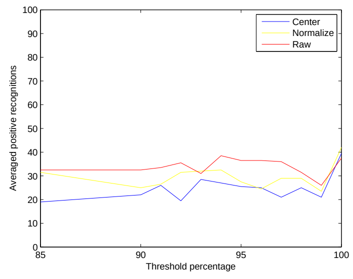
Figure 4.17: Comparison of eigenface information contributions for SIED and MIED

This can explain the better results in facial recognition for a MIED than for a SIED we saw earlier. Nevertheless, MIED and SIED have also been tested when synthesizing more images. In the following tables and graphs, model comparison results are displayed as before (including each database preprocessing method) with MIEDs and SIEDs of 16, 36 and 64 synthesized images. We want to find out what are the best circumstances for optimum SPF comparison, whether it is with a SIED or a MIED, and between those two which preprocessing method to use, and how many images to synthesize.

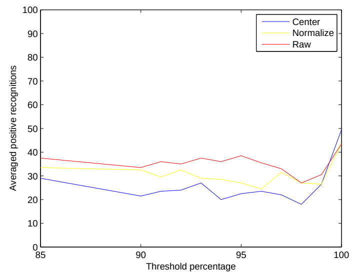
It can be seen on figure 4.17 that in all three cases, the first eigenfaces of a SIED have more energy than the first of a MIED. This difference is gradually reduced as we increment the number of synthesized images in the database. Despite this, the recognition results of the SIED don't gradually liken to those of the MIED. Figure 4.18 shows no apparent increase in the rate of recognitions when we synthesize more images. It leads to think that facespace similarity comparison doesn't depend on how many significant feature vectors are obtained. Enlarging the database with more or less synthesized images doesn't change the success rate. This implies that the information held in an image is limited, and that by generating more images out of it we are not necessarily going to obtain better results.



(a) 16 synthesized images

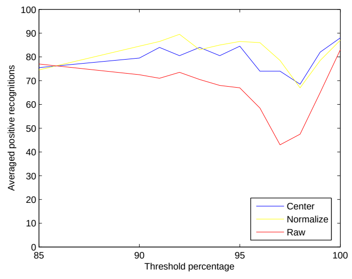


(b) 36 synthesized images

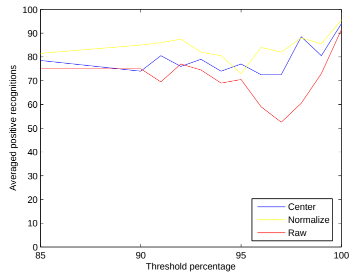


(c) 64 synthesized images

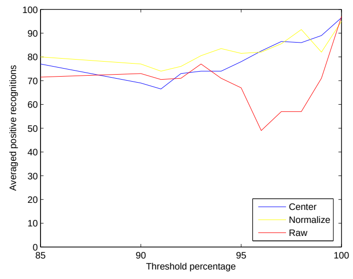
Figure 4.18: Evolution graph for Single Person Subspace with SIED



(a) 16 synthesized images



(b) 36 synthesized images



(c) 64 synthesized images

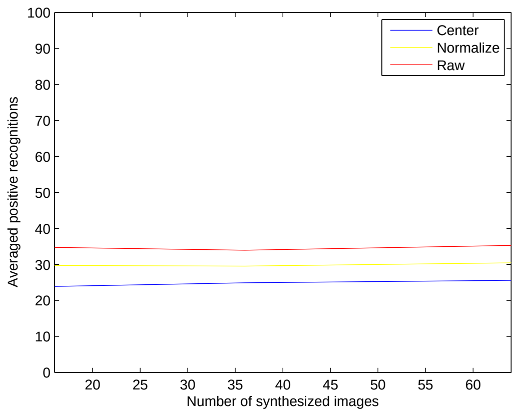
Figure 4.19: Evolution graph for Single Person Subspace with MIED

The same thing applies to the MIED database. The reason of the increased success rate compared to a SIED relies on the information held by each different picture. Indeed, expanding the database helps increase the number of eigenfaces and therefore use the first handful, which provide enough information to perform successful recognitions. On the other hand, as it happened earlier, expanding a lot the database only results in a more even distribution of the feature information. Nevertheless, the relevant information might be held in more eigenvectors when we make the database bigger, but the ratio useful–existing eigenvectors is the same. This can be proved by looking at figure 4.17, where despite the scale being different since there are more eigenfaces each time, the hump of the curve remains at the same point all the time, at around the 13–15% of the total eigenfaces.

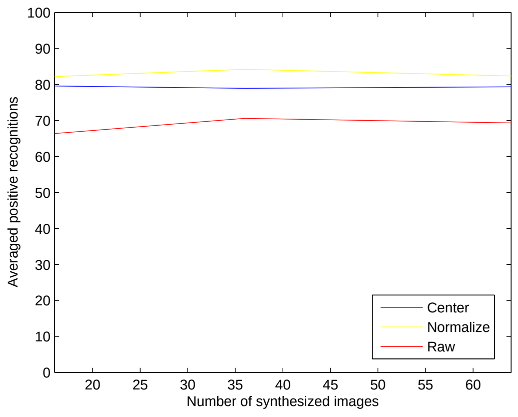
To prove that the average recognition remains intact when we increment the number of synthesized images, the average success rate for each database and preprocessing method has been calculated. The values in table 4.12 represent the average of positive recognitions in all of the thresholds. This can be graphically understood as the mean value of the curves in figures 4.18 and 4.19 along the threshold values. Then, each mean value is considered for the amount of images it has, and plotted in figure 4.20. It can be clearly seen that the average success rate remains almost constant despite generating 16 or 64 images.

	SIED			MIED		
No. of Images	Center	Normalize	Raw	Center	Normalize	Raw
16	23.88	29.70	34.75	79.58	82.20	66.37
36	24.91	29.54	33.95	78.91	84.20	70.58
64	25.58	30.45	35.29	79.33	82.37	69.33

Table 4.12: Total recognition percentages for SIED and MIED averaged throughout all the thresholds



(a) SIED



(b) MIED

Figure 4.20: Total recognition percentages for SIED and MIED averaged throughout all the thresholds

It could be said then that the optimal settings for recognition using PCA depend a lot on the data available, and on the method used. Nonetheless, if using extended databases, the best way to operate is to use a multiple image extended database with the images available, and generate a handful of images with each, no more than 15 or 20 in total, to reduce computational cost.

The following table and figure show the increment in time that takes the software to run PCA on an extended database. We have tested a range of database sizes to determine, combining this information with the recognition rates showed earlier, which database is the most efficient satisfying the compromise success rate–speed. The time comprises the whole program, not just PCA. We tested how much does the computer take to handle the matrices, PCA, eigendecompose them, etc.

