

Summer 2003

Face recognition using principal component analysis

Timothy Kevin Larkin
New Jersey Institute of Technology

Follow this and additional works at: <https://digitalcommons.njit.edu/theses>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Larkin, Timothy Kevin, "Face recognition using principal component analysis" (2003). *Theses*. 652.
<https://digitalcommons.njit.edu/theses/652>

This Thesis is brought to you for free and open access by the Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Theses by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

FACE RECOGNITION USING KERNEL PRINCIPAL COMPONENT ANALYSIS

**By
Timothy Kevin Larkin**

Current methods of face recognition use linear methods to extract features. This causes potentially valuable nonlinear features to be lost. Using a kernel to extract nonlinear features should lead to better feature extraction and, therefore, lower error rates. Kernel Principal Component Analysis (KPCA) will be used as the method for nonlinear feature extraction. KPCA will be compared with well known linear methods such as correlation, Eigenfaces, and Fisherfaces.

**FACE RECOGNITION USING KERNEL
PRINCIPAL COMPONENT ANALYSIS**

**by
Timothy Kevin Larkin**

**A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science**

Department of Computer Science

August 2003

Blank Page

APPROVAL PAGE

**FACE RECOGNITION USING KERNEL
PRINCIPAL COMPONENT ANALYSIS**

Timothy Kevin Larkin

Dr. Chengjun Liu, Thesis Advisor
Assistant Professor of Computer Science, NJIT

Date

Dr. Joseph Leung, Committee Member
Distinguished Professor of Computer Science, NJIT

Date

Dr. Marvin K. Nakayama, Committee Member
Associate Professor of Computer Science, NJIT

Date

BIOGRAPHICAL SKETCH

Author: Timothy Kevin Larkin
Degree: Master of Science in Computer Science
Date: August 2003

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
New Jersey Institute of Technology, Newark, NJ, 2003
- Bachelor of Science in Computer Science,
New Jersey Institute of Technology, 2000

Major: Computer Science

This work is dedicated to
my family

ACKNOWLEDGMENT

I would like to thank Dr. Chengjun Liu for being my advisor throughout this project. I would also like to thank the committee members, Dr. Joseph Leung and Dr. Marvin K. Nakayama. Finally, I would like to thank my family and friends for their constant support.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	1
1.1 Objective	4
1.2 Background Information	5
2 CURRENT METHODS	10
2.1 Correlation	11
2.2 Eigenfaces	14
2.3 Fisherfaces	24
3 KERNEL PCA	33
3.1 Description	33
3.2 Results	35
4 SUMMARY	41
4.1 Future Work	43
APPENDIX A CODE	44
Eigenfaces_load.m	45
Eigenvectors.m	46
Find_nearest.m	48
Fisher_load.m	49
Kernel.m	51
Loo.m	52

TABLE OF CONTENTS

(Continued)

Chapter	Page
Run_cor.m	53
Run_eigenfaces.m	54
Run_fisher.m	55
Run_kpca.m	56
Split_string.m	59
REFERENCES	60

LIST OF TABLES

Table	Page
1.1 Results of the Eigenfaces Method Using Different Sized Images	9
3.1 Results of Random Testing on the Yale Database Using KPCA	38
4.1 Results of Random Testing on the AT&T Database	41
4.2 Results of Leave-One-Out Testing on the AT&T Database	41
4.3 Results of Random Testing on the Yale Database	42
4.4 Results of Leave-One-Out Testing on the Yale Database	42

LIST OF FIGURES

Figure	Page
1.1 Examples of lighting changes	2
1.2 Variations due to facial expression	2
1.3 Variations due to glasses	3
1.4 Examples of faces at different angles	3
1.5 Snapshot of the 400 images from the AT&T face database	6
1.6 Examples from the Yale face database	8
2.1 Example of a test image (left) being incorrectly matched to a subject similar in appearance	11
2.2 Example of a test image (left) being matched to similar features, in this case, glasses, beard, and hair	11
2.3 Example of incorrect classification due to glasses	12
2.4 Example of incorrect classification where very little similarity is obvious	12
2.5 Example of incorrect classification due to lighting	13
2.6 Example of incorrect classification due to lighting changes	13
2.7 Average Image for random testing strategy on the AT&T database	15
2.8 First 12 Eigenfaces generated by using the random testing strategy on the AT&T database	15
2.9 Average Image of training images from random testing strategy on the Yale database	15
2.10 First 12 Eigenfaces from the training images for the random testing strategy on the Yale database	16

LIST OF FIGURES (Continued)

Figure	Page
2.11 Experimental results for the random testing strategy using the AT&T database	17
2.12 Examples of incorrect matching on the AT&T database using Eigenfaces method	18
2.13 Examples of incorrectly classified subjects	19
2.14 Experimental results for the leave-one-out testing strategy using the AT&T database	20
2.15 Experimental results for Eigenfaces method using the random testing strategy on the Yale database	21
2.16 Experimental results for Eigenfaces methods on the Yale database using the leave-one-out strategy	22
2.17 Test image and five nearest matches	22
2.18 First five nearest matches to the test images using Eigenfaces on the Yale database	23
2.19 First 12 Fisherfaces created from AT&T database	25
2.20 Fisherfaces created from the Yale database	25
2.21 Experimental results for the Fisherfaces algorithm using the random testing strategy	26
2.22 Experimental results for the Fisherfaces method using the leave-one-out testing strategy	27
2.23 Example of incorrect classification using Fisherfaces algorithm	28
2.24 Example of correctly classified image and it's nearest matches	28
2.25 Experimental results for the Fisherfaces method using the random testing strategy.	29
2.26 Experimental results for leave-one-out strategy using Fisherfaces	30

LIST OF FIGURES (Continued)

Figure	Page
2.27 Example of correctly classified image	30
2.28 Example of incorrectly classified test image, using Fisherfaces	31
2.29 Five nearest neighbors for classification using the Eigenfaces algorithm	31
3.1 Experimental results using KPCA with the random testing strategy on the AT&T database	36
3.2 Experimental results for KPCA on the AT&T database using the leave- one-out testing strategy	37
3.3 Example of incorrect classification using KPCA	38
3.4 Experimental results for KPCA on the Yale database using leave-one- out testing strategy	39
3.5 Example of incorrect classification using KPCA on the Yale database .	40

CHAPTER 1

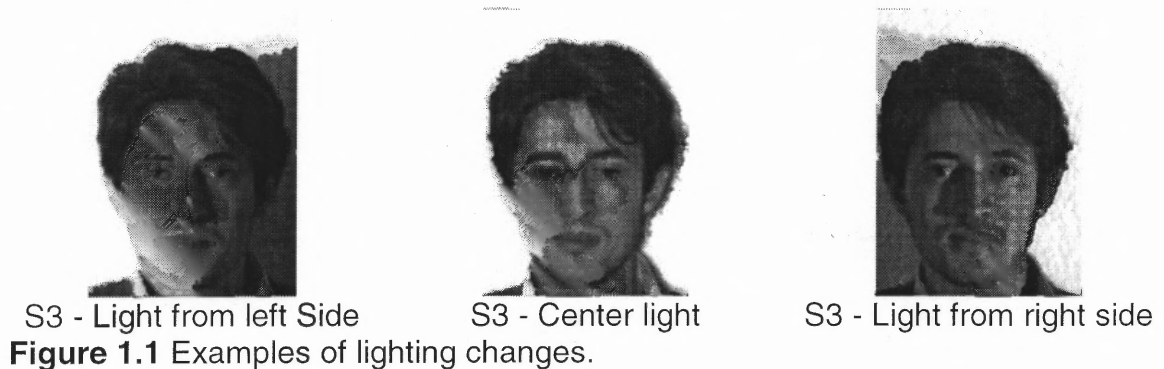
INTRODUCTION

Face recognition is the task of matching an unlabeled image of a face, to a face in a set of labeled face images. The main advantage of face recognition as a biometric is that it requires no interaction from the subject. This allows a subject to be identified without their consent or knowledge. In light of September 11, 2001, this has become a very valuable tool. Face recognition systems are being deployed throughout airports and city streets [7] in order to help law enforcement personnel to locate potential threats.

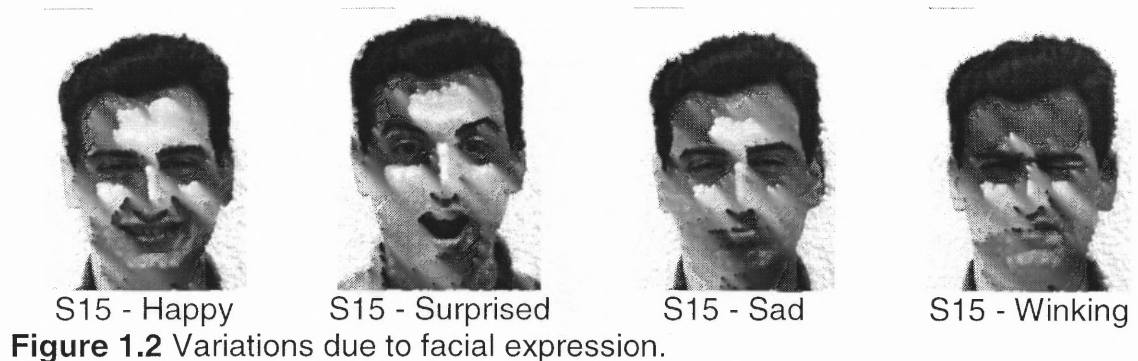
Unfortunately the task of face recognition, while simple for humans, is rather complex for computers. Commercial systems that have been used to date have been less than successful because of too many shortcomings [9]. By providing too many false positives, the systems become useless because security personnel have too much to check and eventually they will ignore them.

There are numerous variations in images of faces that can increase the difficulty of face recognition. All the variations can affect the image in different ways and cause recognition to be inaccurate. One of these variations is due to lighting. Variation in lighting can often drastically change one's appearance depending on the source of the light. As stated in [10], "the variations between the images of the same face due to illumination and viewing direction are almost always larger than the image variations due to change in face identity." Lighting changes depend on two factors: one is the direction of the light and the other is

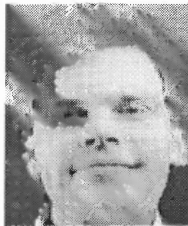
the intensity. Intensity can cause images to become washed out if it is high, or if the intensity is too low then the images will appear very dark, causing many features to become indistinguishable. Figure 1.1 shows some examples of lighting changes from the Yale face database [4].



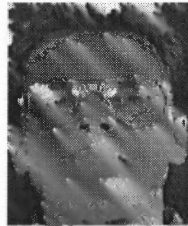
Another variation is due to facial expression. Different facial expressions can also cause changes to the face that could result in incorrect classification. Different expressions cause different portions of the face to become distorted. Figure 1.2 shows some examples of a subject with different expressions from the Yale database.



Occlusion can also cause problems with face recognition. This could be the result of extra clothing worn during winter, such as hats and scarves, or something as simple as glasses. Figure 1.3 has examples of occlusion occurring from glasses from the AT&T face database [5].



S36 – No glasses



S36 – Glasses



S4 – No glasses



S4 - Glasses

Figure 1.3 Variations due to glasses.

Another possible problem may be caused by the angle the face is at relative to the camera. The face might be tilted in any number of directions as well as in varying degrees. Figure 1.4 has some examples of variations due to face tilts from the AT&T database.



S1 - Normal



S1 - Right



S1 - Left



S1 - Up

Figure 1.4 Examples of faces at different angles.

Other problems that might occur have to do with the changing of one's physical appearance. This may be as simple as growing or removing facial hair, changing

a hairstyle, or wearing a simple disguise. It could also be the result of something more drastic, like plastic surgery.

Face recognition also has many ethical problems. Because cameras are becoming less expensive, they are being placed in increasingly more places. Face recognition allows law enforcement to not only see what is happening, but who is doing it. This might raise fears of a “Big Brother” type of society where the government knows where individuals are at all times. There are also complaints that the current inaccuracies do more harm than good. The American Civil Liberties Union (ACLU) has made public their opinions about current systems [8]. The belief is that if security personnel rely fully on inaccurate systems, then potentially obvious warnings will be overlooked. While this is true for inaccurate systems, a system that is accurate would be invaluable to law enforcement.

Face Recognition can also be used for security clearance, which does not have the same ethical issues as using it for locating individuals in a crowd. Besides its shortcomings and ethical issues, face recognition has enough positive qualities to warrant its study.

1.1 Objective

The objective of this paper is to determine if using the nonlinear Kernel PCA (KPCA) method as described in [3] will provide better accuracy in face recognition than traditional linear methods such as PCA [1] and FLD [2]. These various methods will be presented along with their respective results.

1.2 Background Information

The results were gathered based on testing against two different databases, using two different testing strategies on each database for a total of four result sets for each of the four algorithms. This first method of testing was to split the database into two halves. The first half is the training set and the second half is the testing set; this will be called the random testing strategy. The random strategy was designed to see how the algorithms will perform on a small set of training images, which does not contain all possible variations of the subjects. The second testing strategy is to use the leave-one-out strategy [2]. In this strategy, all the images are in the training set except the image currently being tested. This gives a larger training set, with more images trained per subject, making the task of recognition easier than with the random testing strategy.

The first database is the AT&T face database [4]. This database comprises ten images of each of 40 different subjects, for a total of 400 images. Each image is 92 x 112 pixels in size for a total of 10,304 pixels. For the random testing strategy, the test and training sets are equal halves, each containing five images of all 40 subjects, for a total of 200 images each. For the leave-one-out strategy, there are 399 training images and one testing image. Each subject is trained with ten images, except the testing subject, who only has nine training images; the tenth image is used for testing. All images were scaled by a factor of 1/16, for time considerations. Figure 1.5 is a snapshot of all the images in AT&T database.

