Human Face Recognition Technology Using the Karhunen-Loéve Expansion Technique

Anthony Giordano & Michael Uhrig
Regis University
Denver, Colorado

Abstract

We will explore the area of face recognition using the partial singular value decomposition of a matrix and test some of its successes and limitations. We constructed a database consisting of 130 pictures of 65 individuals, and then used the Karhunen-Loéve (KL) Expansion method to relate pictures from outside the database to those in the database. While this method was generally very successful, we were able to test and define several of its limitations.

1 Introduction

The Karhunen-Loéve (KL) Expansion technique for face recognition is a widely used method using a portion of the singular value decomposition of a matrix from linear algebra. Turk and Pentland in 1991 were the first to conduct significant tests using this technique in the area of face recognition [1], however Sirovich and Kirby used this technique as early as 1987 [2]. Using a large database the authors were able to test the robustness of the system over variations in lighting, orientation of the head, and size [3] and [4]. The goal of this project was to construct a system in *MATLAB* using the KL expansion to further test potential limitations of this technique.

The KL expansion relies on our Proposition 2.8, which states that given a matrix A, the first j eigenvectors corresponding to the largest j eigenvalues of the covariance matrix of A provide the best j-dimensional approximation to the rows of A [5]. The KL expansion then takes each image in the database (all of which must have the same pixel dimensions) and reshapes them so that each image corresponds to a row of a very large matrix A. Using our theorem we can then create a j-dimensional approximation by orthogonally projecting each picture vector onto the space spanned by the first j eigenvectors. The pictures outside the database are then reshaped in the same way and projected onto the same subspace. At this point, it is a simple matter of taking the Euclidean distance between the coefficients of the orthogonal projections of these outside pictures and the images inside the database and finding which picture is the closest match.

Using *MATLAB*, we were able to automate the importing of images to form the database, reshape the images, and construct our j-dimensional approximation. By taking multiple pictures of each person, we were able to test the robustness of the system to changes in the pictures, such as presence of glasses, hairstyle, facial expression, and head tilt. We found that the presence or lack of glasses in the pictures had little to no effect on the results. Hairstyle,

however, was very influential on the results, in fact in our first database the presence or lack of long hair was determined to be the most influential characteristic in the recognition process. Contorted facial expressions and a head tilt of $\pm 5^\circ$ or more also had a significantly negative impact on the results.

2 Linear Algebra Applications to Face Recognition

The KL Expansion relies on properties of the Singular-Value Decomposition (SVD) of a matrix. The SVD of any matrix A provides us with a way to decompose that matrix to the form

$$A = U\Sigma V^T$$

where if A is $p \times n$ then U is a $p \times p$ orthogonal matrix whose columns provide a basis for the columnspace of A and V is an $n \times n$ matrix whose columns provide a basis for the rowspace of A. For our purposes only a portion of the SVD will be used.

We should note here that the following propositions are all well established already. However, for the sake of completeness and to provide the reader with a greater understanding of this area we decided to include all of the proofs. Most of the following proofs can be found in [7], although for proof of Proposition 2.5 a good source would be [6]. The purpose of the first six propositions is to establish certain properties of the $n \times n$ matrix $A^T A$, namely that it has a set of eigenvectors which form a basis for $\mathbb{R}^n$. The way that we will prove this is by showing that this matrix is diagonalizable. This will suffice as it is a commonly known fact from linear algebra that a diagonal matrix has a set of eigenvectors which form a basis for $\mathbb{R}^n$.

Proposition 2.1. *Given any matrix A, the matrix $A^T A$ is symmetric.*

Proof. Define the matrix A as the set of vectors

$$A = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{pmatrix}$$

and

$$A^T = \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{pmatrix}.$$

Then the ij^{th} entry of $A^T A$ is

$$\mathbf{v}_i^T \mathbf{v}_j = \mathbf{v}_i \cdot \mathbf{v}_j.$$

Similarly, the ji^{th} entry is

$$\mathbf{v}_j^T \mathbf{v}_i = \mathbf{v}_j \cdot \mathbf{v}_i.$$

Since the dot product of two vectors is commutative, $\mathbf{v}_i \cdot \mathbf{v}_j = \mathbf{v}_j \cdot \mathbf{v}_i$. □

Proposition 2.2. *For any symmetric matrix A, the eigenvectors from different eigenspaces are always orthogonal.*

Proof. Let $\mathbf{x}_a$ and $\mathbf{x}_b$ be two eigenvectors of A corresponding to two distinct eigenvalues λ_a and λ_b, respectively. Then,

$$\begin{aligned}
&\lambda_a \mathbf{x}_a \cdot \mathbf{x}_b \\
&= (A\mathbf{x}_a)^T \mathbf{x}_b \\
&= \mathbf{x}_a^T A^T \mathbf{x}_b \\
&= \mathbf{x}_a^T A \mathbf{x}_b \\
&= \mathbf{x}_a^T \lambda_b \mathbf{x}_b \\
&\lambda_a \mathbf{x}_a \cdot \mathbf{x}_b = \lambda_b \mathbf{x}_a \cdot \mathbf{x}_b \\
&(\lambda_a - \lambda_b)\mathbf{x}_a \cdot \mathbf{x}_b = 0.
\end{aligned}$$

Since λ_a and λ_b are distinct, $\lambda_a - \lambda_b \neq 0$ and $\mathbf{x}_a \cdot \mathbf{x}_b$ must equal zero. Therefore, the vectors $\mathbf{x}_a$ and $\mathbf{x}_b$ are orthogonal.

□

Proposition 2.3. *The eigenvalues of $A^T A$ are all non-negative and real.*

Proof. Let $\mathbf{x}_a$ be a normalized eigenvector of $A^T A$ and λ_a be the corresponding eigenvalue. Then,

$$\begin{aligned}
&\|A\mathbf{x}_a\|^2 \\
&= (A\mathbf{x}_a)^T (A\mathbf{x}_a) \\
&= \mathbf{x}_a^T A^T A \mathbf{x}_a \\
&= \mathbf{x}_a^T (\lambda_a \mathbf{x}_a) \\
&\|A\mathbf{x}_a\|^2 = \lambda_a.
\end{aligned}$$

Therefore, the eigenvalues (λ_a) that are produced by $A^T A$ are all non-negative and real since the length of a vector squared is always nonnegative and real.

□

Definition 2.4. The **singular values** of a matrix A are equal to the length of $A\mathbf{x}$, where $\mathbf{x}$ is an eigenvector of $A^T A$. The set of singular values is denoted by $\sigma_1, \ldots, \sigma_n$. From Proposition 2.3 we also know that the singular values are equivalent to the square root of the eigenvalues of $A^T A$.

For the remainder of this paper it will be assumed that the singular values are arranged such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. It will also be assumed that the corresponding eigenvalues are arranged in decreasing order as well, such that $\lambda_a = {\sigma_a}^2$.