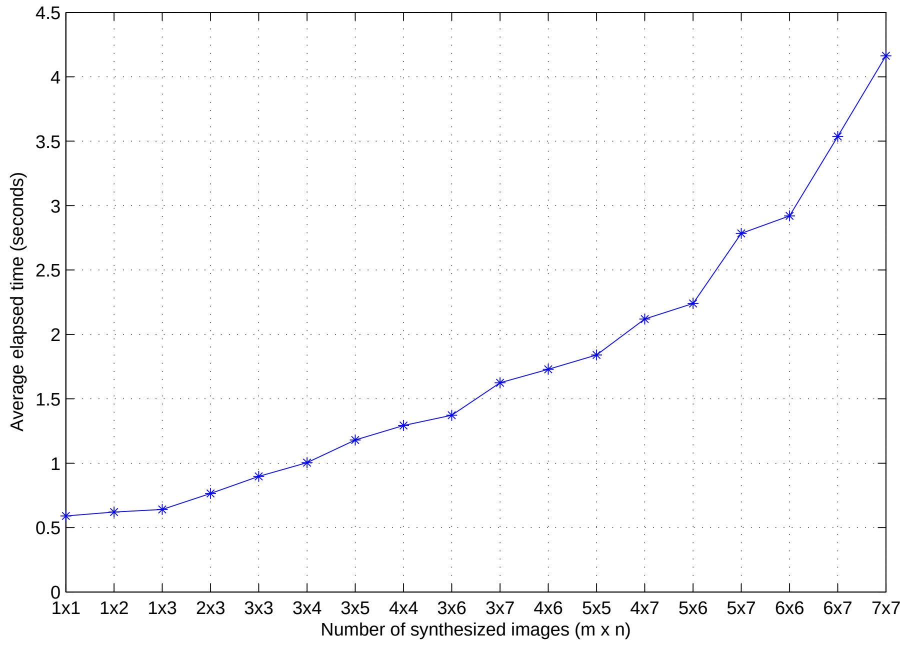


Figure 4.21: Elapsed time depending on the number synthesized images

Synthesized images $(m \times n)$	1×1	1×2	1×3	2×3	3×3	3×4	3×5	4×4	3×6
Number of images $(m \times n \times 4)$	4	8	12	24	36	48	60	64	72

Synthesized images $(m \times n)$	3×7	4×6	5×5	4×7	5×6	5×7	6×6	6×7	7×7
Number of images $(m \times n \times 4)$	84	96	100	112	120	140	144	168	196

Table 4.13: Relationship of $m \times n$ variables to number of images generated

The results have been obtained by changing the values m and n for expanding the database, using the variation to generate a MIED. This is the reason why the resulting images are 4 times bigger than the product $m \times n$, because we have performed $m \times n$ to each of the images to shift, and for a MIED we have four of them.

As the database gets larger and larger a clear increment in the time it needs to perform a loop in the code (access the database, generate the matrix $\mathbf{X}$, preprocess, expand the database, eigendecompose and span the SPFs, compare principal angles and give out a verdict) increases exponentially.

Chapter 5

Conclusions

As we have seen, there exists many different ways to implement PCA to perform face recognition. We can center, normalize and leave the database raw and the results will occasionally vary a lot. Amongst all the different options we have encountered and tested, we will try to choose the one which gives the best results possible yet having a reasonably low computational cost. This computational cost may be looked at as hard–drive space to store the images, processing power to read the code and manipulate the data as fast as possible, and also as time needed to do so.

5.1 Database

Having seen that the database used, LFWcrop Image database, is a set of images taken in *the wild*, the results obtained are in overall pretty decent. Studies and papers published often use databases which have been created under controlled circumstances. This might apply in real life too, like a criminal database where the subject is explicitly asked to pose one way or another, but sometimes it can't apply, for example when capturing images through a security camera system in a street.

The results obtained with this database are, therefore, very good and the small difference of PCA's performance found in other papers might be due to the *wildness* of the images it contains.

Aside from the kind of database used, we have seen that the preprocessing methods help improve PCA's performance. The preferred option regarding the results obtained and considering its computational benefits would be normalizing the database. As we saw earlier, a normalized database has pretty much the same performance as a centered database in all of the tests carried out. In some cases even the normalized database outperforms the centered one.

This fact, added to not needing to calculate the mean face and subtract it to all of the images (with the inconvenience that if a new image is added to the database, the centering has to be done again) and store it and use it as an input parameter for PCA, we can conclude that normalizing is the best option indeed.

5.2 Threshold

Literature shows that a common threshold value is somewhere between 80 and 95%. For a database facespace, the threshold is important since there are many images who add relevant features to the facespace. Looking at the Leave–One–Out and 8–Fold Cross–Validation result graphs we can see that in the mentioned range the recognitions are the highest. In the case of single person facespaces, the optimal threshold may vary depending on the way the database is expanded.

5.3 Projection Distance in a Facespace or SPF Comparison

When using a group database to span the facespace and project the images onto it, we have seen that the results are not too good when the tested image is not included in the training set. Looking at the image database we can see that some people have similar pictures and very different pictures of themselves. Further analysis could be made to study whether the successful recognitions are between a pair of similar images of the same person, and if the incorrect recognitions were induced by a greater similarity between two different persons' poses rather than two images of the same person.

On the other hand, when the tested image was introduced in the training set, we obtained excellent results with a consistent average of 80%. This leads us to think that projecting images onto a facespace is a good and accurate method as literature has already proven. We may blame the inconsistency of the poses in the image database to for this difference in the results. It is highly probable that if a pose-controlled image database was used, the results would have been good for the standard Leave–One–Out and 8–Fold Cross–Validation tests.

The other method used, constructing personal facespaces and comparing their principal angles, showed very weak at first but then proved it could be tuned up to obtain great recognition results.

Since the database is small, having to buid two facespaces for each person was a real problem. The number of images available for each one was even smaller than for the database facespace. For a database facespace, we had 7 training images of the subject to compare the test image with. Now, we can only build the facespace with 4 images. By software, this number could be changed and use, for instance, 5 images for the test facespaces and 3 for the training facespaces, but all along 4 images each were used since it seemed more consistent. Using 7 and 1, for example, would have put all the information in one facespace and almost none in the other, making it unfair for principal angle comparison.

Anyhow, using 4 images per facespace showed very low rates of recognition, since the facespaces spanned had only 4 eigenfaces maximum and their energy were not well evened out. By expanding the database using shifted synthetic images, the results improved considerably. The results show that it is far more efficient to extend the databases by shifting a few images from all the available (MIED) than generating that same amount of images from just one (SIED). That means that is is preferable to generate 4 synthetic images from 4 different than to generate synthetic images from a single one.

If only one or two images per subject are available, a SIED database would be generated. In this case, pretty much the same success rates would be achieved despite synthesizing 16 or 64 images, or using 80 or 100% threshold, as figure 4.18 shows. Therefore, 2– or 3–dimensional SPFs (threshold of around 90%) for a SIED database of 16 synthetic images seems the most suitable, keeping the success rates high and at the same time using reasonably small databases and eigenvector matrices.

In the case of being able to generate a MIED, the results are far more satisfactory. In the same way as for a SIED, there is no big difference between synthesizing more or less images, therefore generating three or four images from an original would be enough, in the case we have four or five originals to use.