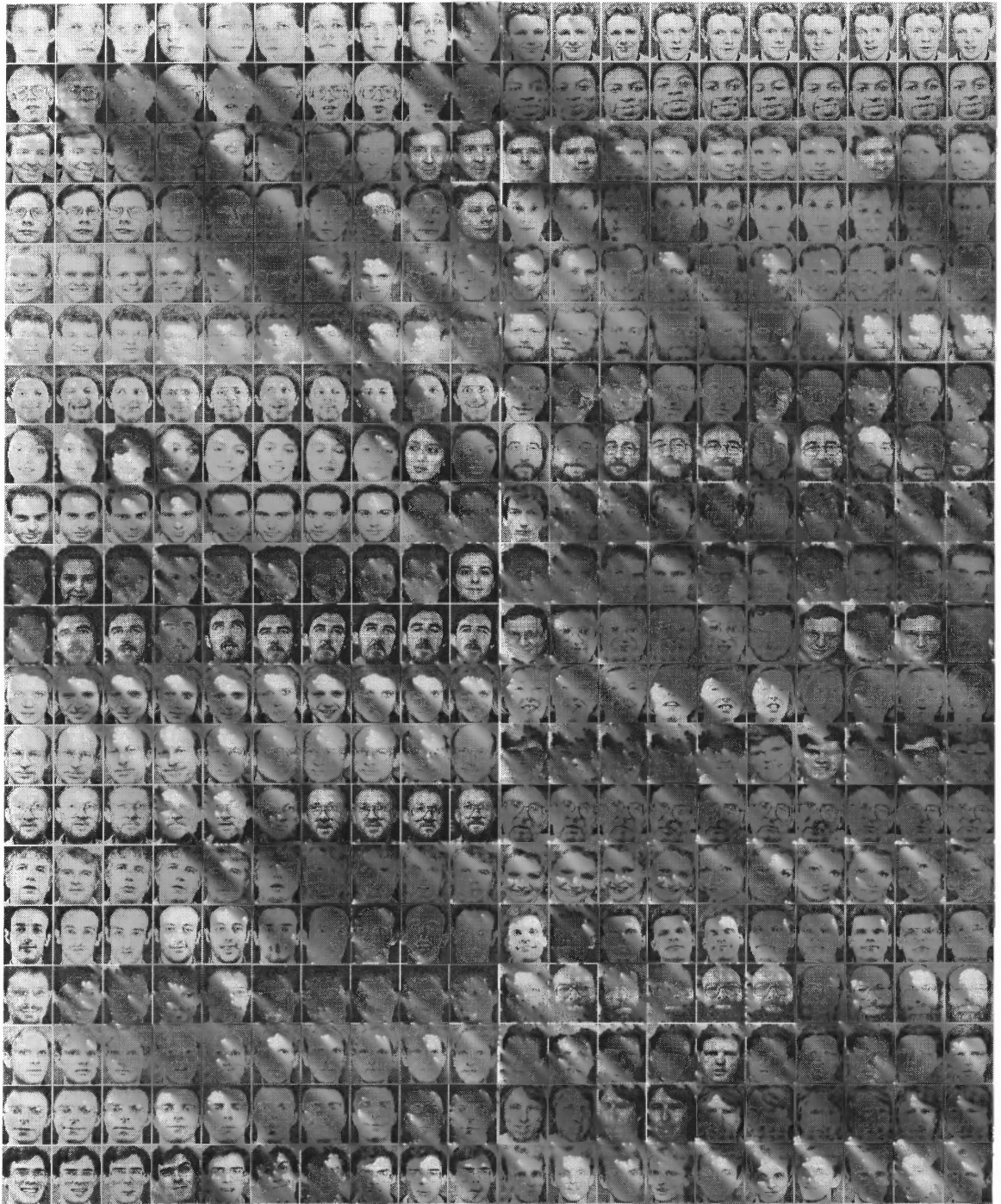


Figure 1.5 Snapshot of the 400 images from the AT&T face database.

The second database is the Yale face database [4]. This database contains 165 images of 15 subjects, each in 11 varying conditions. Each image started out at a size of 320x243 pixels. These images contained a lot of white space in the background and the faces were not centered. Because of this, they were cut to the size of 174x242 pixels and converted to 256 color grayscale PGM files, with each face centered within the image. The different conditions include facial expressions (happy, surprised, winking, sleepy, sad), lighting direction (center, right, left), and occlusions (glasses, no glasses). For the random testing strategy, four images of each of the 15 subjects were randomly chosen as the testing images, for a total of 60 testing images. This leaves seven images per subject for training for a total of 105 training images. For the leave-one-out method, each subject had 11 training images except the testing subject, who had ten training images with the 11th image being used for testing. All images were scaled by a factor of 1/36, for speed considerations. Figure 1.6 has some examples of the images that were resized and converted.

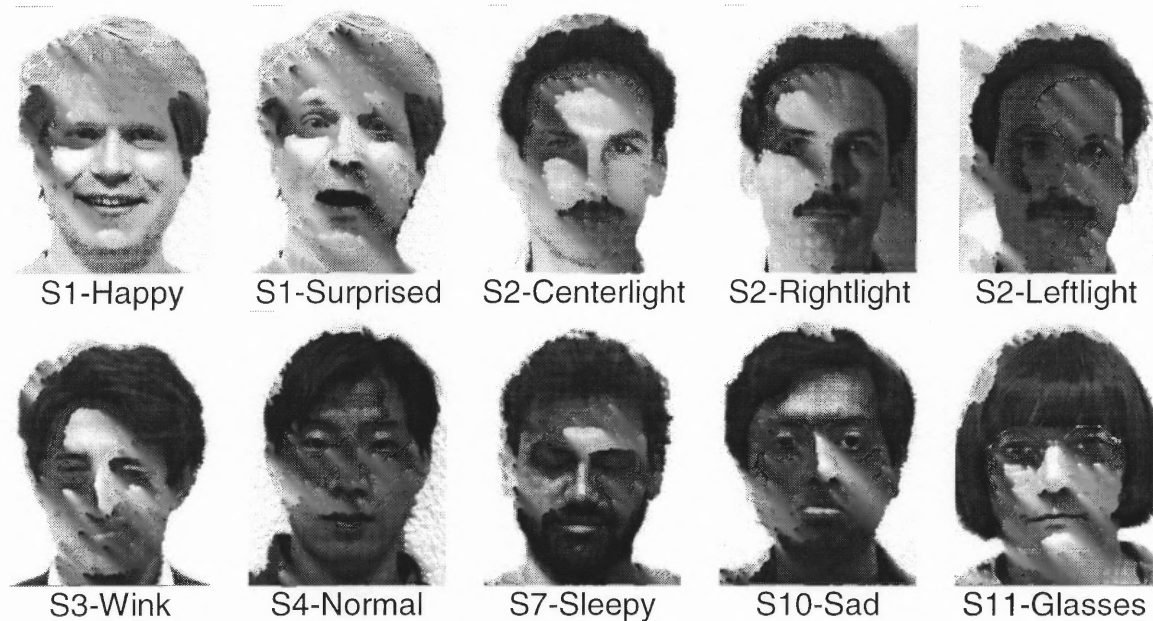


Figure 1.6 Examples from the Yale face database.

The reason for using two different databases was to test how the different algorithms performed over different types of data. The images in the AT&T database have a small range of variation in them. The images have slight angle changes, small expression changes, and occlusion changes due to glasses being worn only sometimes by certain subjects. The images in the Yale database have much larger variations due to facial expression and lighting.

As noted above, the images in both databases were scaled down from their original size for speed considerations. Note that not only does scaling down the images change the speed of recognition, it also impacts the effectiveness. Table 1.1 shows the error rates for the Eigenfaces algorithm using the random testing strategy, 40 components, and differently scaled images from the AT&T database.

Table 1.1 Results of the Eigenfaces Method Using Different Sized Images

Image Scale	Error Rate
1/4	10.5%
1/16	8.0%
1/36	11.0%
1/64	9.5%
1/100	8.0%
1/144	10.0%
1/196	20.0%

The error rate is the number of incorrectly classified test images divided by the total number of test images. Because the variations in size cause changes in accuracy, it was necessary to make all tests on one database to use the same size images.

All coding was done using Matlab from Mathworks, which provides powerful matrix and image manipulation tools that made coding and testing much easier.

CHAPTER 2

CURRENT METHODS

There are currently numerous ways to approach the problem of face recognition. One of the simplest methods is correlation or the nearest neighbor method. While correlation may work fairly well in idealized conditions, it does have a number of serious drawbacks, most important of all being a large recognition time. Recognition time is the time required to classify a test image. The Eigenfaces method uses Principal Component Analysis (PCA) to transform an image into a lower-dimensional subspace while still retaining the face differences in the new lower-dimension space. By lowering the dimension, the cost of performing the recognition is significantly reduced when compared with correlation. One of the problems with Eigenfaces is that it does not take class-specific information into account and thus it maximizes the scatter over all of the classes [2], where a class is all the images of a particular subject and scatter is the variations between images or sets of images. In doing so, it retains differences in facial expression and lighting that should be ignored for the purpose of face recognition. The Fisherfaces method seeks to correct this problem by using the Fisher Linear Discriminant (FLD) to select the principal components such that “the ratio of the between-class scatter and the within-class scatter is maximized” [2]. This chapter will present the three algorithms in detail and present the results for each one, using both databases and both testing strategies.

2.1 Correlation

The correlation method uses a Euclidian distance between the training and test images in height x width dimension. To do this, the training images are converted into column vectors of size height x width. Each test image is then converted into a column vector of the same size and the label of the training image that is the nearest in the image space, is used as the label for the test image. Using the random testing strategy with the AT&T database, the correlation method performed with 92% accuracy, classifying 16 out of 200 incorrectly. Using the leave-one-out method and the AT&T database, correlation performed with 97.5% accuracy, incorrectly classifying ten of the 400 images. When using both methods, the images that resulted in errors mostly had features that were similar to other subjects. In particular, subject 40 and subject 5 closely resemble each other. Some others had similar facial hair and glasses. Figures 2.1 – 2.4 are examples of the incorrectly classified images using the random testing strategy.



Figure 2.1 Example of a test image (left) being incorrectly matched to a subject similar in appearance.

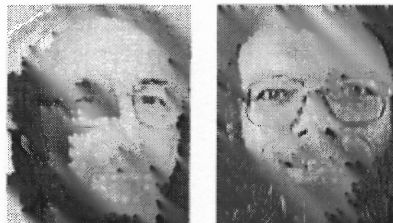


Figure 2.2 Example of a test image (left) being matched due to similar features, in this case glasses, beard, and hair.



Figure 2.3 Example of incorrect classification due to glasses. In this case, the test subject (left) was incorrectly matched to another subject without glasses in the top section, and incorrectly matched to a subject with glasses in the bottom.



Figure 2.4 Example of incorrect classification where very little similarity is obvious.

With the Yale database, using the random testing strategy, correlation had 83.3% accuracy, incorrectly classifying ten of the 60 test images. With the leave-one-out strategy, correlation had 83% accuracy, incorrectly matching 28 of 165 test images. When examining the error images for both methods, it becomes obvious that lighting accounts for most of the error. 90% (9/10) of the errors using the random testing strategy, and 93% (26/28) for the leave-one-out strategy, were due to lighting changes. Figures 2.5 and 2.6 are examples of the incorrectly

classified images from the Yale database using the leave-one-out testing strategy.



Figure 2.5 Example of incorrect classification due to lighting. The lighting causes the subject to appear darker, thus causing him to be matched with a subject of darker skin color.



Figure 2.6 Example of incorrect classification due to lighting changes. In this case it seems the background was the major reason for the incorrect classification.

Although correlation is fairly accurate in simple situations, it does have some disadvantages. It requires a large training set, and variations in lighting cause problems. This could be solved by having training images that have examples of all possible lighting conditions, but this would be difficult to obtain as well as requiring a large amount of storage space. More images would also cause larger recognition times because there would be that many more images to compare against [2].

2.2 Eigenfaces

The Eigenfaces method as posed in [1], attempts to use dimension reduction to greatly improve upon the speed of recognition. With correlation, finding the nearest neighbor (training image least distance from the test image) means computing distances in the width x height dimension, which for even small images, is an expensive operation. Using PCA, the Eigenfaces method extracts the principal components from the covariance matrix by finding its largest eigenvalues. Let M be the number of images in the training set, and let vectors $\Gamma_1, \Gamma_2, \Gamma_3, \dots, \Gamma_M$ be the images in the training set. Note that each Γ_i is a vector of width x height dimension. We also define

$$\Psi = \frac{1}{M} \sum_{n=1}^M \Gamma_n, \quad (2.2)$$

which is the average over all the image in the training set. Then we define the covariance matrix as

$$C = \frac{1}{M} \sum_{n=1}^M \Phi_n \Phi_n^T, \quad (2.1)$$

where $\Phi_n = \Gamma_n - \Psi$ and the superscript T denotes transpose. Once the covariance matrix has been calculated, the next step is to compute the eigenvalues and eigenvectors. A value M^1 is chosen for the number of components to be used. The M^1 eigenvectors with the largest corresponding eigenvalues are used as the principal components. This new lower dimension is known as “face space”. The eigenvectors are known as “Eigenfaces” due to their resemblance to human faces. Below are some examples of the average image and Eigenfaces generated from the AT&T and Yale databases.

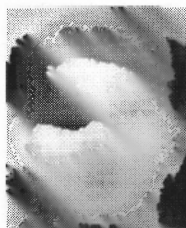


Figure 2.7 Average Image for random testing strategy on the AT&T database.