Proposition 2.5. *If $A(n \times n)$ has n real eigenvalues then there exists a matrix U with orthonormal columns such that $U^T AU$ is upper triangular.*

Proof. Since A has n real eigenvalues $(\lambda_1 \dots \lambda_n)$ we know that each eigenspace has at least one corresponding eigenvector. Take any one of these normalized vectors and let it be $\mathbf{v}_1$ with corresponding eigenvalue λ_1. It is possible to use this vector to create an orthonormal basis for $\mathbb{R}^n$. To do this, create $n-1$ vectors that are linearly independent of $\mathbf{v}_1$ and use the Gram-Schmidt process to create an orthonormal basis for $\mathbb{R}^n$. Call this set of new vectors $\{\mathbf{w}_1 \dots \mathbf{w}_{n-1}\}$, and use this set of vectors in tandem with the eigenvector of A to create the following $n \times n$ matrix.

$$U_1 = \begin{pmatrix} \mathbf{v}_1 & \mathbf{w}_1 & \cdots & \mathbf{w}_{n-1} \end{pmatrix}$$

For notation purposes it is also necessary to define the set of $\mathbf{w}$ vectors.

$$W = \begin{pmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_{n-1} \end{pmatrix}$$

The initial step in the formation of an upper triangular matrix is to multiply A and U_1 together to produce the matrix B, such that $B = U_1^T A U_1$.

$$\begin{aligned}
B &= \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_{n-1}^T \end{pmatrix} \begin{pmatrix} A \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 & \mathbf{w}_1 & \cdots & \mathbf{w}_{n-1} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_{n-1}^T \end{pmatrix} \begin{pmatrix} A\mathbf{v}_1 & A\mathbf{w}_1 & \cdots & A\mathbf{w}_{n-1} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_{n-1}^T \end{pmatrix} \begin{pmatrix} \lambda_1\mathbf{v}_1 & A\mathbf{w}_1 & \cdots & A\mathbf{w}_{n-1} \end{pmatrix} \\
&= \left(\begin{array}{c|ccc} \lambda_1\mathbf{v}_1 \cdot \mathbf{v}_1 & \mathbf{v}_1 \cdot (A\mathbf{w}_1) & \cdots & \mathbf{v}_1 \cdot (A\mathbf{w}_{n-1}) \\ \hline \lambda_1\mathbf{w}_1 \cdot \mathbf{v}_1 & & & \\ \vdots & & W^T AW & \\ \lambda_1\mathbf{w}_{n-1} \cdot \mathbf{v}_1 & & & \end{array}\right)
\end{aligned}$$

Since $\mathbf{v}_1$ is a normal vector and is orthogonal to the columns of W we can simplify this matrix to

$$B = \left(\begin{array}{c|c} \lambda_1 & \mathbf{v}_1^T AW \\ \hline \mathbf{0} & W^T AW \end{array}\right).$$

At this point it is important to note that the matrix B has the same eigenvalues as the original matrix A, this can be easily proven through the use of the characteristic polynomial.

$$\begin{aligned}
\text{char}(U_1^T A U_1) &= \det(U_1^T A U_1 - \lambda I) \\
&= \det(U_1^T A U_1 - U_1^T \lambda U_1) \\
&= \det(U_1^T (A - \lambda I) U_1) \\
&= \det(U_1^T) \det(A - \lambda I) \det(U_1) \\
&= \det(U_1^T U_1) \det(A - \lambda I) \\
&= \det(I) \det(A - \lambda I) \\
&= \det(A - \lambda I) \\
&= \text{char}(A)
\end{aligned}$$

Since B has the same eigenvalues as A $(\lambda_1 \dots \lambda_n)$ then the lower right partition of B, the $n-1 \times n-1$ matrix $W^T A W$, has $n-1$ eigenvalues $(\lambda_2 \dots \lambda_n)$. Now define $\mathbf{v}_2$ as being the normalized eigenvector of $W^T A W$, which corresponds to the eigenvalue λ_2. This vector is used to create an orthonormal basis for $\mathbb{R}^{n-1}$, $\{\mathbf{v}_2, \mathbf{y}_1, \dots, \mathbf{y}_{n-2}\}$. Use this set of vectors to create the matrix U_2.

$$U_2 = \left(\begin{array}{c|ccc} 1 & \multicolumn{3}{c}{\mathbf{0}} \\ \hline \mathbf{0} & \mathbf{v}_2 & \mathbf{y}_1 \cdots & \mathbf{y}_{n-2} \end{array}\right)$$

It will also become convenient later to define the $n-1 \times n-2$ matrix Y whose columns are the set $\{\mathbf{y}_1, \dots, \mathbf{y}_{n-2}\}$. Multiply U_2 and B in the same manner as previously conducted, which provided us with B.

$$\begin{aligned}
U_2^T B U_2 &= \left(\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{0} & \begin{array}{c} \mathbf{v}_2^T \\ \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_{n-2}^T \end{array} \end{array}\right) \left(\begin{array}{c|c} \lambda_1 & \mathbf{v}_1^T A W \\ \hline \mathbf{0} & W^T A W \end{array}\right) \left(\begin{array}{c|ccc} 1 & \multicolumn{3}{c}{\mathbf{0}} \\ \hline \mathbf{0} & \mathbf{v}_2 & \mathbf{y}_1 \cdots & \mathbf{y}_{n-2} \end{array}\right) \\
&= \left(\begin{array}{c|c} \lambda_1 & \mathbf{v}_1^T A W \\ \hline \mathbf{0} & \begin{bmatrix} \mathbf{v}_2^T \\ \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_{n-2}^T \end{bmatrix} W^T A W \end{array}\right) \left(\begin{array}{c|ccc} 1 & \multicolumn{3}{c}{\mathbf{0}} \\ \hline \mathbf{0} & \mathbf{v}_2 & \mathbf{y}_1 \cdots & \mathbf{y}_{n-2} \end{array}\right)
\end{aligned}$$

$$= \left(\begin{array}{c|c} \lambda_1 & \mathbf{v}_1^T AW \\ \hline \mathbf{0} & \begin{array}{c} \lambda_2 \mathbf{v}_2^T \\ Y^T W^T AW \end{array} \end{array}\right) \left(\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{v}_2 \quad \mathbf{y}_1 \cdots \quad \mathbf{y}_{n-2} \end{array}\right)$$

$$= \left(\begin{array}{c|cc} \lambda_1 & \mathbf{v}_1^T AW\mathbf{v}_2 & \mathbf{v}_1^T AWY \\ \hline \mathbf{0} & \lambda_2 & \lambda_2 \mathbf{v}_2^T Y \\ & Y^T \lambda_2 \mathbf{v}_2 & Y^T W^T AWY \end{array}\right)$$

$$= \left(\begin{array}{c|cc} \lambda_1 & \mathbf{v}_1^T AW\mathbf{v}_2 & \mathbf{v}_1^T AWY \\ \hline \mathbf{0} & \lambda_2 & \lambda_2 \mathbf{v}_2^T Y \\ & \mathbf{0} & Y^T W^T AWY \end{array}\right)$$