A great way to use Single Person Facespaces is when using video to acquire images. Video is a natural and fast way to obtain many images from the subject. This would prevent the problem of small facespaces, and maybe then synthesize the images to even out the information the images provide (get more eigenvalues so that they are smaller).

As for the threshold for a MIED, in figure 4.19 we can see that when using 100% of the threshold we achieve very high success rates (over 90% successful recognition). It may be necessary to consider this threshold as optimal, specially if the software uses big databases. Using 100% threshold results in a much greater success rate than using 90% for 36 and 64 synthetic image MIEDs, showing an increase of about 20%. For a 16 synthetic image MIED, the rate is fairly the same between 90 and 100%. If the computer where the software is run on can handle bigger databases and manages to reduce process times (as the ones showed in figure 4.21, a high threshold and big extended databases may result in an extremely effective and efficient face recognizer.

Appendix A

Source Codes

A.1 StartDetection.m

```
clc
clear all
close all

%% QUESTION ASKING: SELECT OPTIONS AND TESTS ====================================

if exist('STPRtoolPath.mat') == 0
    STPRtoolPath = uigetdir('', 'Select STPR Tools stprpath.m path' );
    save STPRtoolPath.mat STPRtoolPath
else
    load STPRtoolPath.mat
end

addpath(genpath(STPRtoolPath))
stprpath
clc

if exist('TrainDatabasePath.mat') == 0  %First time asks for Train Images folder path
    TrainDatabasePath = uigetdir('', 'Select image database folder' );
    save TrainDatabasePath.mat TrainDatabasePath
else
    load TrainDatabasePath.mat
end

rrrr = [];
wantFigures = 'N';

fprintf('User mode: the program asks the user to select recognition options\n')
fprintf('Debug mode: no questions asked. The program uses the variables preset
        in the code\n')
fprintf('(to change them, please open StartDetection.m and change the values
        where "DEBUG variable" is written)\n\n')

prompt = {'User mode or Debug mode? (U/D)'};
dlg_title = '';
num_lines= 1;
def = {'U'};
debug  = inputdlg(prompt,dlg_title,num_lines,def);
debug = char(debug);
if debug == 'd'
    debug = 'D';
elseif debug == 'u'
    debug = 'U';
elseif debug ~= 'U' && debug ~= 'D'
    debug = 'U';
end

if debug == 'U'
    prompt = {'Do you want to obtain average results? (Y/N) '};
    dlg_title = '';
    num_lines= 1;
    def = {'N'};
    answer  = inputdlg(prompt,dlg_title,num_lines,def);
    if strcmp(answer,'Y') || strcmp(answer, 'y')
        prompt = {'Select number of rounds'};
        dlg_title = '';
        num_lines= 1;
        def = {'20'};
        loop  = inputdlg(prompt,dlg_title,num_lines,def);
        loop = str2double(char(loop));
    elseif strcmp(answer,'n') || strcmp(answer,'N')
        loop = 1;
    else
         errordlg('Unknown option. Please select Y or N', 'Javier makes you notice')
         error('Unknown option')
    end
else
    loop = 5; %DEBUG variable
end

promptControl = 0;

for tttt=1:loop
CorrectRecognitions = 0;
ConfusionMatrix = zeros(10,10); %Columns: real people,
                                %Rows: predicted people.
                                %Number: number of times it predicted who
LogLineOut = 0;
```

```
% NumOfPicsForSpace  = 0;
TrainSpaceStruct = 0;
testData = 0;
round = 0;
tiempo = [];

LeaveOneOutTest = 0; %1 = perform test, 0 = just do one recognition
CrossValidationTest = 0;
SinglePersonSubspace = 0;
SinglePersonSubspaceExtendedDatabase = 0;

if promptControl == 0
    if debug == 'U'
        prompt = {'Which test do you wish to perform? 1=Single Recognition,
                  2=Leave-One-Out, 3=Cross-Validation, 4=SPF, 5=SPF-SIED, 6=SPF-MIED'};
        dlg_title = 'Test';
        num_lines= 1;
        def = {'1'};
        test  = inputdlg(prompt,dlg_title,num_lines,def);
        test = str2double(char(test));
    else
        test = 6; %DEBUG variable
    end

    switch test
        case 1

        case 2
            LeaveOneOutTest = 1;
        case 3
            CrossValidationTest = 1;
        case 4
            SinglePersonSubspace = 1;
        case 5
            SinglePersonSubspaceExtendedDatabase = 1;
            version = 1;
        case 6
            SinglePersonSubspaceExtendedDatabase = 1;
            version = 2;
        otherwise
             errordlg('Unknown method. Please chose between 1 and 6',
                      'Javier makes you notice')
             error('Unknown method. Please chose between 1 and 6')
    end

    if debug == 'U'
        prompt = {'Which preprocessing method would you like? 1: Center,
                   2: Raw, 3: Normalize'};
        dlg_title = 'Choose between preprocessing methods';
        num_lines= 1;
        def = {'1'};
        Option  = inputdlg(prompt,dlg_title,num_lines,def);
        Option = str2double(char(Option));
        if Option < 0 || Option > 3
            errordlg('Unknown method. Please chose between 1 and 3',
                     'Javier makes you notice')
            error('Unknown method. Please chose between 1 and 3')
        end
    else
        Option = 1; %DEBUG variable
    end
    NumberOfPicsPerPerson = 8;

    if test == 1
        if debug == 'U'
            prompt = {'Select person (1 to 10)'};
            dlg_title = 'Please choose a person';
            NumberOfTestImage  = inputdlg(prompt,dlg_title,num_lines,def);
            NumberOfTestImage = str2double(char(NumberOfTestImage));
            prompt = {'Select threshold percentage'};
            dlg_title = 'Single Recognition';
            def = {'80'};
            threshold  = inputdlg(prompt,dlg_title,num_lines,def);
            threshold = str2double(char(threshold));
        else
            NumberOfTestImage = 3; %DEBUG variable
            threshold = 40; %DEBUG variable
        end
        TestPerformed = 'Single Detection';

    elseif LeaveOneOutTest == 1
        NumberOfTestImage = 1:80;
        if debug == 'U'
            prompt = {'Please select threshold (0 = a range of thresholds)'};
            def = {'80'};
            dlg_title = 'Leave One Out test';
            threshold  = inputdlg(prompt,dlg_title,num_lines,def);
            threshold = str2double(char(threshold));
            if threshold == 0
                threshold = [20 30 40 50:5:100];
            end
        else
            threshold = [20 30 40 50:5:100]; %DEBUG VARIABLE (single or range)
        end
        TestPerformed = 'Leave One Out Test';

    elseif CrossValidationTest == 1
        NumberOfTestImage = 1:10;
        if debug == 'U'
            prompt = {'Do you want to see a confusion matrix for a threshold (1)
                       or general results for a range of thresholds (2)?'};
            dlg_title = '8-Fold Cross Validation';
            def = {'1'};
            select  = inputdlg(prompt,dlg_title,num_lines,def);
            select = str2double(char(select));
            switch select
                case 1
                     prompt = {'Select threshold (percentage)'};
                     dlg_title = '8-Fold Cross Validation';
                     def = {'80'};
                     threshold  = inputdlg(prompt,dlg_title,num_lines,def);
                     threshold = str2double(char(threshold));
                case 2
                   threshold = [20 30 40 50:5:100];
            end
        else
            threshold = [20 30 40 50:5:100]; %DEBUG VARIABLE (single value or range)
        end
        successes = zeros(size(threshold));
        SubspaceSizeVector = zeros(size(threshold));
        TestPerformed = '8-fold Cross Validation';

    elseif SinglePersonSubspace == 1 || SinglePersonSubspaceExtendedDatabase == 1
        if debug == 'U'
            prompt = {'Please select test subject (1 to 10, 0=all of them)'};
            dlg_title = 'SPF';
            def = {'0'};
            NumberOfTestImage  = inputdlg(prompt,dlg_title,num_lines,def);
            NumberOfTestImage = str2double(char(NumberOfTestImage));

            if NumberOfTestImage == 0
                NumberOfTestImage = 1:10; %%Person to recognize
            end

            prompt = {'Please select number of images for test facespace (2 to 7)'};
            def = {'4'};
            dlg_title = 'SPF';
            NumOfPicsForSpace  = inputdlg(prompt,dlg_title,num_lines,def);
            NumOfPicsForSpace = str2double(char(NumOfPicsForSpace));

            if NumOfPicsForSpace > 7 || NumOfPicsForSpace < 2
            NumOfPicsForSpace = 4;
        end
    else
        NumOfPicsForSpace = 4;
        NumberOfTestImage = 1:10; %DEBUG VARIABLE (single or range)
    end

    if SinglePersonSubspaceExtendedDatabase == 0
        TestPerformed = 'Single Person Facespace';
        threshold = [40 60 80 100];
    else
        TestPerformed = 'Single Person Facespace with Extended Database';
        if debug == 'U'
            prompt = {'Please select threshold (0 = a range of thresholds)'};
            def = {'80'};
            dlg_title = 'SPF';
            threshold  = inputdlg(prompt,dlg_title,num_lines,def);
            threshold = str2double(char(threshold));
            if threshold == 0
                threshold = [85 90:1:100];
            end
        else
            threshold = 100; %DEBUG VARIABLE (single or range)
        end

        if debug == 'U'
            prompt = {'Please select shifting variable "m":'};
            def = {'4'};
            dlg_title = 'SPF';
            mm  = inputdlg(prompt,dlg_title,num_lines,def);
            mm = str2double(char(mm));

            prompt = {'Please select shifting variable "n":'};
            def = {'4'};
            dlg_title = 'SPF';
            nn  = inputdlg(prompt,dlg_title,num_lines,def);
            nn = str2double(char(nn));
        else
            if version == 1
                mm = 4;%8, 6 y 4 -----10 %DEBUG VARIABLES mm, nn
                nn = 4;
            elseif version == 2
                mm = 4;%4, 3 y 2 ---- 5
                nn = 4;
            end
        end
    end
```

```
    end

    prompt = {'Do you want to see the eigenfaces at the end? (Y/N)'};
    dlg_title = 'Final Results';
    def = {'N'};
    wantFigures = inputdlg(prompt,dlg_title,num_lines,def);
    wantFigures = char(wantFigures);
end

%% TESTS START =====================================================

%% Leave one out test or normal recognition------------------------------

if strcmp(TestPerformed,'Single Detection') || strcmp(TestPerformed,'Leave One Out Test')
    for r = 1:length(NumberOfTestImage)
        for k = 1:length(threshold)
            for i = 1:(NumberOfPicsPerPerson*10)
                round = round + 1;
                [TrainDatabasePath option threshold(k) DatabaseSizeUsed TestImage
                    TestImagePath] = SetUpVariables(TrainDatabasePath,
                    NumberOfTestImage(r), Option, threshold(k),
                    NumberOfPicsPerPerson, k, TestPerformed);

                faceStruct = StructureData(TrainDatabasePath,DatabaseSizeUsed,
                    LogLineOut, TestPerformed, k,  NumberOfTestImage(r));

                [A m]= CenterData(faceStruct, option);

                [Eigenfaces, V, D] = ComputeEigenfaces(A, threshold(k));
                [OutputName Recognized_index Recognized_label] =