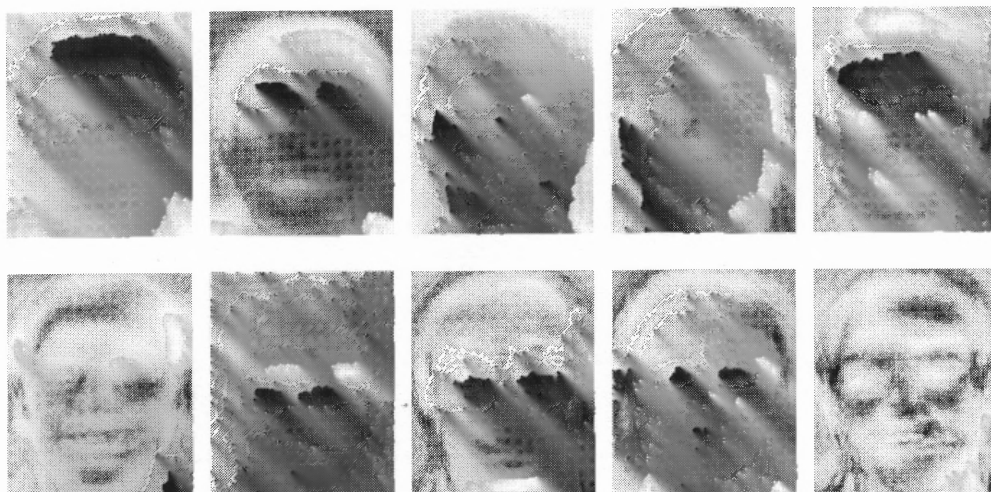


Figure 2.8 First 12 Eigenfaces generated by using the random testing strategy on the AT&T database.



Figure 2.9 Average Image of training images from random testing strategy on the Yale database.

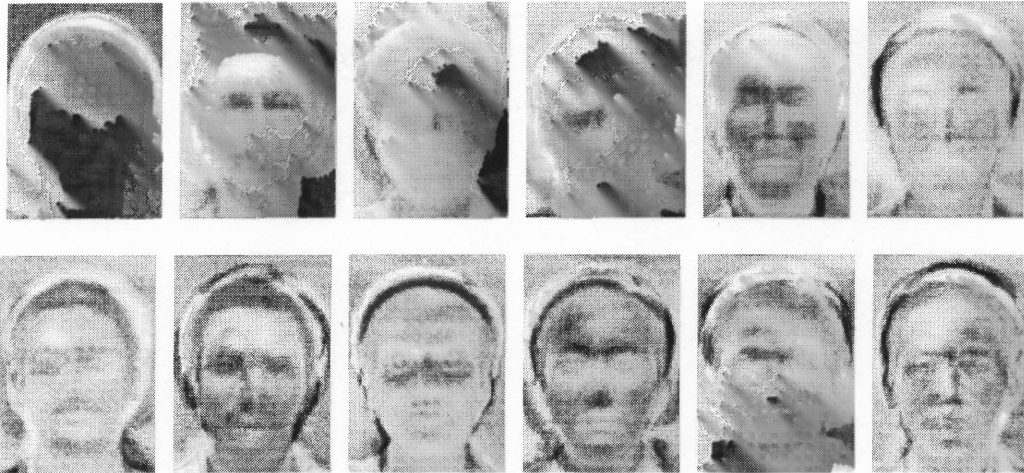


Figure 2.10 First 12 Eigenfaces from the training images for the random testing strategy on the Yale database.

Once the principal components are chosen, the learning set is projected into the new lower-dimension space and is stored; these are the values that will be checked against during the recognition step. When an image is being tested, it is first projected into the lower-dimension space and then checked against the training set projections, and the nearest neighbor is used as the match.

The performance of the Eigenfaces method on the AT&T database matched or exceeded that of the correlation method, but did it in much less time. The correlation took about 2 minutes to classify 200 images using the random testing strategy on the AT&T database, while the Eigenfaces method only took about ten seconds. For the random testing strategy using 90 components, the Eigenfaces method had an error rate of 8% which is exactly the error rate of the correlation method. Figure 2.11 shows the error rates as the number of components is increased.

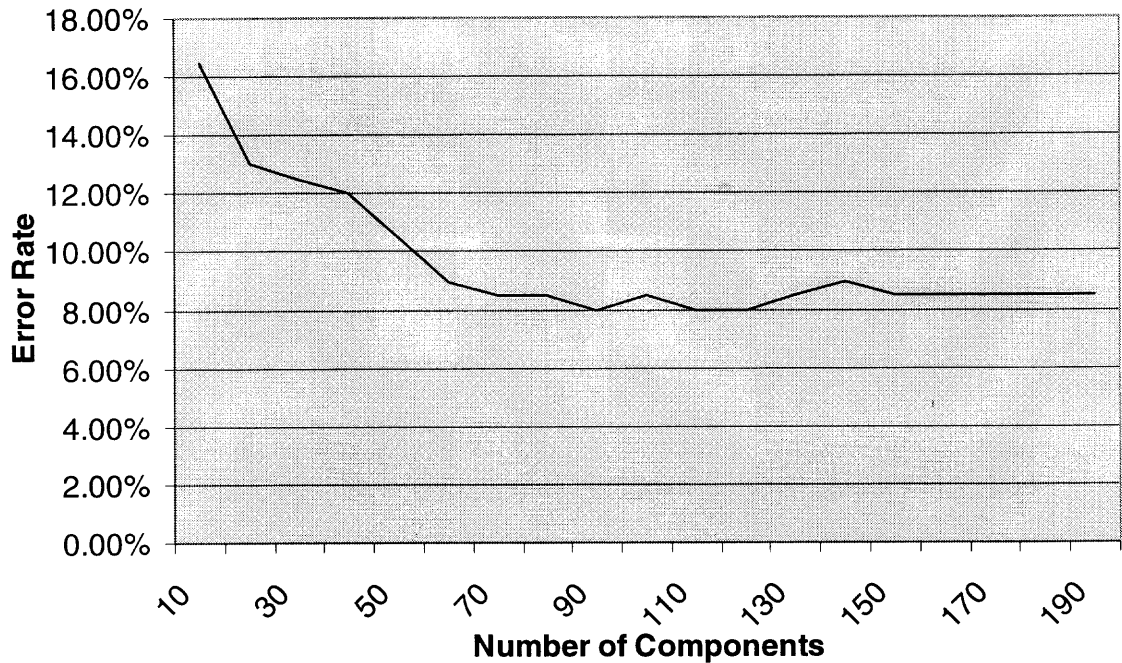


Figure 2.11 Experimental results for the random testing strategy using the AT&T database.

Figure 2.12 shows one of the test images that was incorrectly classified along with the five nearest training images. The second nearest match was in fact correct, but the other four were not. Out of the 17 incorrect images in this test, only 10 had the correct match somewhere in the five nearest matches. This example was done using 40 components.

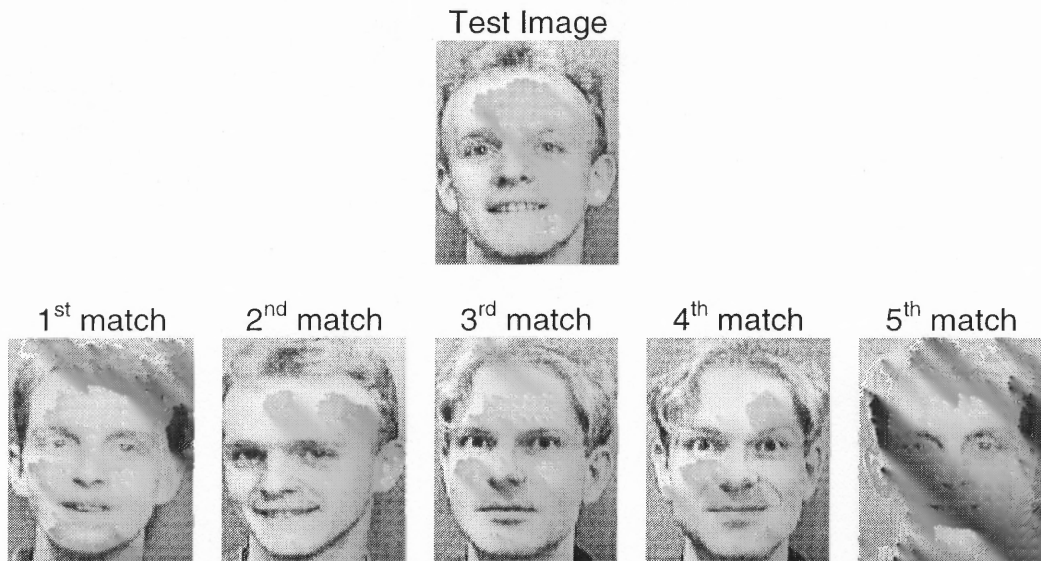


Figure 2.12 Examples of incorrect matching on the AT&T database using Eigenfaces method.

Figure 2.13 is another example using 40 components, that shows the correct answer nowhere in the five nearest training images. It is not until the 17th nearest match, that the correct classification is found. The nearby images were all similar in appearance. Subject 2, who appears in matches one, two, and five, has similar hair and facial structure. Subject 15, who appears in matches three and four, has a similarly shaped head as well as similar eyes. The skin color in all five incorrect matches is also close to the test subject.

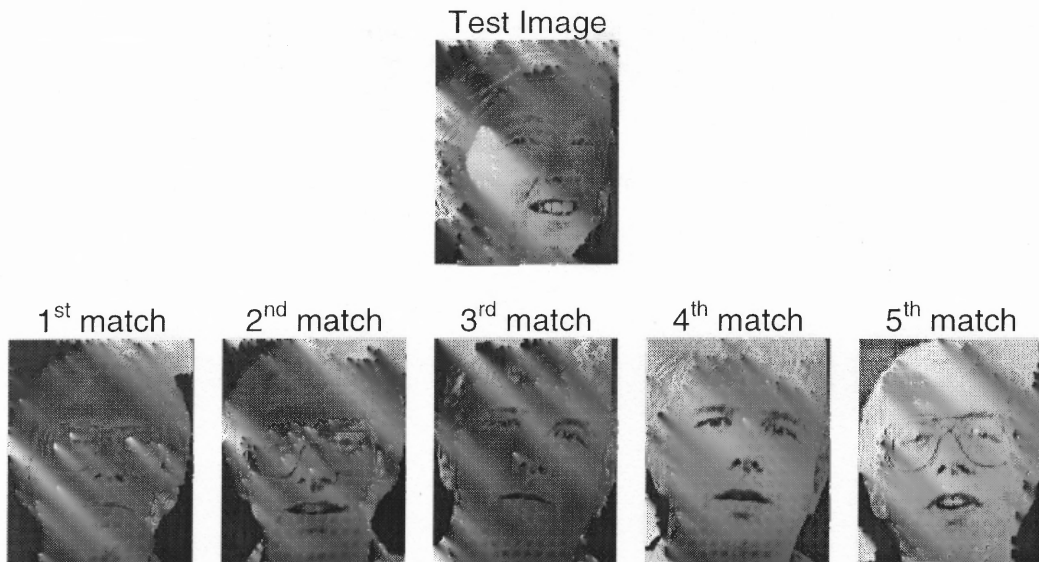


Figure 2.13 Examples of incorrectly classified subjects.

For the leave-one-out strategy, the Eigenfaces method outperformed the correlation method. Using 40 components, the Eigenfaces method incorrectly classified six out of the 400 images for an error rate of 1.5%. All of the test images that were incorrectly classified using the leave-one-out strategy with the Eigenfaces method were also incorrectly matched using the correlation method. All involved subjects looked similar enough that their features could not keep the classes separate in the M^1 dimension space. Figure 2.14 is a graph showing the error rates using an increasing number of components.

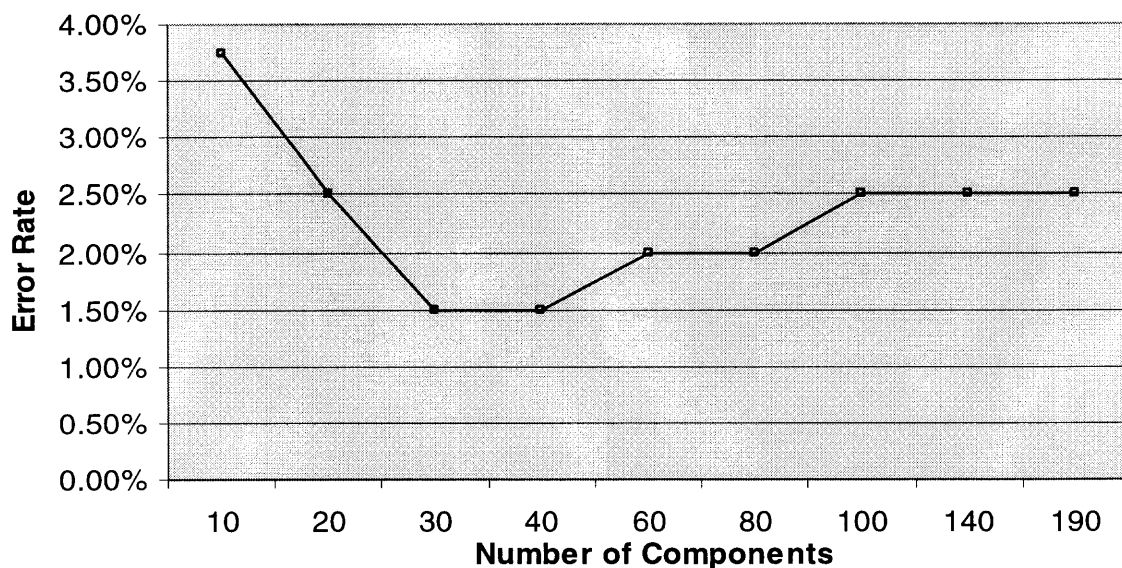


Figure 2.14 Experimental results for the leave-one-out testing strategy using the AT&T database.

As with the AT&T database, the different testing strategies for the Eigenfaces method on the Yale database performed about equal to the correlation method. For the random testing strategy, Eigenfaces had an error rate of 15.0%, incorrectly matching ten out of the 60 test images. Figure 2.15 is a graph showing the performance using an increasing number of components.