The bottom right partition of this result $Y^T W^T AWY$ has eigenvalues $\lambda_3, \ldots, \lambda_n$ Using this knowledge, the same process can be repeated once again, using $\mathbf{v}_3$, the eigenvector of $Y^T W^T AWY$ to create an orthonormal basis for $\mathbb{R}^{n-2}$. This matrix will be placed in the bottom right partition of an $n \times n$ matrix U_3, which will be filled in with 1's along the two blank diagonal entries and zeroes in all other blank entries. If this process is repeated n times, then the result will be in the form

$$U_n^T \ldots U_1^T AU_1 \ldots U_n.$$

Since all of the U_i matrices are orthonormal, the results of the matrix multiplication $U_1 \ldots U_n$ and $U_n^T \ldots U_1^T$ are orthonormal. Furthermore, if we define $U = U_1 \ldots U_n$, then $U^T = U_n^T \ldots U_1^T$. Therefore, U is in the form that we desire, and the matrix multiplication $U^T AU$ yields an upper triangular matrix with the eigenvalues of A in the diagonal entries. □

Proposition 2.6. *If A is symmetric then $U^T AU$ is symmetric.*

Proof. To prove $U^T AU$ is symmetric, we simply need to prove that $(U^T AU)^T = U^T AU$.

$$\begin{aligned} (U^T AU)^T &= U^T A^T U^{T^T} \\ &= U^T AU \end{aligned}$$

□

An easy corollary to the last two propositions is that any symmetric matrix A is diagnolizable, since a symmetric, upper triangular matrix will necessarily be diagonal.

In the next proposition we will use the preceding conclusions, which established a basis for $\mathbb{R}^n$ from the eigenvectors of $A^T A$, to derive a basis for the columnspace of A. Before we begin the next section, recall that the columnspace of a $p \times n$ matrix A, denoted $\operatorname{Col} A$, is defined as the subspace of $\mathbb{R}^p$ spanned by the columns of A. Similarly, the rowspace of A, denoted $\operatorname{Row} A$, is the subspace of $\mathbb{R}^n$ spanned by the rows of A.

Proposition 2.7. *Define $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ as an orthonormal basis for $\mathbb{R}^n$ consisting of eigenvectors of A^TA. These vectors also correspond to the eigenvalues of A^TA; $\lambda_1, \lambda_2, \ldots, \lambda_n$. Now, suppose that A^TA has r nonzero eigenvalues, then the set of vectors $\{A\mathbf{x}_1, A\mathbf{x}_2, \ldots, A\mathbf{x}_r\}$ is an orthogonal basis for the columnspace of A.*

Proof. For any $a \neq b$, $\mathbf{x}_a$ is orthogonal to $\mathbf{x}_b$.

$$(A\mathbf{x}_a) \cdot (A\mathbf{x}_b) = (A\mathbf{x}_a)^T(A\mathbf{x}_b) = \mathbf{x}_a^T A^T A\mathbf{x}_b = \mathbf{x}_a^T \lambda_b \mathbf{x}_b = \lambda_b \mathbf{x}_a \cdot \mathbf{x}_b = 0$$

Therefore, $\{A\mathbf{x}_1, A\mathbf{x}_2, \ldots, A\mathbf{x}_r\}$ is an orthogonal set. From Proposition 2.3 the eigenvalues of A^TA are equivalent to the lengths of the vectors squared. Since there are r nonzero eigenvalues, $A\mathbf{x}_a \neq 0$ if and only if $1 \leq a \leq r$, which means that $\{A\mathbf{x}_1, A\mathbf{x}_2, \ldots, A\mathbf{x}_r\}$ is a linearly independent set in Col A. Any vector $\mathbf{z}$ in Col A can be written as $\mathbf{z} = A\mathbf{v}$ and $\mathbf{v}$ can be written as a linear combination of $\mathbf{x}_1, \ldots \mathbf{x}_n$ with coefficients $c_1 \ldots c_n$.

$$\mathbf{z} = A\mathbf{v} = Ac_1\mathbf{x}_1 + \cdots + Ac_r\mathbf{x}_r + \cdots + Ac_n\mathbf{x}_n$$

$$= Ac_1\mathbf{x}_1 + \cdots + Ac_r\mathbf{x}_r + 0 + \cdots + 0$$

Therefore, $\mathbf{z}$ is in Span $\{A\mathbf{x}_1, A\mathbf{x}_2, \ldots, A\mathbf{x}_r\}$, which means that this set is a basis for Col A. □

Finally we are in a position to understand how the Karhunen-Loéve method will work. In the interest of clarity, let's first consider a very simple example of how the method works.

EXAMPLE

Consider the matrix

$$Q = \begin{pmatrix} 2 & 3 & 5 & 6 \\ 5 & 7 & 8 & 10 \end{pmatrix}.$$

In this case, Col Q is simply $\mathbb{R}^2$, and the vectors that form the columns can actually be plotted, as in Figure 1.

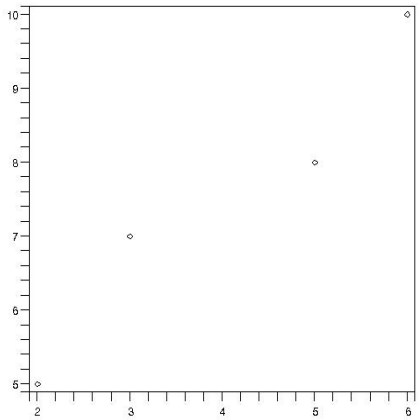

Figure 1: Vectors formed by the Columns of Q Plotted as Points in the Euclidean Plane

At this point, we can take the mean vector

$$M = \frac{1}{4}\begin{pmatrix} 2+3+5+6 \\ 5+7+8+10 \end{pmatrix} = \begin{pmatrix} 4 \\ 7.5 \end{pmatrix}$$

and subtract it from each of the columns of Q to obtain our new mean-subtracted matrix,

$$Qsub = \begin{pmatrix} -2 & -1 & 1 & 2 \\ -2.5 & -.5 & .5 & 2.5 \end{pmatrix}$$

the plot of which is now centered around the origin. Now we can calculate the small matrix

$$Qsub * Qsub^T = \begin{pmatrix} 10 & 11 \\ 11 & 13 \end{pmatrix}.$$

If we calculate the eigenvalues and corresponding eigenvectors, we get $\{22.6, .398\}$ for the eigenvalues and

$$\begin{pmatrix} .6576 \\ .7534 \end{pmatrix}, \begin{pmatrix} .7534 \\ -.6576 \end{pmatrix}$$

for the corresponding eigenvectors. If we view the graph of the space spanned by the first eigenvector corresponding to the larger eigenvalue, then we see it creates a nice "line of best fit" for the points in $Qsub$ (see Figure 2).