                    Recognition(TestImagePath, m, A, Eigenfaces,
                    LogLineOut, CrossValidationTest, k, TestPerformed,
                    TrainSpaceStruct, testData, faceStruct, option);
                [CorrectRecognitions person SubspaceSize Test_label] =
                    ResultDisplay(CorrectRecognitions, TrainDatabasePath,
                    OutputName, TestImage, TestImagePath, NumberOfTestImage(r),
                    Recognized_index, faceStruct, NumberOfPicsPerPerson,
                    threshold(k), option, Eigenfaces);

                ConfusionMatrix(Recognized_label,Test_label) =
                    ConfusionMatrix(Recognized_label,Test_label) + 1;

                if strcmp(TestPerformed,'Single Detection')
                    break
                elseif strcmp(TestPerformed,'Leave One Out Test')
                    close all
                    break
                end
            end
            successes(k) = CorrectRecognitions;
            SubspaceSizeVector(k) = SubspaceSize;
            CorrectRecognitions = 0;
        end
        Results(r,:) = successes;
    end
end

%% Cross Validation Test ---------------------------------------------------
if strcmp(TestPerformed,'8-fold Cross Validation')
    for j = 1:length(NumberOfTestImage)
        for i = 1:length(threshold)
            for k = 0:(NumberOfPicsPerPerson-1)
                round = round + 1;
                [TrainDatabasePath option threshold(i) DatabaseSizeUsed
                    TestImage TestImagePath] = SetUpVariables(TrainDatabasePath,
                    NumberOfTestImage(j), Option, threshold(i), NumberOfPicsPerPerson,
                    k, TestPerformed);

                faceStruct = StructureData(TrainDatabasePath, DatabaseSizeUsed,
                     LogLineOut, TestPerformed, k);

                [A m]= CenterData(faceStruct, option);
                [Eigenfaces, V, D] = ComputeEigenfaces(A, threshold(i));

                [OutputName Recognized_index Recognized_label] =
                    Recognition(TestImagePath, m, A, Eigenfaces,
                    LogLineOut, CrossValidationTest, k, TestPerformed,
                    TrainSpaceStruct, testData, faceStruct, option);

                [CorrectRecognitions person SubspaceSize Test_label] =
                    ResultDisplay(CorrectRecognitions, TrainDatabasePath,
                     OutputName, TestImage, TestImagePath, NumberOfTestImage(j),
                     Recognized_index, faceStruct, NumberOfPicsPerPerson,
                     threshold(i), option, Eigenfaces);

                if length(threshold) > 1 || length(NumberOfTestImage) > 1
                    close all
                end

                ConfusionMatrix(Recognized_label,Test_label) =
                    ConfusionMatrix(Recognized_label,Test_label) +1;

            end
            successes(i) = CorrectRecognitions;
            SubspaceSizeVector(i) = SubspaceSize;
            CorrectRecognitions = 0;
        end
        Results(j,:) = successes;
    end
end

%% Subspace Distance Evaluation ----------------------------------------
if strcmp(TestPerformed,'Single Person Facespace') || strcmp(TestPerformed,
                        'Single Person Facespace with Extended Database')

    columns = randperm(8);
    for r = 1:length(NumberOfTestImage)
        for k = 1:length(threshold)
            round = round + 1;
            tic

            [TrainDatabasePath option threshold(k) DatabaseSizeUsed TestImage
                TestImagePath] = SetUpVariables(TrainDatabasePath,
                NumberOfTestImage(r), Option, threshold(k), NumberOfPicsPerPerson,
                k, TestPerformed);

            faceStruct = StructureData(TrainDatabasePath,DatabaseSizeUsed,
                LogLineOut, TestPerformed, k);

            [A m]= CenterData(faceStruct, option);

            testDataOriginal = A(:,find(faceStruct.y == NumberOfTestImage(r)));

            if strcmp(TestPerformed,'Single Person Facespace')

                         %Selects, from all pics of the test person,
                         %the desired test space size images randomly
                testDatabase = testDataOriginal(:,columns(1:NumOfPicsForSpace));

            elseif strcmp(TestPerformed,'Single Person Facespace with Extended Database')
               if version == 1
                         %selects one random image of the test person
                         %to generate a Single Image subspace
                   testDatabase = testDataOriginal(:,randint(1,1, [1 8]));
                   testDatabase = expandDatabase(testDatabase,mm,nn);
               elseif version == 2
                   testDatabase = [];
                   aux = randint(1,8, [1 8]);
                   for v = 1:NumOfPicsForSpace
                        %selects one random image of the test person
                        %to generate a Single Image subspace
                       testDatabaseTEMP = testDataOriginal(:,aux(v));
                       testDatabaseTEMP = expandDatabase(testDatabaseTEMP,mm,nn);
                       testDatabase = [testDatabase testDatabaseTEMP];
                   end
               end
            end

            [testData, V, D] = ComputeEigenfaces(testDatabase, threshold(k));

            TrainDataStruct.info = 'Matrixes A (centered) of each person
                    to generate each subspace';
            Acomplete = A;
            for i = 1:10
                trainDataOriginal = Acomplete(:,find(faceStruct.y == i));
                if strcmp(TestPerformed,'Single Person Facespace')
                    trainDatabase =
                        trainDataOriginal(:,columns((NumOfPicsForSpace+1):end));
                elseif strcmp(TestPerformed,
                    'Single Person Facespace with Extended Database')
                    if version == 1
                        trainDatabaseTEMP = trainDataOriginal(:,randint(1,1, [1 8]));
                        trainDatabase = expandDatabase(trainDatabaseTEMP,mm,nn);
                    elseif version == 2
                    trainDatabase = [];
                       for v = NumOfPicsForSpace+1:8
                                %selects one random image of the test person
                                %to generate a Single Image subspace
                           trainDatabaseTEMP = trainDataOriginal(:,aux(v));
                           trainDatabaseTEMP = expandDatabase(trainDatabaseTEMP,mm,nn);
                           trainDatabase = [trainDatabase trainDatabaseTEMP];
                       end
                   end
                end
                temp = ['ImagesPerson' num2str(i)];
                eval( ['TrainDataStruct.' sprintf(temp) '=trainDatabase;'] );

                eval(['A=TrainDataStruct.' sprintf(temp) ';'])
                [Eigenfaces, V, D] = ComputeEigenfaces(A, threshold(k));
                eval( ['TrainSpaceStruct.' sprintf(temp) '=Eigenfaces;'] );
            end

            [OutputName Recognized_index Recognized_label TrainSpaceStruct
                testData] = Recognition(TestImagePath, m, A, Eigenfaces,
                LogLineOut, CrossValidationTest, k, TestPerformed,
                TrainSpaceStruct, testData, 0, option);

            [CorrectRecognitions person SubspaceSize Test_label] =
                ResultDisplay(CorrectRecognitions, TrainDatabasePath,
                OutputName, TestImage, TestImagePath, NumberOfTestImage(r),
                Recognized_index, faceStruct, NumberOfPicsPerPerson,
                threshold(k), option, Eigenfaces);

            if length(threshold) > 1 || length(NumberOfTestImage) > 1
                close all
            end

            ConfusionMatrix(Recognized_label,Test_label) =
                ConfusionMatrix(Recognized_label,Test_label) + 1;

            successes(k) = CorrectRecognitions;
            SubspaceSizeVector(k) = SubspaceSize;
            CorrectRecognitions = 0;
            tiempo = [tiempo toc];
        end
        Results(r,:) = successes;
    end
end
round = round/length(threshold);
percentages = sum(Results)/round*100;

%% Result Printing
if test ~= 1
    clc
end
fprintf(['Test Performed: ' TestPerformed '\n'])
if test == 5
    fprintf('Single Image (SIED)\n')
elseif test == 6
    fprintf('Multiple Image (MIED)\n')
end

if test == 5 || test == 6
    fprintf(['Shifting variables: m = ' num2str(mm) ', n = '
         num2str(nn) '\n'])
    fprintf(['Number of images selected for test images prior
         to shifting:  ' num2str(NumOfPicsForSpace) '\n'])
    fprintf(['Number of images selected for train images prior
         to shifting: ' num2str(8-NumOfPicsForSpace) '\n'])
    fprintf(['Number of shifted images per person in
         test SPFs:  ' num2str(size(testDatabase,2)) '\n'])
    fprintf(['Number of shifted images per person in
         train SPFs: ' num2str(size(A,2)) '\n\n'])
end
```

```
if length(NumberOfTestImage) == 1
    fprintf(['Correct Recognitions of ' person '\n'])
    fprintf('Threshold    ')
    disp(threshold)
    fprintf('Subspace Size')
    disp(SubspaceSizeVector)
    fprintf('Successful   ')
    disp (successes)
elseif length(NumberOfTestImage) > 1
    fprintf('Threshold \n ')
    disp(threshold)
    fprintf('Subspace Size  \n')
    disp(SubspaceSizeVector)
    fprintf('Successful  \n')
    disp (Results)
    fprintf('Total per threshold \n')
    disp(sum(Results))
    fprintf('Percentage per threshold \n')
    disp(percentages)
end
if sum(sum(ConfusionMatrix)) ~= 0
    fprintf(['Confussion Matrix for threshold ' num2str(threshold) '\n'])
    disp(ConfusionMatrix)
end
if length(threshold) > 1
    plot(threshold,percentages)
    axis([threshold(1) threshold(end) 0 100])
    xlabel('Threshold percentage')
    ylabel('Averaged positive recognitions')
end

rrrr = [rrrr; percentages];
if loop > 1
    promptControl = 1;
end
end
 close all
 if loop > 1
    clc
    fprintf(['Test Performed: ' TestPerformed '\n'])
    if test == 5
        fprintf('Single Image (SIED)\n')
    elseif test == 6
        fprintf('Multiple Image (MIED)\n')
    end
    if test == 5 || test == 6
        fprintf(['Shifting variables: m = ' num2str(mm) ', n = '
            num2str(nn) '\n'])
        fprintf(['Number of images selected for test images
            prior to shifting:  ' num2str(NumOfPicsForSpace) '\n'])
        fprintf(['Number of images selected for train images
            prior to shifting: ' num2str(8-NumOfPicsForSpace) '\n'])
        fprintf(['Number of shifted images per person in
            test SPFs:  ' num2str(size(testDatabase,2)) '\n'])
        fprintf(['Number of shifted images per person in
            train SPFs: ' num2str(size(A,2)) '\n\n'])
    end
    average = sum(rrrr)/size(rrrr,1);
    fprintf(['Average results after ' num2str(size(rrrr,1)) ' repetitions\n'])
    fprintf('Threshold \n ')
    disp(threshold)
    fprintf('Subspace Size\n')
    disp(SubspaceSizeVector)
    fprintf('results (percentage)\n')
    disp(average)
    if length(threshold) >1
        plot(threshold,average)
        axis([threshold(1) threshold(end) 0 100])
        xlabel('Threshold percentage')
        ylabel('Averaged positive recognitions')
    end
 end
 if wantFigures == 'Y' || wantFigures == 'y'
    ShowFigures
 end
```