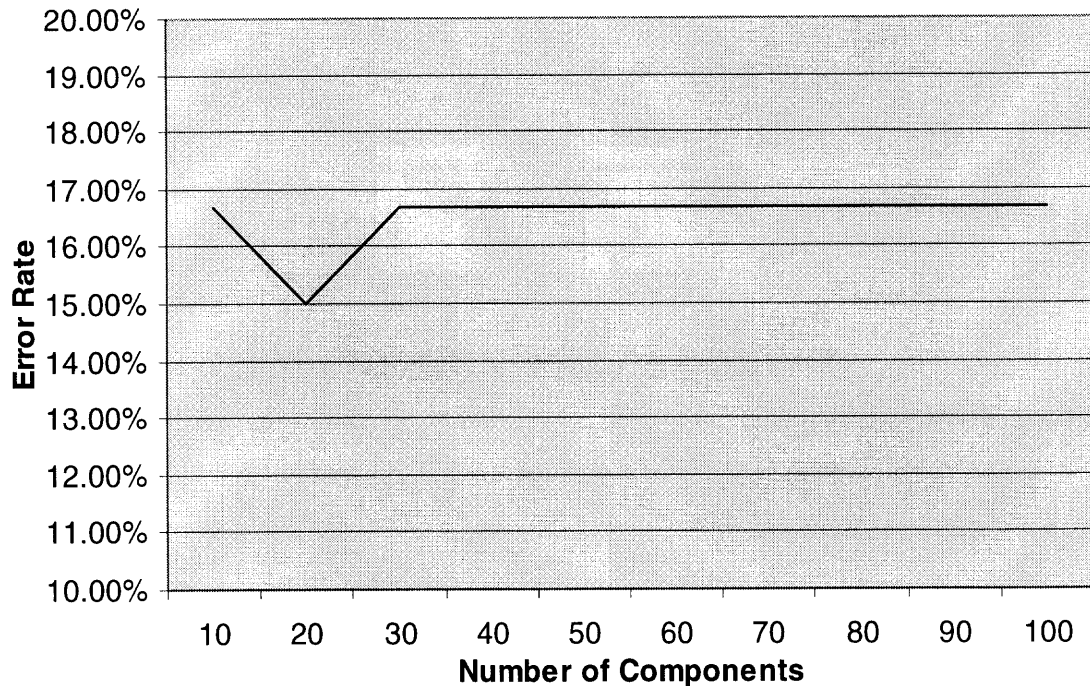


Figure 2.15 Experimental results for Eigenfaces method using the random testing strategy on the Yale database.

For the leave-one-out strategy, an additional step was taken. It has been shown that ignoring the three highest ranked eigenvectors helps reduce error from variations in lighting [2]. This happens because the first three eigenvectors tend to account for most of the variations from lighting. The method was done along with the normal algorithm. While the normal version performed equal with correlation, leaving out the first three components improved the accuracy considerably. Leaving out the first three components achieved an error rate of 10.9% (18/165), while using them achieved an error rate of 16.9% (28/165). Figure 2.16 shows the performance of both Eigenfaces methods and how they perform with an increasing number of components.

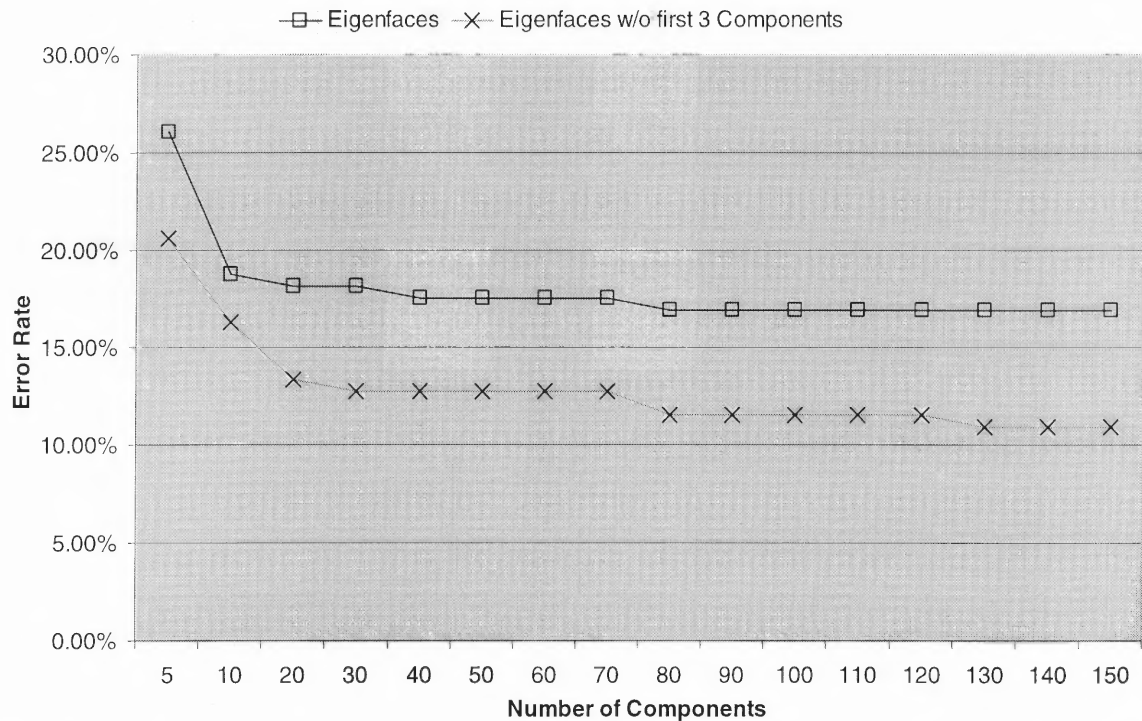


Figure 2.16 Experimental results for Eigenfaces methods on the Yale database using the leave-one-out strategy.

Figure 2.17 shows an example of one of the incorrect classifications and its five nearest matches, when using the Eigenfaces method and leave-one-out testing strategy.

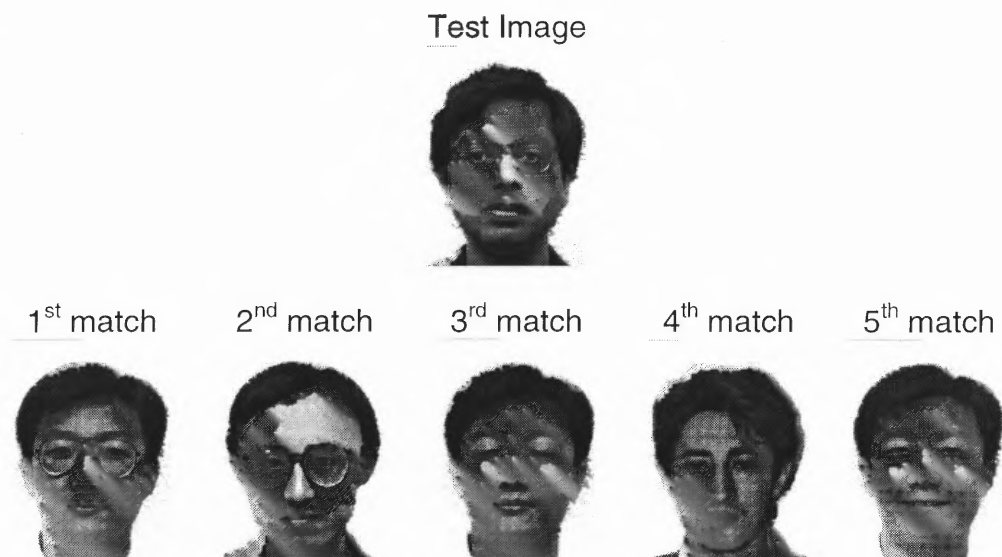


Figure 2.17 Test image and five nearest matches.

The subject in the test image only wears glasses for one of the pictures in the database, which means that none of the training images have him wearing glasses, so the nearest matches are to subjects who are wearing glasses who also have a similar appearance. The subject who appears in the first, third, and fifth image has similar features, which accounts for the matches without glasses.

Figure 2.18 is an example of another incorrect match using the Eigenfaces method on the Yale database. This time, the correct subject appears twice in the first five matches. Hair and skin color seemed to be the most decisive features for this image. The first match has similar skin tone; the second subject has similar hair as well as facial hair. The third match is the correct subject. What is interesting is that it matched the only image of the subject with glasses.

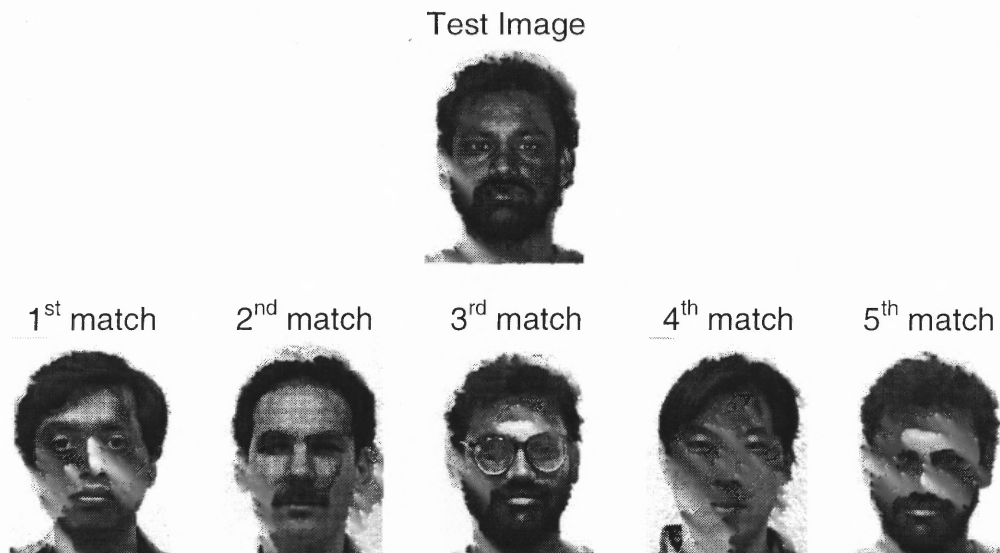


Figure 2.18 First five nearest matches to the test images using Eigenfaces on the Yale database.

Overall, the Eigenfaces method performed as well and in some cases, better than, the correlation method. Its main advantage over the correlation method is space and time. Instead of storing whole images, only the projected vectors need to be stored for the purpose of recognition. This makes it more feasible to store for each subject many different variations, such as lighting and facial expressions. With both databases, the images that correlation incorrectly classified, the Eigenfaces method also misclassified.

2.3 Fisherfaces

While the Eigenfaces method is about as accurate as the correlation method and is much faster, there is still much room for improvement. The Fisherfaces method proposed in [2] seeks to improve upon the Eigenfaces method. One of the main problems with the Eigenfaces method is that it “yields projection directions that maximize the total scatter across all classes”; thus, it “retains unwanted variations due to lighting and facial expression” [2]. The Fisherfaces method tries to correct this by selecting a covariance matrix “in such a way that the ratio of the between-class scatter and the within-class scatter is maximized” [2]. The between-class scatter is defined as

$$S_B = \sum_{i=1}^c N_i (\mu_i - \mu)(\mu_i - \mu)^T \quad (2.3)$$

and the within-class scatter is defined as

$$S_w = \sum_{i=1}^c \sum_{x_k \in X_i} (x_k - \mu)(x_k - \mu)^T, \quad (2.4)$$

where X is the set of all classes, μ_i is the average image for class X_i , N_i is the number of images in class X_i , μ is the average of the training images, and c is the number of classes. Once these are computed, the Eigenfaces method is used to create the component matrix W_{pca} , using an M^1 value of $N - c$. Then the FLD matrix is computed as follows:

$$W_{fld} = \arg \max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|}, \quad (2.5)$$

where $|A|$ denotes the determinant of a matrix A . Then W_{opt} is computed as

$$W_{opt} = W_{fld}^T W_{pca}^T. \quad (2.6)$$

Once W_{opt} is found, the images in the learning set are projected into the lower-dimension space, as was done with the Eigenfaces method. The images below are some examples of the Fisherfaces created via this algorithm.

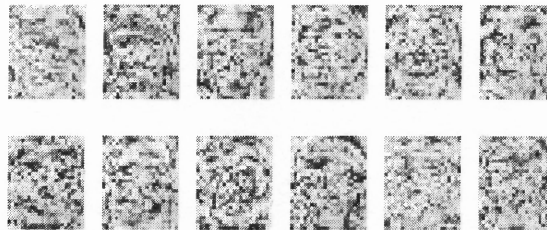


Figure 2.19 First 12 Fisherfaces created from AT&T database.



Figure 2.20 First 12 Fisherfaces created from the Yale database.

The performance of the Fisherfaces algorithm on the AT&T databases using both testing strategies, were slightly worse than the Eigenfaces and correlation methods. Using the random testing strategy, the lowest error rate that was achieved was with 39 components and that was 9.5%. Figure 2.21 shows the performance using an increasing number of components.

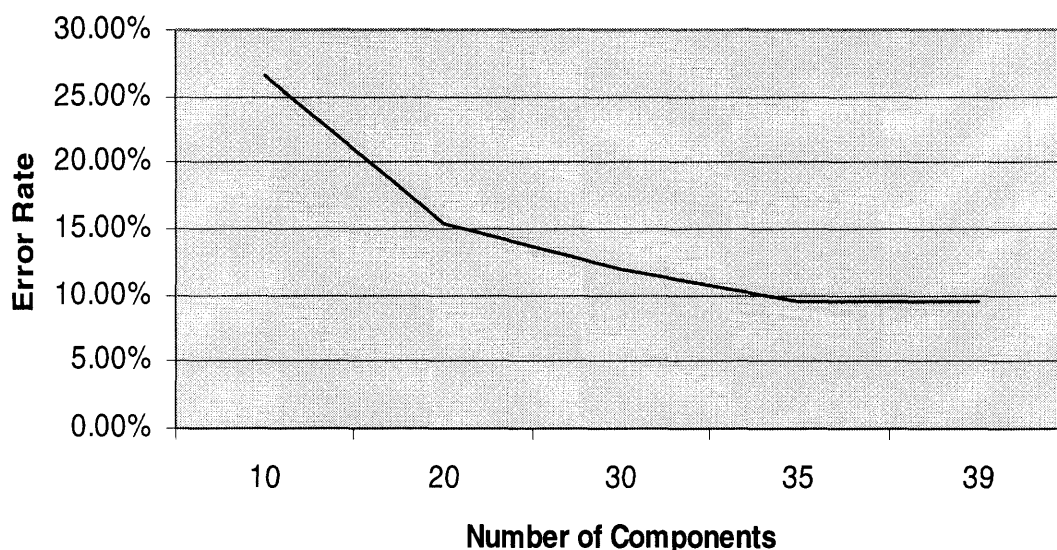


Figure 2.21 Experimental results for the Fisherfaces algorithm using the random testing strategy. It reaches its lowest error rate at 39 components with an error rate of 9.5%.