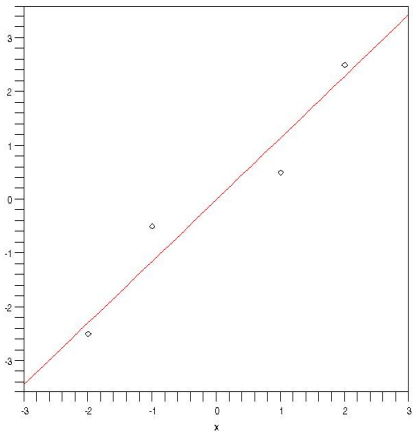

Figure 2: The Columns of $Qsub$ Plotted as Points with the One-Dimensional Subspace Spanned by the First Eigenvector of $Qsub * Qsub^T$

In fact, what the following proposition will establish is that this vector spans the best one-dimensional approximation of the data that we can find. For the purpose of maintaining consistency with the literature, we will define the matrix $Qsub * Qsub^T$ as the covariance matrix of $Qsub$, though $Qsub^T Qsub$ is also given the same title. It is important to note that in the literature, it is much more typical to define the covariance matrix as $\frac{1}{p}Qsub^T Qsub$ or $\frac{1}{n}Qsub * Qsub^T$. However, for our purposes the presence of the fraction at the beginning is unimportant, as it does not affect the eigenvectors and only scales the eigenvalues by $\frac{1}{p}$ or $\frac{1}{n}$.

Therefore, leaving it out will not effect our final result and in the interest of simplicity we have left it out.

Furthermore, in the proposition we contend that this method provides the "best" j-dimensional approximation of the rowspace of a matrix. While the word "best" may seem imprecise at first, it actually means we are minimizing the orthogonally projected distance. So in our previous example, if all of our four points were projected onto the one-dimensional subspace spanned by the vector $\begin{pmatrix} .6576 \\ .7534 \end{pmatrix}$, then total of the distances that each point would have to travel to get to that line would be the smallest possible distance for any one-dimensional subspace. We must also note that although we used the term "line of best fit" earlier, this was simply an analogous comparison. The process we are using minimizes the orthogonal distance, whereas the least squares line of best fit from elementary statistics minimizes the vertical distance.

Proposition 2.8. *Given any $p \times n$ matrix X, define its covariance matrix $C = X^T X$. If the eigenvalues of C are $(\lambda_1, \ldots, \lambda_n)$ with corresponding eigenvectors are $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$, then the first j eigenvectors of C provide the best j-dimensional approximation to the rows of X iff $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.*

Proof. Given a matrix X, we name the rows $\{\mathbf{x}_1, \ldots, \mathbf{x}_p\}$ and write

$$X = \begin{pmatrix} & & \\ \mathbf{x}_1 & \cdots & \mathbf{x}_p \\ & & \end{pmatrix}^T.$$

Every $\mathbf{x}_i$ is a vector in $\mathbb{R}^n$. Additionally, define Φ as the matrix whose columns consist of the set of vectors $\{\phi_1, \ldots, \phi_n\}$ which form an orthonormal basis for $\mathbb{R}^n$. Therefore, any $\mathbf{x}_i$ is contained within the span of Φ and can be written as a linear combination of the columns:

$$\mathbf{x}_i = \sum_{k=1}^{n} \alpha_k \phi_k.$$

Since the $\phi' s$ form an orthonormal basis for $\mathbb{R}^n$, the coefficients α_k are easy to calculate;

$$\alpha_k = \mathbf{x}_i \cdot \phi_k = \langle \mathbf{x}_i, \phi_k \rangle.$$

Although the matrix Φ provides the full orthonormal basis for $\mathbb{R}^n$, in our application we will not be using the full basis. If we use, for example, only the first j vectors in Φ, then it is possible to split up the summation and write it as

$$\mathbf{x}_i = \sum_{k=1}^{j} \alpha_k \phi_k + \sum_{k=j+1}^{n} \alpha_k \phi_k.$$

When it is written this way, the first term $\left(\sum_{k=1}^{j} \alpha_k \phi_k\right)$ is the j-dimensional approximation of $\mathbf{x}_i$ and the second term can be thought of as the error. For notation purposes, define

$$\mathbf{x}_i^{(err)} = \sum_{k=j+1}^{n} \alpha_k \phi_k.$$

At this point we can also define the total error of the orthogonal projection of X onto the linear subspace of $\mathbb{R}^n$ spanned by the first j vectors of Φ.

$$\begin{aligned}
\text{Error} &= \sum_{t=1}^{p} \parallel \mathbf{x}_t^{(err)} \parallel^2 \\
&= \sum_{t=1}^{p} \parallel \left(\sum_{k=j+1}^{n} \alpha_k \phi_k \right) \left(\sum_{k=j+1}^{n} \alpha_k \phi_k \right) \parallel \\
&= \sum_{t=1}^{p} \parallel \left(\alpha_{j+1}\phi_{j+1} + \cdots + \alpha_n \phi_n \right) \left(\alpha_{j+1}\phi_{j+1} + \cdots + \alpha_n \phi_n \right) \parallel
\end{aligned}$$

Since $\phi_m \cdot \phi_n = 0$ for all $m \neq n$ and $\phi_m \cdot \phi_m = 1$ for all m, the previous equation can be simplified.

$$\begin{aligned}
\text{Error} &= \sum_{t=1}^{p} \parallel \left(\alpha_{j+1}^2 + \cdots + \alpha_n^2 \right) \parallel \\
&= \sum_{t=1}^{p} \left(\sum_{k=j+1}^{n} \left(\alpha_k \right)^2 \right) \\
&= \sum_{t=1}^{p} \left(\sum_{k=j+1}^{n} \left(\mathbf{x}_t \cdot \phi_k \right)^2 \right) \\
&= \sum_{t=1}^{p} \left(\sum_{k=j+1}^{n} \left(\mathbf{x}_t \cdot \phi_k \right) \left(\mathbf{x}_t \cdot \phi_k \right) \right) \\
&= \sum_{t=1}^{p} \left(\sum_{k=j+1}^{n} \left(\phi_k \cdot \mathbf{x}_t \right) \left(\mathbf{x}_t \cdot \phi_k \right) \right) \\
&= \sum_{t=1}^{p} \left(\sum_{k=j+1}^{n} \phi_k^T \mathbf{x}_t \mathbf{x}_t^T \phi_k \right)
\end{aligned}$$

Since both of the summation functions are finite sums, it is possible to commute the order.

$$\begin{aligned}
\text{Error} &= \sum_{k=j+1}^{n} \left(\sum_{t=1}^{p} \phi_k^T \mathbf{x}_t \mathbf{x}_t^T \phi_k \right) \\
&= \sum_{k=j+1}^{n} \left(\left(\phi_k^T \mathbf{x}_1 \mathbf{x}_1^T \phi_k + \cdots + \phi_k^T \mathbf{x}_p \mathbf{x}_p^T \phi_k \right) \right) \\
&= \sum_{k=j+1}^{n} \left(\phi_k^T \left(\left(\mathbf{x}_1 \mathbf{x}_1^T + \cdots + \mathbf{x}_p \mathbf{x}_p^T \right) \right) \phi_k \right) \\
&= \sum_{k=j+1}^{n} \langle \phi_k, \left(\sum_{t=1}^{p} \mathbf{x}_t \mathbf{x}_t^T \right) \phi_k \rangle
\end{aligned}$$