A.2 SetUpVariables.m

```
function [TrainDatabasePath option threshold DatabaseSizeUsed TestImage
    TestImagePath] = SetUpVariables(TrainDatabasePath, NumberOfTestImage,
    Option, threshold, NumberOfPicsPerPerson, k, TestPerformed)

% script that makes sure the variables entered are correct,
% and sets up initial data to feed other functions
% See also  StartDetection, GenerateDatabase, CenterData,
% ComputeEigenfaces, Recognition, ResultDisplay, ShowFigures

switch Option
    case 1
        option = 'center';
    case 2
        option = 'dontcenter';
    case 3
        option = 'normalize';
end

if threshold > 100
    threshold = 100;
    fprintf('Threshold Percentage is too large, 100 will be used\n')
end
if threshold <=0
    threshold = 50;
    fprintf('Threshold Percentage can´t be negative: 50 will be used\n')
end

fid = fopen('ImagesLogFile.txt');
i = 0;
if strcmp(TestPerformed,'8-fold Cross Validation')
            % jumps from blocks of 10 pics for test
            %depending on k (k=0 images _0001;k=1 images _0002...)
    while i ~= (NumberOfTestImage+10*k)
        TestImage = fgetl(fid);
        tabs = find(TestImage==' ');
        TestImage = TestImage(1:(tabs-1));
        i = i + 1;
    end
else
    while i ~= NumberOfTestImage
        TestImage = fgetl(fid);
        tabs = find(TestImage==' ');
        TestImage = TestImage(1:(tabs-1));
        i = i + 1;
    end
```

```
end
fclose(fid);


TestImagePath = strcat(TrainDatabasePath,'\',TestImage);


fid = fopen('ImagesLogFile.txt');
LogFileSize = 0;
while 1
    tline = fgetl(fid);
    if ~ischar(tline), break, end
    LogFileSize = LogFileSize +1;
end
fclose(fid);

DatabaseSizeWanted = 10*NumberOfPicsPerPerson;

if NumberOfPicsPerPerson ~= -1
    if DatabaseSizeWanted > LogFileSize
        num = num2str(LogFileSize/10);
        fprintf(['Selected number is bigger than database; ', num,'
            images per person will be used\n']);
        DatabaseSizeUsed = LogFileSize;
    else
        DatabaseSizeUsed = DatabaseSizeWanted;
    end
else
    DatabaseSizeUsed = LogFileSize;
end
```