The results for Fisherfaces using the leave-one-out strategy were similar to those of the random strategy. The results were slightly worse than the other algorithms, achieving the lowest error rate of 2.0% using 39 components. Figure 2.22 shows the performance of the Fisherfaces algorithm using the leave-one-out strategy.

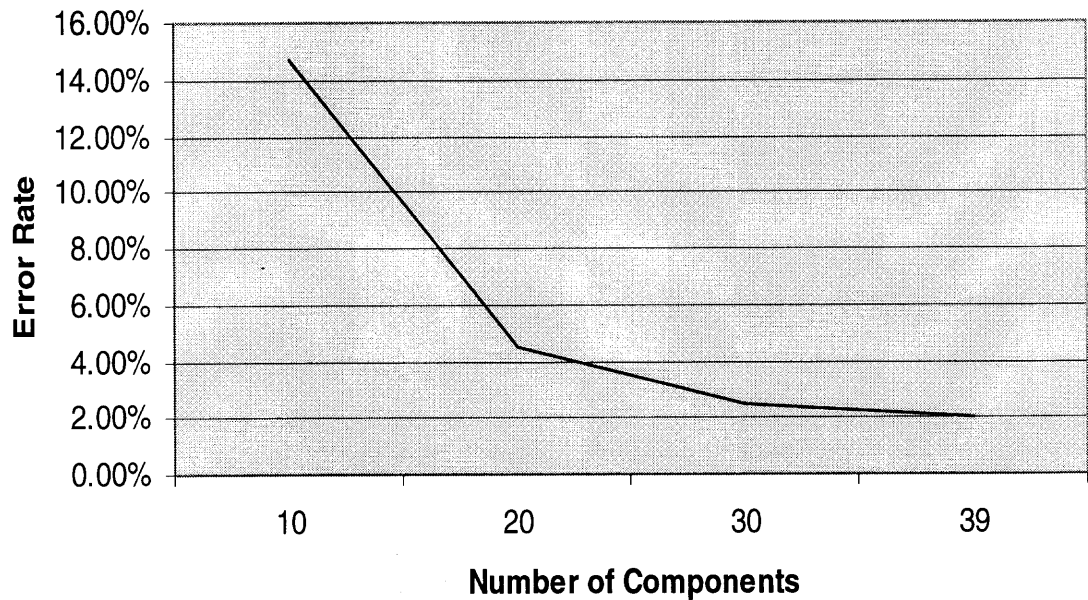


Figure 2.22 Experimental results for the Fisherfaces method using the leave-one-out testing strategy. The lowest error rate was at 39 components, which was 2.0%.

Upon examining the results of the Fisherfaces experiments, it becomes clear that this method does a very good job of separating the different classes. In the cases that classification is correct, all of the nearest neighbors to the test image will also be from the correct class. This also means that when a test image is incorrectly classified, it will have nearest neighbors that are all of the same incorrect class. Figure 2.23 is an example of an incorrectly classified test image. In this case, the first ten nearest matches were all from the same incorrect class. It was not until the 15th match that the correct classification was found. What this shows is that the images within the classes are more tightly clustered, which should lead to better recognition rates.

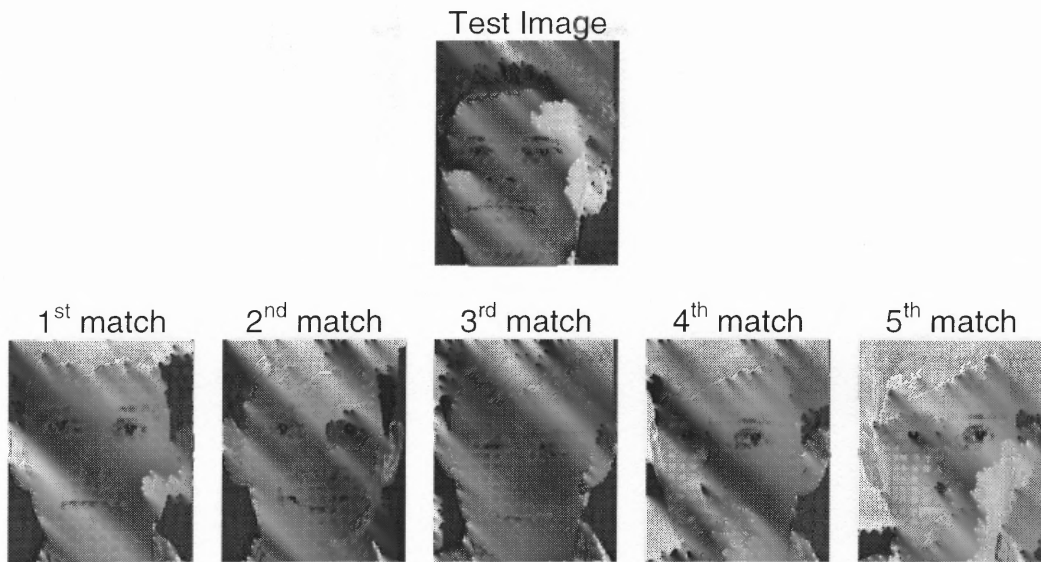


Figure 2.23 Example of incorrect classification using Fisherfaces algorithm.

Figure 2.24 is an example of the same testing subject as in Figure 2.23, but in this case the image was classified correctly. In this case, the first nine nearest matches are from the same correct class. The nearest images are not only correct, but they also contain different variations such as lighting, tilts, facial expressions, and glasses.

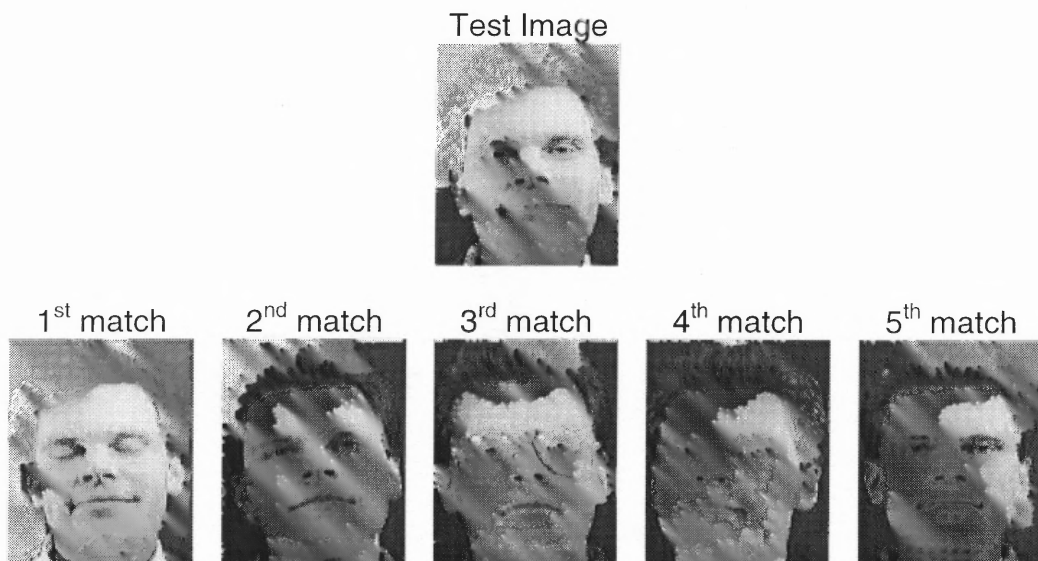


Figure 2.24 Example of correctly classified image and its nearest matches.

Fisherfaces performed considerably better than the other algorithms when testing on the Yale database. This is as expected as Fisherfaces was designed to handle within-class variations better than the other algorithms. Using the random testing strategy with only ten components, the Fisherfaces algorithm had a low error rate of 11.7%, which is 4% lower than the other algorithms. Figure 2.25 shows the results of the random testing using an increasing number of components.

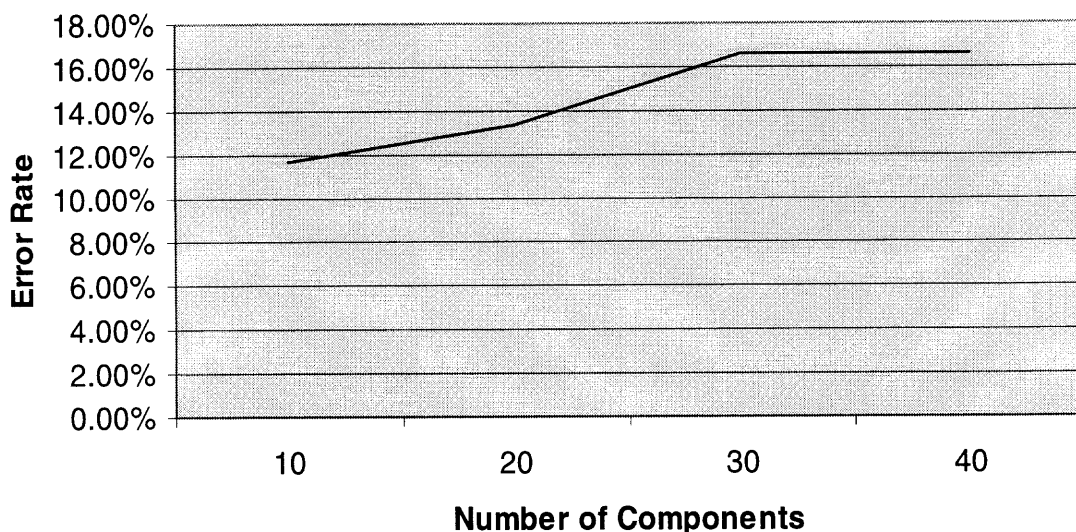


Figure 2.25 Experimental results for the Fisherfaces method using the random testing strategy.

With the leave-one-out strategy, Fisherfaces performed slightly better than the Eigenfaces method when the first three components are ignored. The error rate of 10% was achieved using 20 components. Figure 2.26 shows the results for the leave-one-out testing strategy.

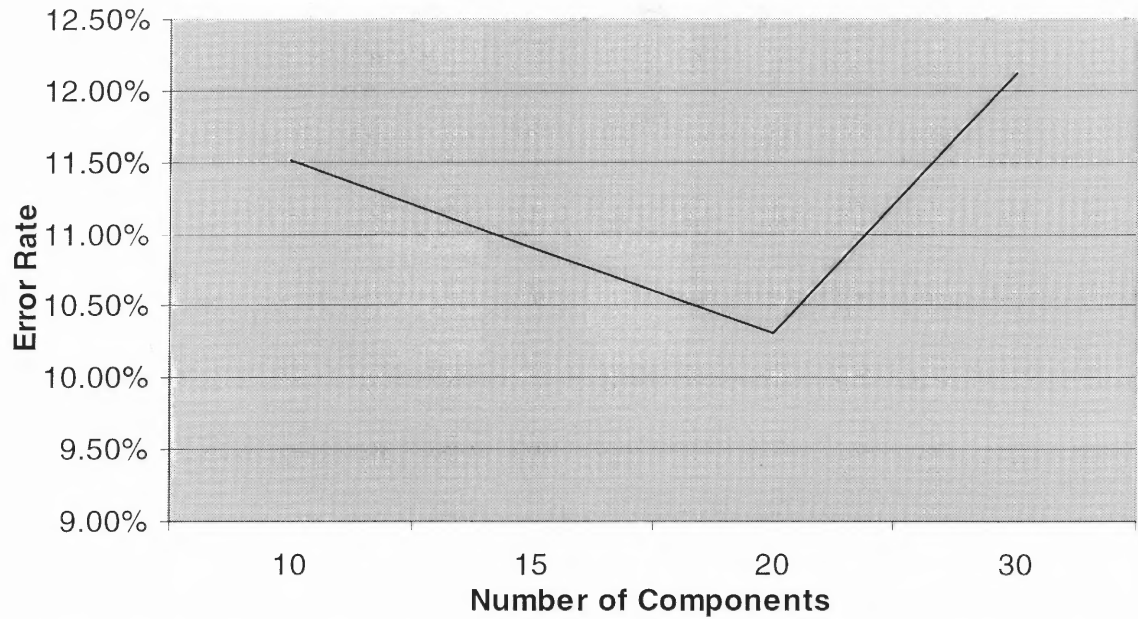


Figure 2.26 Experimental results for leave-one-out strategy using Fisherfaces.

Even with examples of the more extreme variations of the Yale database, the Fisherfaces algorithm still managed to keep the classes more tightly clustered than the other algorithms. Figure 2.27 shows an example of a correctly classified image and its five nearest matches when using the leave-one-out strategy.