In order to proceed at this point, it is necessary to define some new notation. $\psi_t^{(a)}$ is equal to the a^{th} entry in the vector $\mathbf{x}_t$. Therefore,

$$\begin{aligned}
\sum_{t=1}^{p} \mathbf{x}_t \mathbf{x}_t^T &= \begin{pmatrix} \psi_1^{(1)}\psi_1^{(1)} & \cdots & \psi_1^{(1)}\psi_1^{(n)} \\ \vdots & \ddots & \vdots \\ \psi_1^{(n)}\psi_1^{(1)} & \cdots & \psi_1^{(n)}\psi_1^{(n)} \end{pmatrix} + \cdots + \begin{pmatrix} \psi_p^{(1)}\psi_p^{(1)} & \cdots & \psi_p^{(1)}\psi_p^{(n)} \\ \vdots & \ddots & \vdots \\ \psi_p^{(n)}\psi_p^{(1)} & \cdots & \psi_p^{(n)}\psi_p^{(n)} \end{pmatrix} \\
&= \begin{pmatrix} \psi_1^{(1)}\psi_1^{(1)} + \ldots + \psi_p^{(1)}\psi_p^{(1)} & \cdots & \psi_1^{(1)}\psi_1^{(n)} + \ldots + \psi_p^{(1)}\psi_p^{(n)} \\ \vdots & \ddots & \vdots \\ \psi_1^{(n)}\psi_1^{(1)} + \ldots + \psi_p^{(n)}\psi_p^{(1)} & \cdots & \psi_1^{(n)}\psi_1^{(n)} + \ldots + \psi_p^{(n)}\psi_p^{(n)} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_p \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_p \end{pmatrix}^T \\
&= X^T X.
\end{aligned}$$

Substituting this into the equation for the error yields

$$\begin{aligned}
\text{Error} &= \sum_{k=j+1}^{n} \langle \phi_k, \left(\sum_{t=1}^{p} \mathbf{x}_t \mathbf{x}_t^T \right) \phi_k \rangle \\
&= \sum_{k=j+1}^{n} \langle \phi_k, \left(X^T X \right) \phi_k \rangle \\
&= \sum_{k=j+1}^{n} \langle \phi_k, C\phi_k \rangle.
\end{aligned}$$

This provides a simple way to look at the error produced by the j-dimensional approximation of X. Now let's just assume for a moment that $j = 0$. In this case, the error can be expressed as

$$\text{Error} = \sum_{t=1}^{p} \|\mathbf{x}_t^{(err)}\|^2$$

and $\mathbf{x}_t^{(err)}$ can be expressed as

$$\mathbf{x}_t^{(err)} = \sum_{k=1}^{n} \alpha_k \phi_k = \mathbf{x}_t.$$

Since $\mathbf{x}_t^{(err)} = \mathbf{x}_t$ when $j = 0$, we can substitute into the previous equation to obtain

$$\text{Error} = \sum_{t=1}^{p} \|\mathbf{x}_t\|^2 = \text{Constant}.$$

This is the same as saying

$$\sum_{k=1}^{n} \langle \phi_k, C\phi_k \rangle$$

is also a constant and can be split up in the following manner:

$$\langle \phi_1, C\phi_1 \rangle + \sum_{k=2}^{n} \langle \phi_k, C\phi_k \rangle.$$

Notice now that the second term ($\sum_{k=2}^{n} \langle \phi_k, C\phi_k \rangle$) is equal to the error for the 1-dimensional approximation of X, so minimizing the error in this case is equivalent to maximizing

$$\langle \phi_1, C\phi_1 \rangle.$$

Recall that the set of eigenvectors of the covariance matrix form an orthogonal basis for $\mathbb{R}^n$ (which we will assume has been normalized). If we define V as

$$V = \begin{pmatrix} & & \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ & & \end{pmatrix}$$

then any of the columns of Φ can be written as a linear combination of the columns of V. In other words

$$\phi_i = \gamma_1 \mathbf{v}_1 + \cdots + \gamma_n \mathbf{v}_n = V\Gamma$$

for

$$\Gamma = \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_n \end{pmatrix}.$$

Furthermore, if

$$\Lambda = \begin{pmatrix} \lambda_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \lambda_n \end{pmatrix}$$

$$CV = \begin{pmatrix} & & \\ \lambda_1 \mathbf{v}_1 & \cdots & \lambda_n \mathbf{v}_n \\ & & \end{pmatrix} = V\Lambda.$$

Putting these two equalities together and applying it to our problem allows us to put the inner product in a different form:

$$\begin{aligned} \langle \phi_1, C\phi_1 \rangle &= \phi^T C \phi \\ &= \Gamma^T V^T C V \Gamma \\ &= \Gamma^T V^T V \Lambda \Gamma \\ &= \Gamma^T \Lambda \Gamma \\ &= \begin{pmatrix} \gamma_1 & \cdots & \gamma_n \end{pmatrix} \begin{pmatrix} \lambda_1 \gamma_1 \\ \vdots \\ \lambda_n \gamma_n \end{pmatrix} \\ &= \lambda_1 \gamma_1^2 + \cdots + \lambda_n \gamma_n^2. \end{aligned}$$

At this point, it is relatively easy to prove that

$$\langle \phi_1, C\phi_1 \rangle = \lambda_1 \gamma_1^2 + \cdots + \lambda_n \gamma_n^2 \leq \lambda_1.$$

First we show that, since ϕ_1 is a normal vector

$$\begin{aligned}\langle \phi_1, \phi_1 \rangle &= \|\phi_1\|^2 \\ &= \phi_1^T \phi_1 \\ &= \Gamma^T V^T V \Gamma \\ &= \Gamma^T \Gamma \\ &= \gamma_1^2 + \cdots + \gamma_n^2.\end{aligned}$$

Since this sum is equal to $\|\phi_1\|^2 = 1$, we can simply multiply it into our previous equation to get

$$\lambda_1 \gamma_1^2 + \cdots + \lambda_n \gamma_n^2 \leq \lambda_1(\gamma_1^2 + \cdots + \gamma_n^2).$$

Since the eigenvalues are organized such that $\lambda_1 \geq \cdots \geq \lambda_n$, the first term on the left and right sides of the equation will be equal, and each of the subsequent terms on the left side of the equation will always be less than or equal to its corresponding term on the right side since all $\gamma_i^2 \geq 0$. This proves that

$$\langle \phi_1, C\phi_1 \rangle \leq \lambda_1,$$

and since we want to maximize the left side of the equation, we want

$$\langle \phi_1, C\phi_1 \rangle = \lambda_1.$$

This result is easily obtained by setting ϕ_1 equal to $\mathbf{v}_1$. Finally we can conclude that the best 1-dimensional approximation to the rows of X is the eigenvector corresponding to the largest eigenvalue of the covariance matrix of X. This exact same argument can be applied to higher dimensional approximations, so that we can conclude that the best j-dimensional approximation to the rows of X is given by the first j sorted eigenvectors of the covariance matrix.