A.3 StructureData.m

```
function faceStruct = StructureData(TrainDatabasePath, DatabaseSizeUsed,
                        LogLineOut, TestPerformed, k, NumberOfTestImage)

X = [];
y = [];

 fid = fopen('ImagesLogFile.txt');
 str = fgetl(fid);

if strcmp(TestPerformed,'Single Detection') || strcmp(TestPerformed,'Leave One Out Test')
    for i = 1 : DatabaseSizeUsed
        if i ~= LogLineOut
            if i ~= NumberOfTestImage
                tabs = find(str==' ');
                str1 = strcat(TrainDatabasePath,'\',str(1:(tabs-1)));
                img = imread(str1);
                [irow icol] = size(img);
                temp = reshape(img,irow*icol,1);
                X = [X temp];
                str2 = str((tabs+1):end);
                y = [y str2num(str2)];
            end
        end
        str = fgetl(fid);
    end
elseif strcmp(TestPerformed,'8-fold Cross Validation')
    for i = 1 : DatabaseSizeUsed
        if i < (10*k+1) || i > (10*k+10)
            tabs = find(str==' ');
            str1 = strcat(TrainDatabasePath,'\',str(1:(tabs-1)));
            img = imread(str1);
            [irow icol] = size(img);
            temp = reshape(img,irow*icol,1);
            X = [X temp];
            str2 = str((tabs+1):end);
            y = [y str2num(str2)];
        end
        str = fgetl(fid);
    end
else
    for i = 1 : DatabaseSizeUsed
        if i ~= LogLineOut
            tabs = find(str==' ');
            str1 = strcat(TrainDatabasePath,'\',str(1:(tabs-1)));
            img = imread(str1);
            [irow icol] = size(img);
            temp = reshape(img,irow*icol,1);
            X = [X temp];
            str2 = str((tabs+1):end);
            y = [y str2num(str2)];
        end
        str = fgetl(fid);
    end
end

faceStruct.X=X;
faceStruct.y=y;
faceStruct.inf='X = matrix with each image in a column, y = labels of each image';

fclose(fid);
```

A.4 CenterData.m

```
function [ A m ] = CenterData(faceStruct, option)

switch option

    case 'center',
        m = mean(faceStruct.X,2);      % Computing the average face image m
        Train_Number = size(faceStruct.X,2);
                                       % Calculating the deviation of each
                                       %image from mean image
        j = ones(1,size(faceStruct.X,2));
        A = double(faceStruct.X) - m*j;

    case 'normalize',
        A = double(faceStruct.X);
        A = A - 128;
        A = A./128;
        m = 0;

    case 'dontcenter',
        A = double(faceStruct.X);
        m = 0;

end
```

A.5 expandData.m

```
function DataExtended = expandDatabase(Data,mm,nn)

% Function developed by Javier de Alfonso
% Shifts the images placed as column vectors in "Data"
%to result in size(Data,2) x (mm x nn) shifted images

DataImage = reshape(Data, 64,64);
r = 1;
ll = size(DataImage,1) - mm + 1;
rr = size(DataImage,2) - nn + 1;
DataExtended = zeros(ll*rr, (size(Data,2)*mm*nn));
for i = 1:mm
    for j = 1:nn
        tempShiftImage = DataImage(i:(ll+i-1),j:(rr+j-1));
        [irow icol] = size(tempShiftImage);
        tempShiftImage = reshape(tempShiftImage,irow*icol,1);
        for ii = 1:length(tempShiftImage)
            DataExtended(ii,r) = tempShiftImage(ii);
        end
        r = r + 1;
    end
end
```