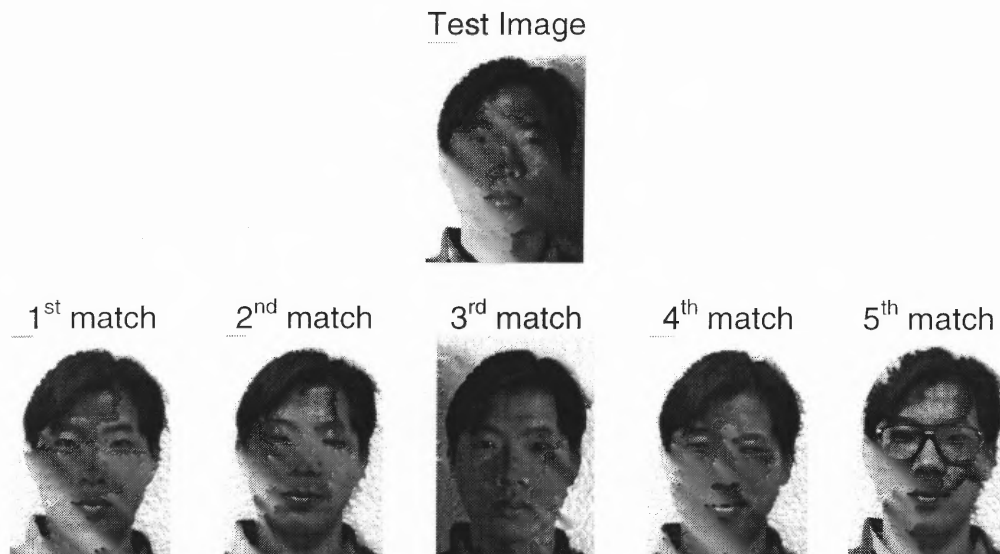


Figure 2.27 Example of correctly classified image.

Again, in the case that an image was incorrectly classified, it was matched to numerous images of the same incorrect class. Figure 2.28 shows one such example where the same test subject that appeared in Figure 2.27 is incorrectly classified using the leave-one-out strategy.

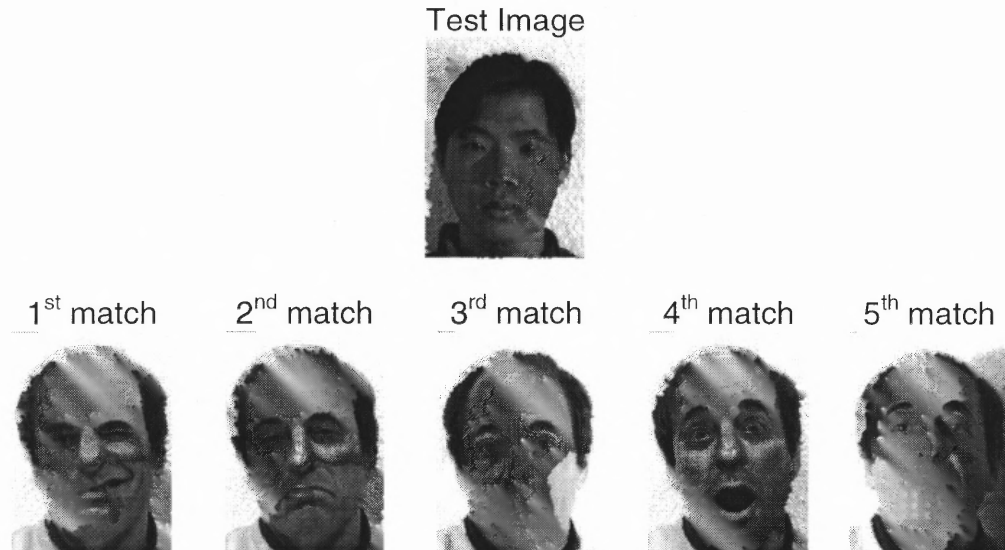


Figure 2.28 Example of incorrectly classified test image, using Fisherfaces.

To contrast this, Figure 2.29 shows the five nearest neighbors for the same test image except using the Eigenfaces method. What is seen is that the test image is nearest to all images that have a right light source.

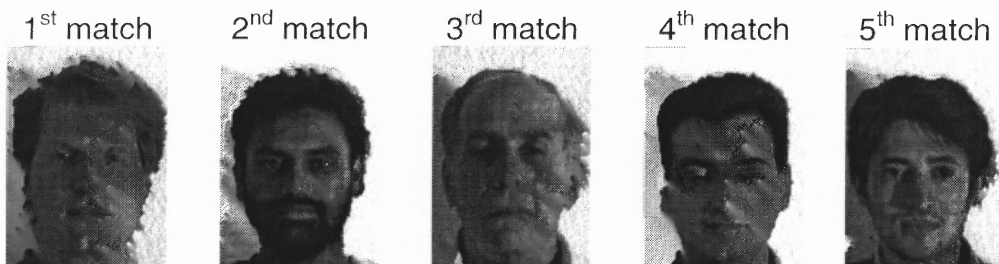


Figure 2.29 Five nearest neighbors of classification using the Eigenfaces algorithm.

Overall, the Fisherfaces algorithm performed as well and, in the case of the Yale database, better than the Eigenfaces and correlation methods. This is because the algorithm was designed to take advantage of the known within-class differences to extract features that better represented the classes.

CHAPTER 3

KERNEL PCA

3.1 Description

PCA attempts to extract linear features from the input data set by means of “an orthogonal transformation of the coordinate system” [6] in which the data is described. Because PCA is a linear technique, it ignores any possible features that exist in nonlinear feature space [6]. Kernel Principal Component Analysis (KPCA) “computes the principal components of the data set mapped nonlinearly into some high dimensional feature space F ” [3]. Using the algorithm as described in [3], the feature space is calculated in the following way. Given a set of centered observations ($\sum_{i=1}^M x_i = 0$), x_k , where $k=1\dots M$, the traditional way of formulating the covariance matrix using PCA is

$$C = \frac{1}{M} \sum_{j=1}^M x_j x_j^T. \quad (3.1)$$

Now the nonlinear feature space F must be defined. F is related to the input space by a possibly nonlinear map

$$\Phi : R^N \mapsto F. \quad (3.2)$$

The covariance matrix in F can now be defined as

$$\bar{C} = \frac{1}{M} \sum_{j=1}^M \Phi(x_j) \Phi(x_j)^T. \quad (3.3)$$

We then determine each eigenvalue λ and corresponding eigenvector V of \bar{C} , which satisfy

$$\lambda V = \overline{C}V. \quad (3.4)$$

All solutions of V with $\lambda \neq 0$ lie in the span $\Phi(x_1), \dots, \Phi(x_M)$. There also exists coefficients α_i such that

$$V = \sum_{i=1}^M \alpha_i \Phi(x_i) \quad (3.5)$$

The $M \times M$ kernel matrix $K = (K_{ij} : i, j = 1, 2, \dots, M)$ is then defined with

$$k_{ij} = (\Phi(x_i) \cdot \Phi(x_j)), \quad (3.6)$$

where \cdot denotes dot product. Thus the KPCA problem is determining k to satisfy

$$M\lambda K\alpha = k^2\alpha, \quad (3.7)$$

where α denotes the column vector with entries $\alpha_1, \dots, \alpha_M$. To find the solutions of (3.7), one solves

$$M\lambda\alpha = k\alpha. \quad (3.8)$$

Once this equation has been solved, the images can be projected into the lower-dimensional space, using the top M^1 eigenvectors. Then the testing images are projected into the lower-dimension space and using the nearest neighbor method, they are classified.

To compute the dot products, kernel representations are used. Kernel representations are functions that allow the value of the dot product in F to be computed without carrying out the map Φ [3]. There are a number of different kernels that can be used. First is the Polynomial kernel defined as

$$k(x, y) = (x \cdot y)^d, \quad (3.9)$$

where d is the degree and when d is equal to one, this algorithm will be equivalent to the Eigenfaces method. Other possible functions are the radial basis function (RBF)

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right), \quad (3.10)$$

Where σ is a constant, and the sigmoid kernel

$$k(x, y) = \tanh(w(x \cdot y) + \Theta), \quad (3.11)$$

where Θ and w are constants.

For the purposes of testing, only the polynomial, using degrees two through six, and the RBF kernel functions were used. The scaling on the images when used with the KPCA became very important. As the degrees of the polynomial kernel function got larger, so did the values. Eventually the values reached infinity at which point the calculations returned imaginary values for the eigenvectors. For the RBF kernel function, if the value for σ was too large or too small, the exp function would return ones and zeroes, which causes the eigenvector calculations to be incorrect.

3.2 Results

The performance of the KPCA algorithms on the AT&T database using both testing strategies was slightly worse than the Eigenfaces and correlation algorithms. With the random testing strategy, the lowest error rate achieved was 8.5% which used the RBF kernel function and 80 components. Figure 3.1 shows the performance for the kernel functions using the random testing strategy on the AT&T database.

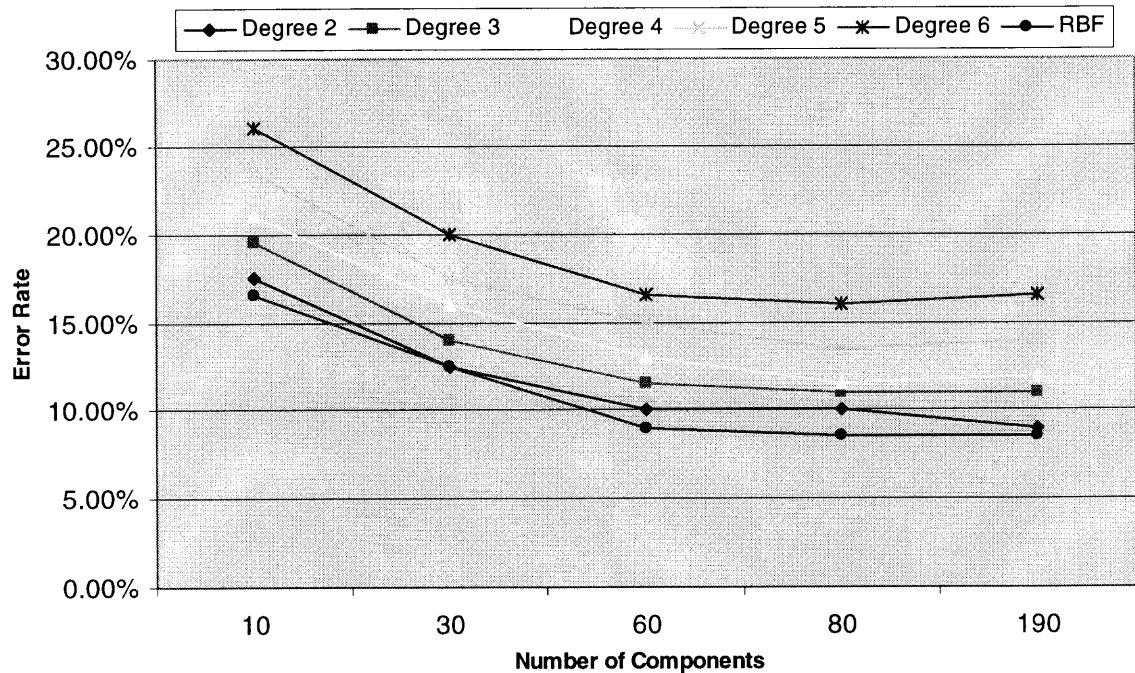


Figure 3.1 Experimental results using KPCA with the random testing strategy on the AT&T database.

The RBF function had the lowest error rates with the second degree polynomial being close to it. From then on, the higher the degree, the larger the error rate.

The leave-one-out testing strategy yielded similar results to the random testing strategy. The lowest error rate of 1.5% was achieved using the RBF kernel function and 40 components. This error rate is equivalent to the error rate achieved by the Eigenfaces algorithm. Figure 3.2 shows the results for the KPCA algorithm using the leave-one-out testing strategy on the AT&T database.

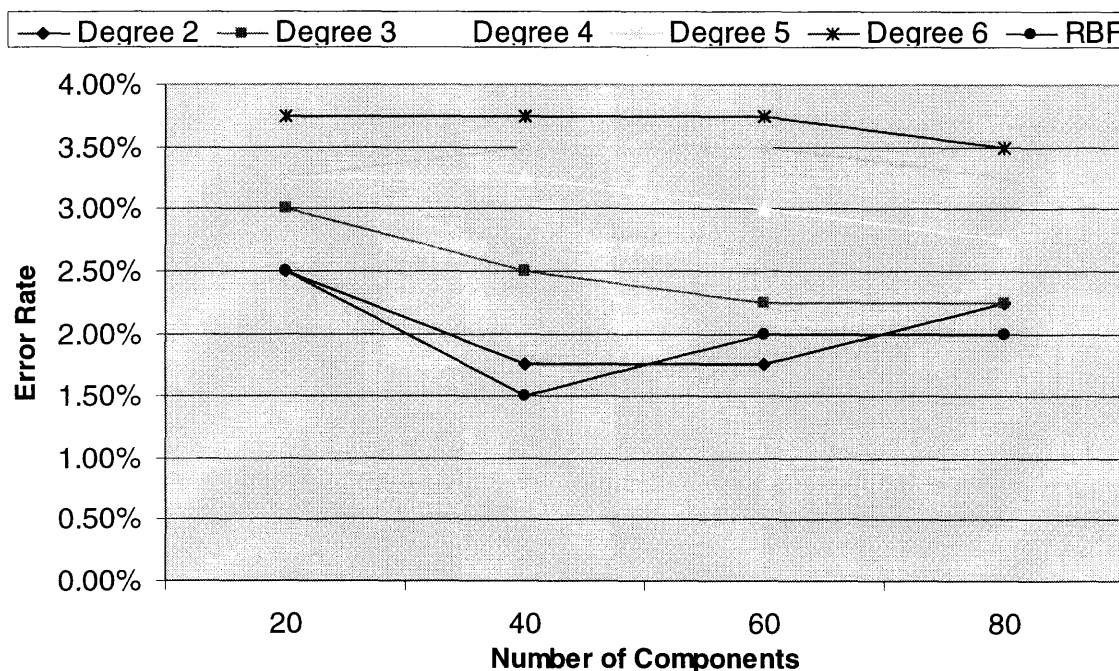


Figure 3.2 Experimental results for KPCA on the AT&T database using the leave-one-out testing strategy.

The nearest matches for KPCA were similar to those of the Eigenfaces method. The classes are not as closely clustered as they were with the Fisherfaces method. Figure 3.3 shows an example of an incorrect classification and its five nearest neighbors. The features that seem to have the largest effect are the hair, skin color, and pronounced cheek bones. In the training image that is correct (5th), the subject is tilting her head in the same way that she is in the test image.