□

This provides a good way to find the best basis for the rowspace of X, however, it requires calculating C, which is a $n \times n$ matrix. If n is significantly larger than p (as it will be in our application) and you still want to find the best j-dimensional approximation of the rowspace without calculating the large matrix C, it is possible to approach the problem from a slightly different angle. Consider the (not quite finished) SVD of the matrix X,

$$X = U\Sigma V^T,$$

where Σ is $k \times k$ and k is the rank of X. By definition, we know that V is determined by the eigenvectors of the covariance matrix of X, so the best j-dimensional approximation to the rows of X is given by the right singular vectors of X, or the columns of V. Now consider the transpose of the singular value decompostion of X:

$$X^T = V\Sigma U^T.$$

In this case, the covariance matrix of X^T is defined as XX^T, which is the much smaller $p \times p$ matrix, and the singular vectors of X^X are related to those of X by

$$\begin{aligned} v_i\sigma_i &= X^T u_i \\ v_i &= \frac{1}{\sigma_i} X^T u_i. \end{aligned}$$

This useful trick gives us an easy way to calculate the columns of V without having to calculate the large $n \times n$ matrix C.

3 Face Recognition Programming Using *MATLAB*

In order to apply the mathematical processes to face recognition, a method to express the database of faces as a matrix is needed. The computer program *MATLAB* can be used to import images from a text file list of filenames and then convert each grayscale image into a $h \times w$ matrix, where w is the width (in pixels) of the image and h is the height. Once each image is imported and converted to the proper *MATLAB* format, it is then reshaped into a vector that is $(w * h) \times 1$, simply by placing the first column on the top and each successive column below its predecessor. Each vector now corresponds to a picture, and these vectors become the columns of a new matrix A, which is $(w * h) \times p$, where p is the total number of images used. *MATLAB* is then used to calculate the mean vector of the columns of A and subtracts this mean vector M from each of the columns in A to create the mean-subtracted matrix L. For a detailed explanation of this process, including the *MATLAB* code that we used, see Appendix A.

It is now possible to perform a simplified version of SVD calculations. The initial step is to calculate the covariance matrix $G = A^T A$. At this point the eigenvalues and their corresponding eigenvectors are calculated and sorted. They are sorted in descending order according to the size of the eigenvalues and the corresponding eigenvectors are placed in the same order. These eigenvectors become the columns of the $p \times p$ matrix V. In order to create a set of basis vectors for the columnspace of L the matrix U is created such that $U = L * V$. The span of the first column of U provides the best one-dimensional approximation of the column space of L. Similarly, the span of the first two columns provides the best two-dimensional approximation, and this line of thinking can be continued to obtain the best desired $j-$dimensional approximation. It is then possible to calculate what percentage of the variance is used by the $j-$dimensional approximation. This percentage (Per) is calculated using the sorted eigenvalues of the covariance matrix $\{\lambda_1 \dots \lambda_n\}$ in the equation

$$Per = \frac{\sum_{i=1}^{j} \lambda_i}{\sum_{i=1}^{n} \lambda_i} * 100\%.$$

In order to test an image from outside the database, *MATLAB* imports and reshapes the image in the same manner as that used for the images of the database. The mean vector M is then subtracted from the test image vector (T) and the coefficients of the first j bases are

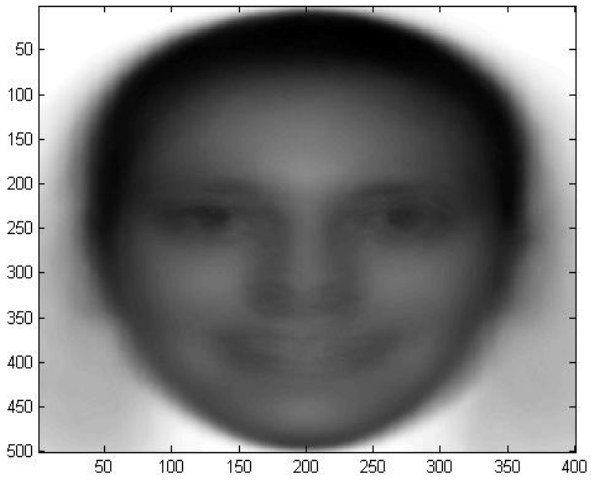

Figure 3: Average Face from First Database

calculated using the formula

$$C_t = \frac{U_t \cdot T}{U_t \cdot U_t}.$$

Where C_t is the t^{th} coefficient for T and U_t is the t^{th} column of U. From here, *MATLAB* simply calculates the Euclidean distance between the first j coefficients of T and the first j coefficients of all the pictures in the database. Once these distances are sorted (in ascending order) it allows for us to determine which image is the closest approximation of the test image. From that point onward, it is up to interpretation by human eyes to determine whether or not a match truly exists. The *MATLAB* code for testing an image from outside the database can be found in Appendix B.

4 First Database

The first database that we tested had 130 pictures of 65 different people. Each picture was 400×500 pixels and none of the pictures contained people wearing glasses. We chose to use images of people without their glasses to eliminate a "glasses" variable, which would potentially limit the amount of accuracy in the system. We used 15 of the 130 basis vectors in the computations, which retained 81.229% of the variance. The computer took 90 minutes (on a 2.8GHz processor with 512MB RAM) to load all 130 images into its memory and perform the required calculations. One output that is initially interesting to look at is the average face. This is simply the mean vector of the columns of A reshaped into a 400×500 picture. This yielded the image in Figure 3.

Another image that is output by *MATLAB* is informally called the eigenface. Mathematically, however, this image is one of the basis vectors that is reshaped into a 400×500 (or whatever the dimensions of the original picture was) image. The following figure is the first basis vector (first eigenface) for the database.

Each picture in the database, as has already been stated, has a coefficient that corresponds

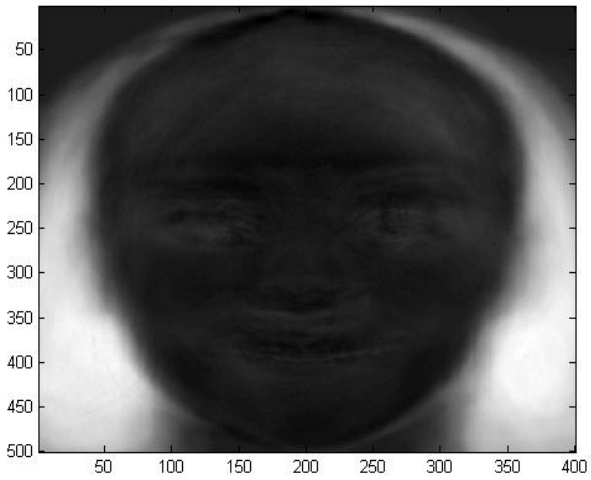

Figure 4: First Eigenface from Database 1

to this particular basis vector. Graphing the coefficients of this basis vector in terms of gender is very revealing to what facial feature this vector accentuates (See Figure 5).