A.6 ComputeEigenfaces.m

```
function  [Eigenfaces, V, D] = ComputeEigenfaces(A, threshold)

L = A'*A; % L is the surrogate of covariance matrix C=A*A'.
[V D] = eig(L);
[valD indxD] = sort(diag(D),'descend');  %eigenvalues from bigger to smaller
V = V(:,indxD);


%% Sorting and eliminating eigenvalues --------------------------------

            %selects the number of eigenvalues needed
infosupplied = cumsum(valD)./sum(valD);

            %to meet the threshold selected
numofEigenVals = find(infosupplied >= threshold/100,1);

%We generate the new matrices after eliminating non important eigenvalues.
L_eig_vec = V(:,1:numofEigenVals);
D_threshold = valD(1:numofEigenVals);

D_threshold = D_threshold.^(-0.5);
D_threshold = diag(D_threshold);

            % A: centered image vectors Eigenfaces = A*V*(D.^-0.5 )
Eigenfaces = A * L_eig_vec  * D_threshold;
```

A.7 Recognition.m

```
function [ OutputName Recognized_index Recognized_label TrainSpaceStruct
    testData] = Recognition(TestImagePath, m, A, Eigenfaces, LogLineOut,
    CrossValidationTest, k, TestPerformed, TrainSpaceStruct, testData,
    faceStruct, option)


if strcmp(TestPerformed,'Single Person Facespace') == 0
    && strcmp(TestPerformed,'Single Person Facespace with
    Extended Database') == 0

    TrainSpaceStruct = 0;
    testData = 0;

        % Projection of centered images into facespace
    ProjectedImages = Eigenfaces'*A;

    % Extracting the PCA features from test image ---------------------
    InputImage = imread(TestImagePath);
    temp = InputImage(:,:,1); % In case pictures are not grayscale

    [irow icol] = size(temp);
    InImage = reshape(temp,irow*icol,1);

    if strcmp(option,'center')
        Difference = double(InImage)-m; % Centered test image
    elseif strcmp(option,'normalize')
        Difference = (double(InImage)-128)./128;
    else
        Difference = double(InImage);
    end

    ProjectedTestImage = Eigenfaces'*Difference; % Test image feature vector


     test.X = ProjectedImages;
     test.y = faceStruct.y;

% Uncomment to plot 2-D graph (facespace must be 2-dimensional)
%      ppatterns(test)
%      hold on
%      plot(ProjectedTestImage(1),ProjectedTestImage(2),'ks')
```

```
    %Calculating Euclidean distances ----------------------------------

    knn = knnrule(test);                            %use of STPR tools
    Recognized_index = knnclass(ProjectedTestImage, knn);

    if Recognized_index > LogLineOut && LogLineOut ~= 0
        Recognized_index = Recognized_index + 1;
    elseif CrossValidationTest == 1
                %if Recognized_index corresponds to one of the ten images
                %we took ot from train to use for test (the images to the
                %right shift their position 10 numbers to the left)
        if Recognized_index > (10*k+1) && Recognized_index < (10*k+10)
             Recognized_index = Recognized_index + 10;
        end
    end

elseif strcmp(TestPerformed,'Single Person Facespace')
        || strcmp(TestPerformed,'Single Person Facespace with Extended Database')
    results = [];

    P = testData;
    k1 = size(P,2);
    for i = 1:10
        temp = ['ImagesPerson' num2str(i)];
        eval(['Q=TrainSpaceStruct.' sprintf(temp) ';'])
        k2 = size(Q,2);
        X = P'*Q;
        distance  = sqrt(k1 + k2 - 2*norm(X,'fro').^2);
        results(i) = distance;
    end
     [subspace_distance_min , Recognized_index] = min(results);
end

fid = fopen('ImagesLogFile.txt');
i = 0;
while i ~= Recognized_index
    Recognized_line = fgetl(fid);
    tabs = find(Recognized_line==' ');
    Recognized_name = Recognized_line(1:(tabs-1));
    Recognized_label = str2double(Recognized_line(tabs+1:end));
    i = i + 1;
end
fclose(fid);
OutputName = Recognized_name;
```

A.8 ResultDisplay.m

```
function [CorrectRecognitions person SubspaceSize Test_label] =
    ResultDisplay(CorrectRecognitions, TrainDatabasePath, OutputName,
    TestImage, TestImagePath, NumberOfTestImage, Recognized_index,
    faceStruct, NumberOfPicsPerPerson, threshold, option, Eigenfaces)


SelectedImage = strcat(TrainDatabasePath,'\',OutputName);
SelectedImage = imread(SelectedImage);
im = imread(TestImagePath);

fid = fopen('ImagesLogFile.txt');
i = 0;

while i ~= NumberOfTestImage
    person_line = fgetl(fid);
    i = i + 1;
end
fclose(fid);
tabs = find(person_line=='_');
tabs2 = find(person_line==' ');
person = [person_line(1:(tabs(1)-1)), ' ',
         person_line((tabs(1)+1):(tabs(2)-1))];
Test_label = str2double(person_line((tabs2+1):end));

tabs = find(OutputName=='_');
personIdentif = [OutputName(1:(tabs(1)-1)), ' ',
                 OutputName((tabs(1)+1):(tabs(2)-1))];

figure('Name','Recognition Result','NumberTitle','off')
subplot(1,2,1)
imshow(im)
title(['Selected Subject: ' person]);
subplot(1,2,2)
imshow(SelectedImage);
title(['Recognized Subject: ' personIdentif]);

if strcmp(person,personIdentif)
    fprintf('RECOGNITION SUCCESSFUL!!!\n')
    CorrectRecognitions = CorrectRecognitions + 1;
else
    fprintf('Recognition failed!\n')
end

fprintf('Selected person: ')
disp(person)
```

```
fprintf('Identified person: ')
disp(personIdentif)

fprintf('\nRecognition Parameters:\n')
SubspaceSize = size(Eigenfaces,2);
fprintf(['Threshold Percentage: ', num2str(threshold), '\n'])
fprintf(['Processing Criteria: ', option '\n'])
fprintf(['Eigenface Subspace Dimension: ' num2str(SubspaceSize) '\n\n'])
```

A.9 ShowFigures.m

```
close all

if strcmp(TestPerformed,'Single Person Facespace') == 0
    && strcmp(TestPerformed,'Single Person Facespace with Extended Database') == 0

    figure('Name','Database Used','NumberTitle','off')
    showim(faceStruct.X,[64,64])

    figure('Name','Eigenfaces','NumberTitle','off')
    showim(Eigenfaces)%[64,64])%,[3,10])

    figure('Name','Matrix A: Preprocessed database','NumberTitle','off')
    showim(A)%,[64,64],[8,10])

    if m ~= 0
        figure('Name','Average face "m"','NumberTitle','off')
        showim(m)
    end
elseif strcmp(TestPerformed,'Single Person Facespace')
    || strcmp(TestPerformed,'Single Person Facespace with Extended Database')

    figure('Name','Images for test person','NumberTitle','off')
    showim(testDataOriginal)

    figure('Name','Extended database for Single Person Subspace','NumberTitle','off')
    showim(testDatabase)

    figure('Name','Eigenvectors of Test Subspace','NumberTitle','off')
    showim(testData)

end
```