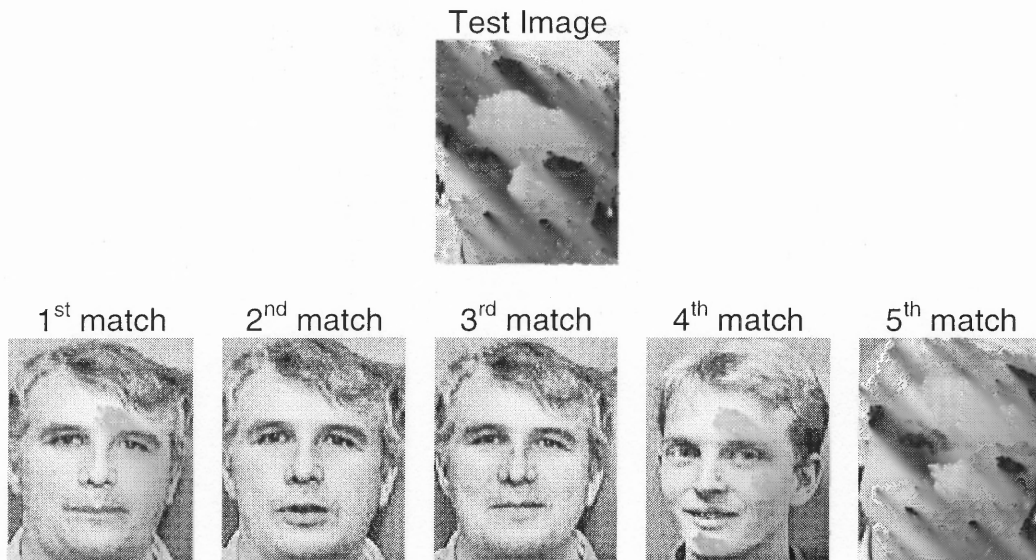


Figure 3.3 Example of incorrect classification using KPCA.

The performance of KPCA on the Yale database was about equal to the Eigenfaces and correlation Algorithms. The random testing strategy showed that the kernel function used, and the number of components used, played little part in the results as most of the error rates quickly converged to 16.7% after ten components. Table 3.1 shows the results for KPCA using the random testing strategy on the Yale database.

Table 3.1 Results of Random Testing on the Yale Database Using KPCA

Comp.	Degree 2	Degree 3	Degree 4	Degree 5	Degree 6	RBF
10	16.7%	18.3%	18.3%	18.3%	23.3%	16.7%
30	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%
40	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%
50	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%
60	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%
80	16.7%	16.7%	15.0%	15.0%	15.0%	16.7%

The results of the leave-one-out strategy on the Yale database are very similar to what they were on the AT&T database. Once again the RBF kernel function had the lowest error rate, with the second-degree polynomial having the second lowest, and then getting worse as the degree increased. Figure 3.4 shows error rates using all the kernel functions and an increasing number of components.

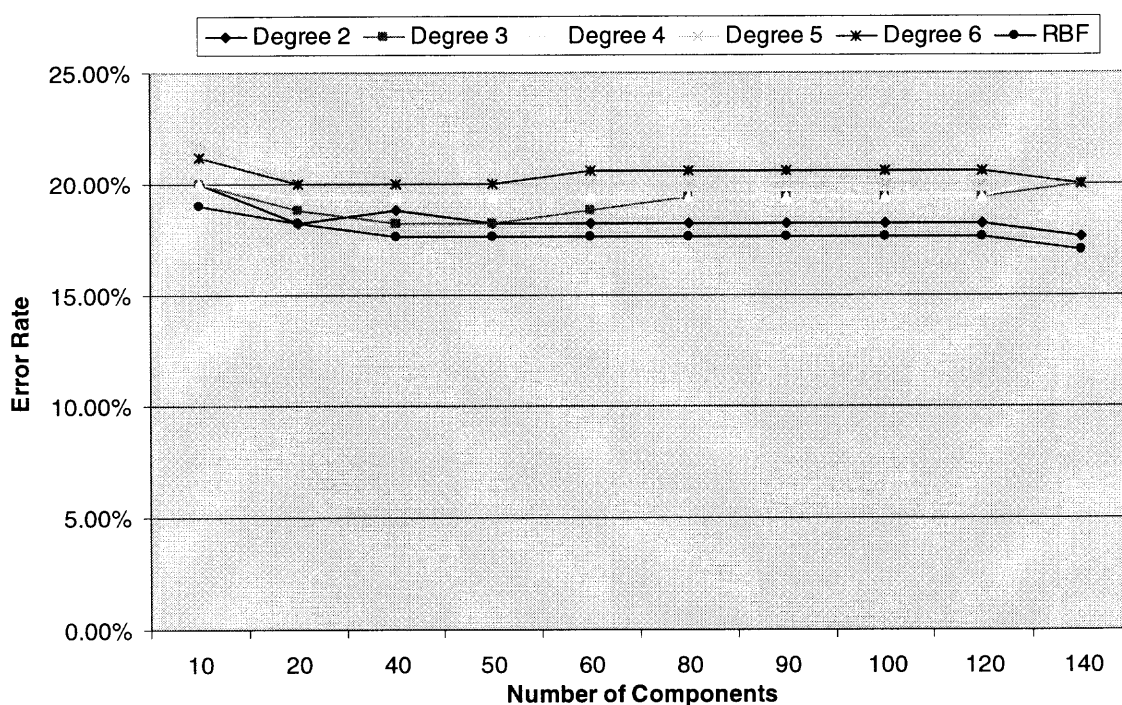


Figure 3.4 Experimental results for KPCA on the Yale database using leave-one-out testing strategy.

Figure 3.5 is an example of lighting causing complications for the KPCA algorithm. Like the Eigenfaces algorithm, KPCA does not deal well with extreme light changes. Examining the results in Figure 3.5, it would seem that the shadow played a large part in classification, as it looks the same in all the images and accounts for a large percentage of the image.

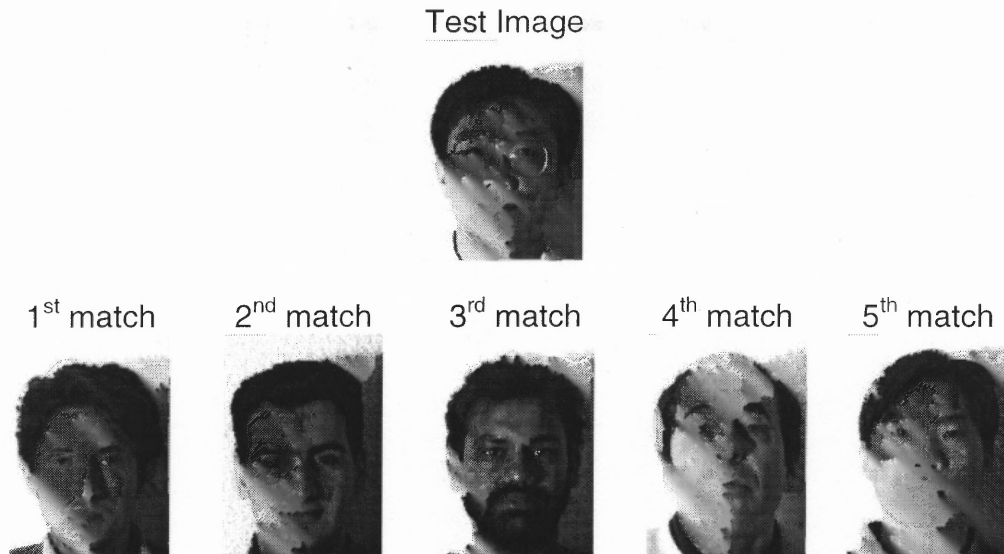


Figure 3.5 Example of incorrect classification using KPCA on the Yale database.

Overall, the testing of the two databases using the two different strategies, KPCA performed about equally with the Eigenfaces method. This may have occurred because with the given training sets, the linear methods were able to capture most of the useful features; thus, the nonlinear extraction was not able to capture any new and useful features and the recognition rates were not any higher. As Figure 3.5 shows, the KPCA algorithm did not handle the lighting very well. This could be partly solved by better cutting of the images to remove more of the background as well as clearing any background pixels.

CHAPTER 4

SUMMARY

Kernel PCA provides a way to extract nonlinear features. This should have led to improved recognition rates over classical linear-based algorithms. Both tests on the AT&T database showed that at best, the KPCA algorithm achieved equal error rates with the Eigenfaces method, but not better. Tables 4.1 and 4.2 show the best results for each of the algorithms on the AT&T database.

Table 4.1 Results of Random Testing on the AT&T Database

Method	Components	Error Rate
Correlation	N/A	8.0%
Eigenfaces	90	8.0%
Fisherfaces	35	9.5%
KPCA-RBF kernel	190	8.5%
KPCA-Degree 2	190	9.0%

Table 4.2 Results of Leave-One-Out Testing on the AT&T Database

Method	Components	Error Rate
Correlation	N/A	2.5%
Eigenfaces	30	1.5%
Fisherfaces	39	2.0%
KPCA-RBF kernel	40	1.5%
KPCA-Degree 2	40	1.75%

In both tests, the lowest error rates were achieved using the Eigenfaces method. Because the AT&T database contains only small variations, the Fisherfaces method was not able to improve on the Eigenfaces method, and in this case, it was slightly worse.

Testing on the Yale database led to the same outcome as the testing on the AT&T database. Once again the KPCA algorithm was only able to achieve equal error rates with its linear counterpart. In the case of the Yale database, it becomes obvious that the Fisherfaces does a much better job of handling more extreme variations in the images, especially changes due to lighting. Tables 4.3 and 4.4 show the best results achieved by each algorithm for the tests done on the Yale database.

Table 4.3 Results of Random Testing on the Yale Database

Method	Components	Error Rate
Correlation	N/A	16.6%
Eigenfaces	20	15.0%
Fisherfaces	10	11.6%
KPCA-RBF kernel	30	16.6%
KPCA-Degree 2	30	16.6%

Table 4.4 Results of Leave-One-Out Testing on the Yale Database

Method	Components	Error Rate
Correlation	N/A	16.9%
Eigenfaces	80	16.9%
Eigenfaces w/o First 3 eigenvectors	130	10.9%
Fisherfaces	20	10.3%
KPCA-RBF kernel	140	16.9%
KPCA-Degree 2	140	17.6%

4.1 Future Work

The databases used in these experiments are both small and contain a large number of faces per subject, which makes the task of recognition easier. They also do not contain any large facial occlusions, such as scarves or hats. Using a larger database with more varied images of each subject might result in different outcomes.

Since Fisherfaces performed the best of all the algorithms on the Yale database, it would make sense to pursue nonlinear generalizations of Fisher's discriminant as described in [6]. Using class knowledge would allow the kernel to extract nonlinear features that better represent the classes.

APPENDIX A

CODE

This appendix contains the code listings required to replicate the results put forth in this paper. Below is an explanation of what each of the included files does.

Eigenfaces_load.m: takes an input training set of images and the number of components to use and returns the highest ranked eigenvectors.

Eigenvectors.m: computes eigenvectors of the given covariance matrix.

Find_nearest.m: finds the training image nearest to the given test image and returns the id number and the distance.

Fisher_load.m: Takes an input of training images and number of components and uses the Fisherfaces algorithm to compute eigenvectors.

Kernel.m: computes kernel representation using given kernel function.

Loo.m: creates leave-one-out test data from a given set of images.

Run_cor.m: runs correlation algorithm and outputs results.

Run_eigenfaces.m: runs the Eigenfaces method and outputs results.

Run_fisher.m: runs Fisherfaces algorithm and output results.

Run_KPCA.m: runs KPCA algorithm and outputs results.

Split_string.m: splits a given string on given delimiter and returns both halves of string.

```

function [psi, vecs, fs] = eigenfaces_load
    (Images, I, vector_length)

%
%
% Purpose: The functions take as parameters the column
%          vector matrix of all the training images, the
%          number of vectors to use and the size of the
%          images.
%
% Inputs:
%   Images: Column vectors of training images
%   I: the number of vectors to use
%   vector_length: size of images.
%
% Outputs:
%   psi: the mean image image of training set.
%   vecs: eigenvectors
%   fs: training images projected into facespace.
%
% get the number of Images
numberofimages = size(Images,2);

% calculate the average face.
psi=mean(Images')';

% allocate space for array A
A = zeros(size(Images,1), numberofimages);

% calculate the difference from the average face for all
images.
for i=1:numberofimages
    A(:,i) = Images(:,i) - psi;
end;

% compute eigenvectors.
vecs = eigenvectors(A, I);

% allocate space for facespace array.
fs = zeros(numberofimages,I);

% calculate all points in I dimension space (facespace).
for i = 1:numberofimages
    fs(i,:) = project_image(vecs, Images(:,i)-psi, I);
end;

```

```

function [Vectors] = eigenvectors(A, numvecs, Psi)
% Purpose: computer eigenvectors by using Turk & pentlands
%          trick to speed up computation.
%
% Inputs:
%   A: images matrix
%   numvecs: vectors to use
%   psi: average matrix
%
% Outputs:
%   Vectors: first numvec eigenvectors
% Based on code by Matthew Dailey 2000

% Check arguments
if nargin ~= 2
    error('usage: eigenvectors(A,numvecs,psi)');
end;

nexamp = size(A,2);

% compute the eigenvectors of the covariance
% matrix,using a little trick from Turk and Pentland
L = A' * A;

% fprintf(1,'Calculating eigenvectors of L...\n');
[Vectors,Values] = eig(L);

% Sort the vectors/values according to size of eigenvalue
[Vectors,Values] = sortem(Vectors,Values);

% Convert the eigenvectors of A'*A into eigenvectors of
A*A'
Vectors = A*Vectors;

% Get the eigenvalues out of the diagonal matrix and
% normalize them so the evalues are specifically for
cov(A'), not A*A'.

Values = diag(Values);
Values = Values / (nexamp-1);

% Normalize Vectors to unit length, kill vectors corr.
% to tiny evalues