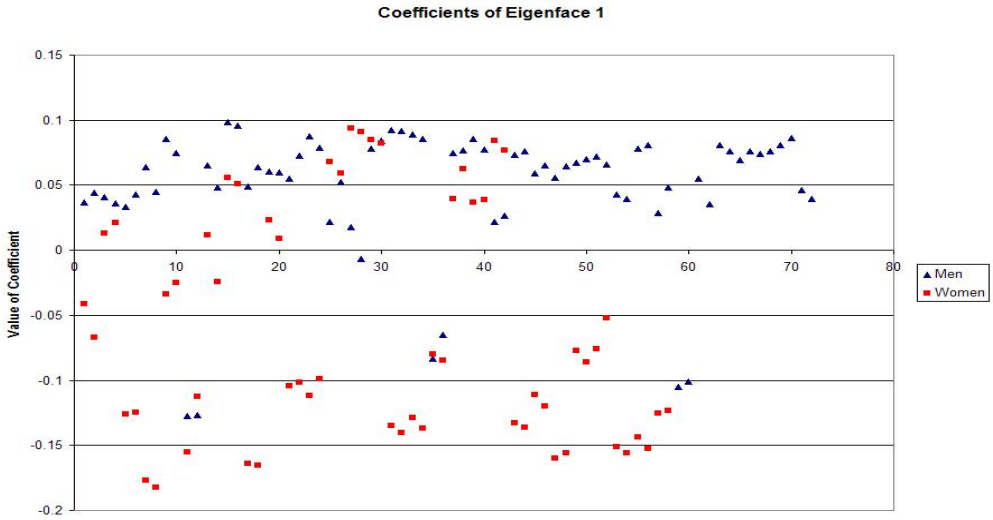

Figure 5: Coefficients of Eigenface 1 for each Face based on Gender

Most women have negative coefficients and the large majority of the males have positive coefficients. Furthermore, when we went back and examined the pictures of men with negative coefficients we noted that they all had long hair. Also, when the images of females were examined, those with a negative coefficient had long hair, while those with a more positive coefficient had shorter hair length. This fact is slightly troublesome because it indicates that the most important aspect of a person's face was the length/amount of hair, which is also the easiest feature of one's face to change. This fact could potentially be a weakness for the system

since any major change in hair length or shape would likely cause failure in the system.

When examining the second eigenface it was less clear which features the system was identifying as important. We hypothesized that one of the possible features the eigenface may be identifying is the difference between a person's hair color and the darkness/lightness of a person's skin tone. We arrived at this conclusion by analysis similar to that used for the previous eigenface. We found it relatively difficult to determine which features were being distinguished in the eigenfaces that followed. Clearly, the program was differentiating between some features, but it was a failure of human perception to determine what those features were.

The next step was to take the first fifteen coefficients of each face and place them in a vector. The formula for the Euclidean length of a vector was used to determine the length of the vector. This allowed us to discover which face had the overall largest magnitude. Thus, the vector with the highest magnitude corresponded to the face that can be best approximated. Figure 6 shows the original face that could be best approximated (had the largest coefficient vector length) and the fifteen vector approximation of this face. This approximated image was obtained by taking the first fifteen vectors multiplying by their corresponding coefficients and adding the resulting vectors together. It is clear from these two images that using only 15 of the 130 basis vectors produces a readily recognizable approximation. In contrast, the face with the worst approximation is also shown in Figure 6.

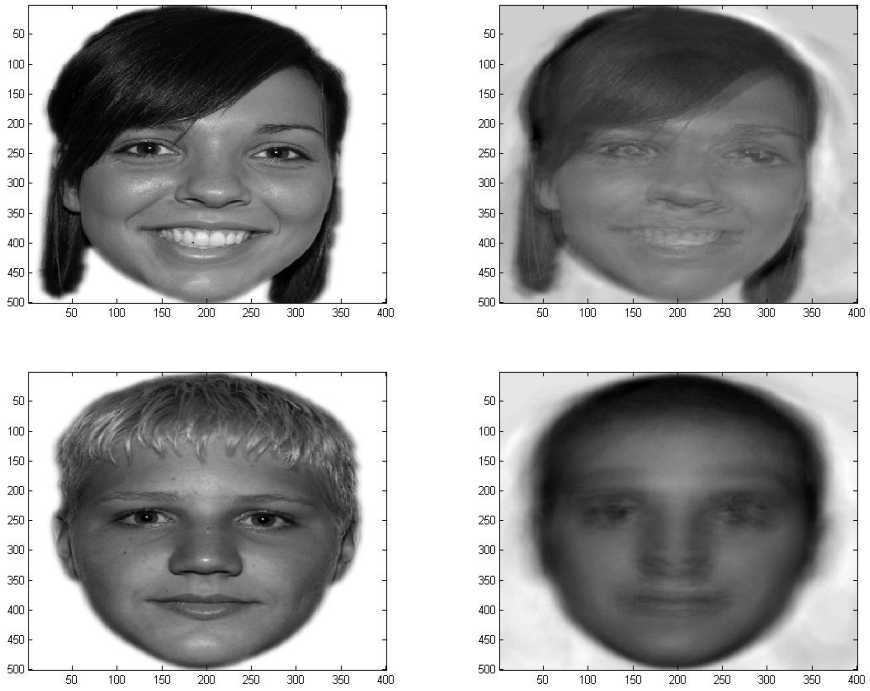

Figure 6: The top two are the original face and the fifteen vector approximation of the face that can be best approximated. Similarly, the bottom two are the original face and the fifteen vector approximation of the face that is most inaccurately approximated.

It is now possible to look at the actual face recognition applications. As has already been stated, this was performed by taking the Euclidean distance between the coefficients of the

orthogonal projection of the outside image with the coefficients of each image in the database. The image that had the smallest distance from the test image was identified by the program as being the closest match, and all the remaining images were ranked from closest to farthest away (in terms of Euclidean distance) from the test image. The results from this series of tests were very positive and indicated that the system worked well overall. Yet, it became clear very quickly that consistency in the orientation of the faces was needed so that proper recognition can occur. When all of the faces were facing forward, with minimal head tilt, the system identified the correct person as being the closest match 83.78% of the time. Furthermore, 97.30% of the trials identified at least one correct match in the top three closest matches. It is important to note that many of the test images that were used had people wearing glasses. We determined that the presence of glasses had little to no influence on the outcome of the results. On the contrary, facial expression did have an impact on the results. This was indicated by several trials where the person had a significantly contorted facial expression compared to the expression in the database. In these cases the system had a more difficult time identifying the proper faces as being a match. We also noticed that several of the failed trials most likely resulted from the tilt of the person's head. Thus, tilt and orientation of the person's face were the two most significant factors that negatively affected the outcome.

Since we noticed that the first eigenface identified long hair as being one of the most significant factors in making a correct match, we tested how well it would match a person when significant changes to their hair occurred. First, we used Photoshop to take one of the female subjects with black hair and change the color to blonde. Despite the color change, the system was still able to correctly identify her in the database (little to no influence on results occurred). Next, we took some of the pictures with long hair (people who had a large value for the first eigenface) and used Photoshop to remove a large portion of their hair. We found that the system was unable to recognize the correct person as the best match when large changes to hair were performed.

Another interesting test that we conducted involved placing a set of twins in the database. We tested two images of each person and in each case the system picked out both pictures in the database as being one of the top 3 matches. Yet, interestingly enough the other brother never appeared in the top three closest matches. This is especially important because even to the human eye it is particularly difficult to differentiate between the two brothers. This can be demonstrated by Figure 7.