A.10 ImagesLogFile.txt

```
Angelina_Jolie_0001.pgm 1
Arnold_Schwarzenegger_0001.pgm 2
Britney_Spears_0001.pgm 3
David_Beckham_0001.pgm 4
Leonardo_DiCaprio_0001.pgm 5
Michael_Jackson_0001.pgm 6
Michael_Schumacher_0001.pgm 7
Muhammad_Ali_0001.pgm 8
Winona_Ryder_0001.pgm 9
Yao_Ming_0001.pgm 10

Angelina_Jolie_0002.pgm 1
Arnold_Schwarzenegger_0002.pgm 2
Britney_Spears_0002.pgm 3
David_Beckham_0002.pgm 4
Leonardo_DiCaprio_0002.pgm 5
Michael_Jackson_0002.pgm 6
Michael_Schumacher_0002.pgm 7
Muhammad_Ali_0002.pgm 8
Winona_Ryder_0002.pgm 9
Yao_Ming_0002.pgm 10

Angelina_Jolie_0003.pgm 1
Arnold_Schwarzenegger_0003.pgm 2
Britney_Spears_0003.pgm 3
David_Beckham_0003.pgm 4
Leonardo_DiCaprio_0003.pgm 5
Michael_Jackson_0003.pgm 6
Michael_Schumacher_0003.pgm 7
Muhammad_Ali_0003.pgm 8
Winona_Ryder_0003.pgm 9
Yao_Ming_0003.pgm 10

Angelina_Jolie_0004.pgm 1
Arnold_Schwarzenegger_0004.pgm 2
Britney_Spears_0004.pgm 3
David_Beckham_0004.pgm 4
Leonardo_DiCaprio_0004.pgm 5
Michael_Jackson_0004.pgm 6
Michael_Schumacher_0004.pgm 7
Muhammad_Ali_0004.pgm 8
Winona_Ryder_0004.pgm 9
Yao_Ming_0004.pgm 10

Angelina_Jolie_0005.pgm 1
Arnold_Schwarzenegger_0005.pgm 2
Britney_Spears_0005.pgm 3
David_Beckham_0005.pgm 4
Leonardo_DiCaprio_0005.pgm 5
Michael_Jackson_0005.pgm 6
Michael_Schumacher_0005.pgm 7
Muhammad_Ali_0005.pgm 8
Winona_Ryder_0005.pgm 9
Yao_Ming_0005.pgm 10

Angelina_Jolie_0006.pgm 1
Arnold_Schwarzenegger_0006.pgm 2
Britney_Spears_0006.pgm 3
David_Beckham_0006.pgm 4
Leonardo_DiCaprio_0006.pgm 5
Michael_Jackson_0006.pgm 6
Michael_Schumacher_0006.pgm 7
Muhammad_Ali_0006.pgm 8
Winona_Ryder_0006.pgm 9
Yao_Ming_0006.pgm 10

Angelina_Jolie_0007.pgm 1
Arnold_Schwarzenegger_0007.pgm 2
Britney_Spears_0007.pgm 3
David_Beckham_0007.pgm 4
Leonardo_DiCaprio_0007.pgm 5
Michael_Jackson_0007.pgm 6
Michael_Schumacher_0007.pgm 7
Muhammad_Ali_0007.pgm 8
Winona_Ryder_0007.pgm 9
Yao_Ming_0007.pgm 10

Angelina_Jolie_0008.pgm 1
Arnold_Schwarzenegger_0008.pgm 2
Britney_Spears_0008.pgm 3
David_Beckham_0008.pgm 4
Leonardo_DiCaprio_0008.pgm 5
Michael_Jackson_0008.pgm 6
Michael_Schumacher_0008.pgm 7
Muhammad_Ali_0008.pgm 8
Winona_Ryder_0008.pgm 9
Yao_Ming_0008.pgm 10
```

Bibliography

[1] Loquendo. `www.loquendo.com`.

[2] Drawing a blank: Tampa Police records reveal poor performance of facial-recognition technology, 03 January 2002.

[3] J Ross Beveridge, Bruce A. Draper, Jen-Mei Chang, Michael Kirby, Holger Kley, and Chris Patterson. Principal Angles Separate Subjects Illumination Spaces in YDB and CMU-PIE. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 31(Issue 2):351–356, February 2009.

[4] Kevin Bonsor and Ryan Johnson. How facial recognition systems work. `www.HowStuffWorks.com`.

[5] Tat-Jun Chin, James U, Konrad Schindler, and David Suter. Face Recognition from Video by Matching Image Sets. *IEEE Proceedings of the Digital Imaging Computing: Techniques and Applications*, 2005.

[6] T Cover and P Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, Volume 13(Issue 1):Pages 21 – 27, January 1967.

[7] Vojtech Franc. Statistical pattern recognition toolbox, February 2000.

[8] Aurél Galántai. *Projections and Projection Methods*. 2004.

[9] Mislav Grgic and Kresimir Delac. Face recognition homepage. `http://www.face-rec.org/algorithms/`.

[10] J. Huang, B. Heisele, and V. Blanz. Component-based Face Recognition with 3D Morphable Models, 09-11 June 2003.

[11] Kyungnam Kim. Face Recognition using Principle Component Analysis. Technical report, University of Maryland.

[12] Jun Liu, Songcan Chen, Zhi-Hua Zhou, and Xiaoyang Tan. Single image subspace for face recognition.

[13] Daniel Monzón. Celda 211, 2009.

[14] Amir Hossein Omidvarnia. PCA-based Face Recognition System. `http://www.mathworks.com/`, October 2007.

[15] Norbert Rosing. Polar bear. Nature's Best Photography Windland Smith Rice International Awards.

[16] SecureIDNews. UK's Stansted Airport deploys biometric e-passport gates, 26 January 2010.

[17] Kameel Stanley. Facial recognition technology proving effective for Pinellas deputies. *St. Petersburg Times*, 21 July 2009.

[18] Xichen Sun, Liwei Wang, and Jufu Feng. Further results on the Subspace Distance. *Pattern Recognition*, 40:328–329, 2007.

[19] Matthew Turk and Alex Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neurosicence*, Volume 3(No.1):pages 71–86, 1991.

[20] Matthew Turk and Alex Pentland. Face Recognition Using Eigenfaces. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*, pages 586–591, Maui, Hawaii, USA, 3-6 June 1991.

[21] L. Wiskott, J.-M. Fellous, N. Krueuger, and C. von der Malsburg. Face Recognition by Elastic Bunch Graph Matching, Chapter 11 in Intelligent Biometric Techniques in Fingerprint and Face Recognition, 1999.

[22] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier Science Ltd, second edition, June 2005.