```

```

num_good = 0;
for i = 1:nexamp
    Vectors(:,i) = Vectors(:,i)/norm(Vectors(:,i));
    if Values(i) < 0.00001
        % Set the vector to the 0 vector; set the value to 0.
        Values(i) = 0;
        Vectors(:,i) = zeros(size(Vectors,1),1);
    else
        num_good = num_good + 1;
    end;
end;

if (numvecs > num_good)
    fprintf(1,'Warning: numvecs is %d; only %d
exist.\n',numvecs,num_good);

    numvecs = num_good;
end;

Vectors = Vectors(:,1:numvecs);

```



```

function [psi, vecs, fs] = fisher_load
    (Images, I, vector_length, ids)
%
%
% Purpose: calculate eigenvectors via Fisherfaces method.
%
% Inputs:
%     Images: matrix of images
%     I: number of vectors to use
%     Vector_length: size of images
%     ids: labels of images.
%
% Outputs:
%     psi: mean image
%     vecs: eigenvectors
%     fs: images in Facespace
%

% set some variables that will be needed later on.
numofclasses = max(ids);
numofimages = size(Images,2);

% get total scatter for N - c points.
psi = mean(Images')';

% allocate space for array A
A = zeros(size(Images,1), numofimages);

% calculate the difference from the average face for all
% images.
for i=1:numofimages
    A(:,i) = Images(:,i) - psi;
end;

% get the eigen values, vectors & average
vecs_pca = eigenvectors(A, (numofimages - numofclasses));

% allocate space for within class scatter
s_w = zeros(vector_length, numofclasses);
psi_w = zeros(vector_length, numofclasses);

ipc = [];
for i=1:numofclasses
    count = 0;
    temp=0;

```

```

clear('temp');
for j=1:numofimages
    if ids(j) == i,
        count = count + 1;
        temp(:,count) = Images(:,j);
    end
end
psi_w(:,i) = mean(temp)';
ipc(i) = count;
end

x_w = zeros(size(Images,1), size(Images,1));
for i = 1:numofimages
    temp = Images(:,i) - psi_w(:,ids(i));
    x_w = x_w + (temp * temp');
end;
s_w = vecs_pca' * x_w * vecs_pca;

% compute the between class scatter.!
class_psi = mean(psi_w)';
x_b = zeros(size(Images,1), size(Images,1));
for i = 1:numofclasses
    cdiff = (psi_w(:,i) - class_psi);
    temp = ipc(i) * (cdiff * cdiff');
    x_b = x_b + temp;
end;
s_b = vecs_pca' * x_b * vecs_pca;

% compute final eigen vectors based on the scatter
% matrices.
temp = (s_b/s_w);
new_temp = ((temp') * (vecs_pca'));

[vecs,vals,psi_n] = pc_evecs( new_temp , I );
vecs = ((vecs') * (vecs_pca'))';

% calculate the facespace points for all of the training
% images.
fs = zeros(numofimages,I);
for i = 1:numofimages
    fs(i,:) = project_image(vecs,(Images(:,i) - psi),I);
end;

```

```

function [k] = kernel(a, b, kernel_type, kernel_param)
%
% Purpose: computer kernel representation
%
% Inputs
%   a: image vector
%   b: support vector
%   Kernel_type: poly or rbf
%   kernen_param: type specific parameter
%
% Ouputs:
%   k: kernel representation

switch kernel_type
    case 'poly'
        k = ((a' * b)+1).^kernel_param;
    case 'rbf'
        for i=1:size(a,2),
            for j=1:size(b,2),
                k(i,j) =
                    exp(-norm(a(:,i) - b(:,j))^2/kernel_param);
            end
        end
end
end

```

```

function [train,trainids,test,testid]=loo(Images,ids, i)
%
%
% Purpose: create a training set and a testing set
%           by removing the ith values from Images
%           for use by leave on out strategy
%
% Inputs:
%   Images: images matrix
%   ids: image labels
%   i: element to remove
%
% Outputs
%   train: training images
%   trainids: training image labels
%   test: test image
%   test_id: test image id

noi = size(Images, 2);

if i==1,
    train = Images(:, 2:noi);
    trainids = ids(2:noi);
else
    if i==noi,
        train = Images(:, 1:noi-1);
        trainids = ids(1:noi-1);
    else
        train = Images(:, [1:i-1 i+1:noi]);
        trainids = ids([1:i-1 i+1:noi]);
    end
end

test = Images(:,i);
testid = ids(i);

```

```

function run_cor(train_file, test_file, scale)
%
%
% Purpose: run correlation procedure and print out
% the number of errors.
%
% Inputs
%   train_file: listing of files to use for training
%   test_file: images to use for testing
%   scale: scale of images.
%
%

% load training data.
[Images,w,h,ids]=load_training(train_file,scale);
[test_images,w,h,testids]=load_training(test_file, scale);

% get the number of items in the test file.
numberofimages = linecount(test_file);

wrong = 0;
% loop over all the images in the test set.
for i = 1:numberofimages
    test_image = test_images(:,i);
    current_id = testids(i);

    [answer, distance] = find_nearest(Images, test_image);

    if current_id ~= ids(answer)
        wrong = wrong + 1;
        fprintf(1, 'correct id = %d (%d) : ', current_id, i);
        fprintf(1, 'matched id = %d (%d)\n',
            ids(answer), answer);
    end;
end;

% output results
fprintf(1, 'percent incorrect: %d / %d = ', wrong,
numberofimages);
fprintf(1, ' %.2f%%\n', (wrong/numberofimages) * 100);

```

```

function run_eigenfaces(train_file, test_file, I, scale)
%
% Purpose: run Eigenfaces algorithm
%
% Inputs:
%   train_file: list of training files
%   test_file: list of testing images
%   I: number of vectors to use
%   scale: scale of images

% load training data.
[Images,w,h,ids]=load_training(train_file, scale);
[test_images,w,h,test_ids] = load_training(test_file,
scale);

% get the number of items in the test file.
numberofimages = linecount(test_file);

% prepare Eigenfaces and facespace
[psi,ef_vecs,facespace] = eigenfaces_load(Images,I,w*h);

% writejpg(psi, 'ef_half_mean.jpg',w,h,1);

wrong = 0;
% loop over all the images in the test set.
for i = 1:numberofimages
    test_image = test_images(:,i);
    current_id = test_ids(i);

    [answer, distance] = nearest_neighbor
    (facespace,project_image(ef_vecs,(test_image-psi),I) );

    if current_id ~= ids(answer)
        wrong = wrong + 1;
        fprintf(1,'correct id = %d (%d) : ', current_id, i);
        fprintf(1,'matched id = %d (%d)\n',
            ids(answer),answer);
    end;
end;

% output results
fprintf(1, '(vectors used: %d) ', I);
fprintf(1, 'percent incorrect: %d / %d = ', wrong,
    numberofimages);
fprintf(1, '%.2f%%\n', (wrong/numberofimages) * 100);

```

```

function run_fisher(train_file, test_file, I, scale)
%
% Purpose: run Fisherfaces method.
%
% inputs:
% training = name of file containing list of training files
% testing = name of file containing list of testing files
% I = the number of eigen vectors to use.
%

% load training data.
[Images,w,h,ids]=load_training(train_file,scale);
[test_images,w,h,test_ids] =
    load_training(test_file,scale);
numofimages = size(test_images, 2);

[psi,vecs,fs] = fisher_load(Images,I,w*h, ids);

wrong = 0;
% loop over all the images in the test set.
for i = 1:numofimages
    % read in images and reshape it as needed.
    test_image = test_images(:,i);
    current_id = test_ids(i);

    % find the nearest neighbor in facespace.
    [answer, distance, close] = nearest_neighbor
    (fs, project_image(vecs,(test_image-psi),I) );

    % check if answer is correct.
    if current_id ~= ids(answer)
        wrong = wrong + 1;
        fprintf(1,'correct id = %d (%d) : ', current_id, i);
        fprintf(1,'matched id = %d (%d)\n',
            fprintf(1,'%d (%d)\n\n', ids(close(5,2)), close(5,2)));
    end;
end;

% output results
fprintf(1, '(vectors used: %d) ', I);
fprintf(1, 'percent incorrect: %d / %d = ', wrong,
    numofimages);
fprintf(1, '%.2f%%\n', (wrong/(numofimages)) * 100);

```

```

function run_kpca
    (training_file, testing_file, kt, kp, I, scale)
%
% based on code by Bernhard Schölkopf
%
% Purpose: KPCA face recognition.
%
% Inputs:
%   training_file: list of training files
%   testing_file: list of testing files
%   kt: kernel type "poly" or "rbf"
%   kp: kernel parameter
%   I: number of vectors to use
%   scale: scale of images.

% load images.
[patterns,w,h,train_ids]=load_training(training_file,scale)
;
[test_patterns,
w,h,test_ids]=load_training(testing_file,scale);

% number of vectors to use.
ev_start = 0;
ev_end = 200;

% set up some vars.
test_num = size(test_patterns,2);
train_num = size(patterns,2);
max_ev=train_num-1;

% set up the size
cov_size = train_num;

%*****
%*****
% carry out Kernel PCA
% this checks out with other code.
%*****
%*****

K=kernel(patterns, patterns,kt, kp);

% set up unit vector that is NxN/N
unit = ones(cov_size, cov_size)/cov_size;

```



```

% centering in feature space!
K_n = K - unit*K - K*unit + unit*K*unit;

% solve for eigen values
[vecs,evals] = eig(K_n/cov_size);
[vecs,evals] = sortem(vecs,real(evals));
evals = real(diag(evals));

% normalize the vector
for i=1:max_ev,
    vecs(:,i) = vecs(:,i)/sqrt(evals(i));
end

% for max_ev=ev_start:ev_end,
max_ev=I;
%*****
% feature Projections
%*****

% unit vector
unit_train = ones(train_num,cov_size)/cov_size;

% allocate space for K
K_train = zeros(train_num,cov_size);

% run kernel function
K_train = kernel(patterns, patterns,kt, kp);

% center the data.
K_train_n = K_train - unit_train*K - K_train*unit +
unit_train*K*unit;

% allocate space
features = zeros(train_num, max_ev);

% project
features = K_train_n * vecs(:,1:max_ev);

% unit vector
unit_test = ones(test_num,cov_size)/cov_size;

```

```

% allocate space for K
K_test = zeros(test_num,cov_size);

% K_test = (test_patterns)' * patterns;
K_test=kernel(test_patterns, patterns,kt, kp);

% center the data.
K_test_n = K_test - unit_test*K - K_test*unit +
unit_test*K*unit;

% allocate space for K
test_features = zeros(test_num, max_ev);

% project
test_features = K_test_n * evecs(:,1:max_ev);

% allocate space for the arrays.
images=zeros(max_ev,train_num);
test_images=zeros(max_ev,test_num);

% reshape the features
images = features';
test_images = test_features';

% use nearest neighbor to find match.
count = 0;
for i=1:test_num,
    [answer,loc] = find_nearest(images,test_images(:,i));
    % fprintf(1,'answer=%d\n', loc);
    if test_ids(i) ~= train_ids(answer),
        count = count + 1;
        fprintf(1,'%d(%d) = %d(%d)\n', test_ids(i),i,
train_ids(answer),answer);
    end
end

fprintf(1, '(vectors used: %d) ', max_ev);
fprintf(1, 'percent incorrect: %d / %d = ', count,
train_num);
fprintf(1, '%.2f%%\n', (count/train_num) * 100);
fprintf(1, '%d,%.2f\n', max_ev, (count/test_num) * 100);

```

```

function [firsthalf, secondhalf]=
    split_string(p_string,p_delim)
%
% purpose: split a string into two halves
% based on the delimiter.
%
% Inputs:
%     p_string: string to be split
%     p_delim: delimiter
%
% Outputs:
%     firsthalf: string before delimiter
%     secondhalf: string after delimiter

% find the location of the delimiter.
n_delim = strfind(p_string,p_delim);

% make sure that the delimiter was found
if isempty(n_delim)
    error('Delimiter was not found in string');
end;

firsthalf = p_string(1:n_delim-1);
secondhalf = p_string(n_delim+1:length(p_string));

```

REFERENCES

- [1] M. Turk and A. Pentland, "Face Recognition Using Eigenfaces," *Proc. IEEE Conf. On Computer Vision and Pattern Recognition*, 1991, pp. 586-591.
- [2] P. Belhumeur and J. Hespanha and D. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection", *IEEE transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, July, 1997, pp. 711-720
- [3] B. Schölkopf, A. Smola, and K.-R. Müller. "Nonlinear component analysis as a kernel eigenvector problem" *Neural Computation*, 10 (5): pp. 1299-1319, 1998.
- [4] Yale Face Database. Yale University. November, 2002.
<<http://cvc.yale.edu/projects/yalefaces/yalefaces.html>>.
- [5] The Database of Faces. AT&T Laboratories Cambridge. October, 2002.
<<http://www.uk.research.att.com/facedatabase.html>>.
- [6] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, A. Smola, and K-R. Müller. "Invariant feature extraction and classification in kernel spaces". In S. Solla, T. Leen, K-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pp. 526-532. MIT Press, 2000.
- [7] D. McGuire. "Virginia Beach Installs Face-Recognition Cameras." *Washington Post*. July 3rd, 2002.
- [8] ACLU Opposes Use of Face Recognition Software in Airports Due to Ineffectiveness and Privacy Concerns. ACLU Website. July 1st, 2003. <<http://archive.aclu.org/features/f110101a.html>>.
- [9] J. Stanley and B. Steinhardt. "Drawing a Blank: Failure of facial Recognition technology in Tampa, Florida" January 3rd, 2002.
<http://archive.aclu.org/issues/privacy/drawing_blank.pdf>.
- [10] Y. Moses, Y. Adini, and S. Ullman, "Face Recognition: The Problem of Compensating for Changes in Illumination Direction," *European Conference Of Computer Vision*, 1994, pp. 286-296.