5 Second Database

We thought at this point that it would be important to see how well another database would work by using smaller image files (fewer pixel dimensions). The use of such downsized images is desirable because the amount of computer processing time greatly decreased with smaller image files. Thus, we used the same pictures, but reduced the size to 350×438 pixels. This reduced the processing time to approximately nine minutes, which is a tenth of the previous database's processing time. Again, we used 15 basis vectors, which retained 70.90% of the variance. As in the previous database, none of the subjects had glasses. However, we did use Photoshop to remove long hair around the neck and lower part of the face, which we hypothesized would reduce the overall amount of influence hair has in the system. Consequently, the average face for this database looked essentially the same, but the outline of hair around the lower half of the face is removed (See Figure 8). The lack of hair greatly influenced the outcome of the first

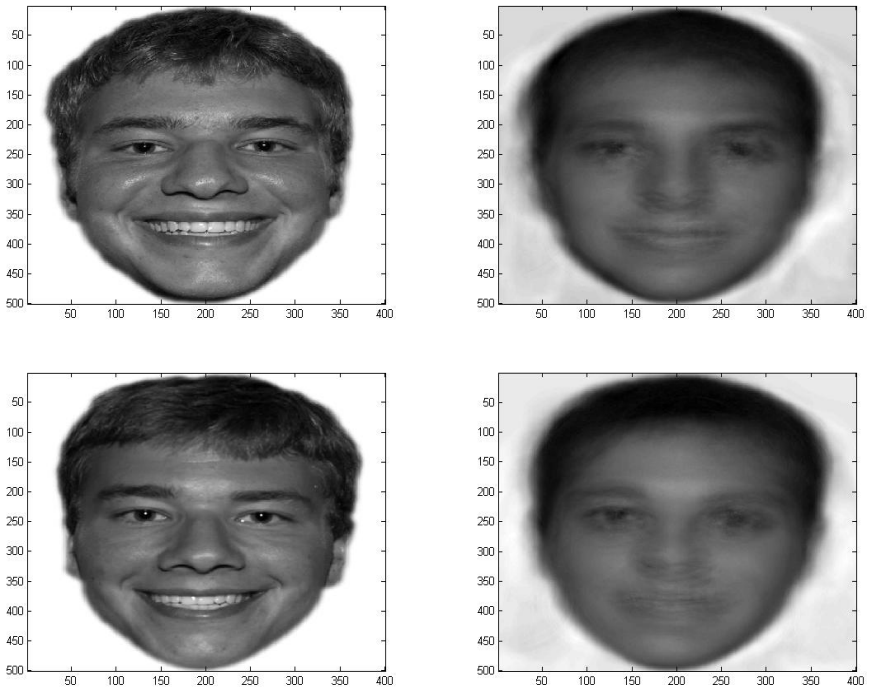

Figure 7: The top two images are the original face and the fifteen vector approximation of one of the brothers. The bottom two are the original face and the fifteen vector approximation of the other brother.

eigenface, as can also be seen in Figure 8.

In this eigenface, there appears to be a contrast between a person's facial color and the color of their hair, as opposed to the first eigenface from the first database, which focused on whether or not a person had long hair. The subsequent eigenfaces were somewhat difficult to determine what was being differentiated. Yet, the second eigenface seemed to be detecting differences in skin tone, and the third eigenface seemed to be looking at how much hair was at the top of a person's head.

We noticed, again, that this system was very sensitive to the rotation of a person's head. Consequently, we decided to test how much tilt could be present before the system started to fail. We began with a different picture of a person who was already in the database and rotated it 25°. When this rotation was used the correct face was the 51^{st} closest match. We then made the rotation progressively closer to center in 5° increments. When the face was 20° from center the closest matching face was number 34. When there was a 15° offset from center the closest match appeared in the 11^{th} position. When the rotation was 10° the fifth match was correct. Once the rotation was both 5° and straight up and down the first match was the correct face.

When we tested images from outside the database we also removed the long hair using the same method to create the database. This experiment yielded a 70.00% success rate. There are several factors that contributed to the decrease in success. The most likely scenario is that the removal of the hair in Photoshop was not performed in a completely controlled/automated

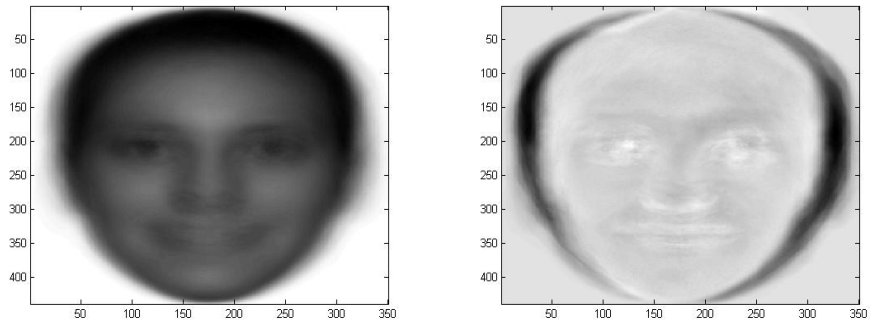

Figure 8: The image on the left is the average face from the second database, and the image on the right is the first eigenface from this database.

manner. Instead the human eye was used to judge how much excess hair needed to be removed. Another possible factor for the decreased success rate originates from the fact that smaller image files were used. These smaller images resulted in less of the variance being retained (compared to the first database) even though the same number of basis vectors were used as in the first database. In order to correctly determine which factor was most influential we needed to run more tests, which will be discussed shortly. We also tested images with the long hair left in the picture and the success rate was 30.00%.

The next test we ran was with people who already had short hair. This produced a 93.75% success rate, in that at least one of the correct images from the database was chosen as one of the top three matches. Furthermore, 83.38% were selected as the number one closest match. These results indicate that in the previous consideration of whether or not human error or the decreased variance contributed most to the smaller success rate, human error was the largest contributing factor. We drew this conclusion on the basis that in the the test where human error was not a factor the success rate only decreased slightly from that of the first database.

Conclusion

The technique is extremely successful and requires a minimal amount of data input. Although it is extremely important for all of the faces to be oriented in the same manner, and once this is achieved, the technique works very well. Although we decreased the size of the images in the second database by over 23% and the processing time by approximately 90%, we were still able to obtain a success rate comparable to that of the first database.

Further research in this area could focus on testing a larger number of variables in the pictures. Another great consideration would be further research on the most effective method to use for the preprocessing of images that are placed in the database. We would also like to pursue greater automation of the entire system so that more tests with a greater amount of images can be performed more efficiently.

Acknowledgements

We would like to extend a special thank you to Dr. Jim Seibert for his continual guidance and support throughout the course of the project. We also wish to thank Dr. Suzanne Caulk for the use of her computer and office during the research process, as well as for listening to our ideas over the summer. We also want to acknowledge the Office of Academic Grants at Regis University for funding the research project.

A Code for Database Formation

The following code was used to load all of the images in the database from a textfile. This textfile was generated using the command line within Windows. For more information regarding how to change directories and produce a bare format filelist see [8].

In general this *MATLAB* function loads all of the images, converts them to the proper ("double") format and reshapes them all into vectors. Then, the function performs all of the computations on the database which are necessary to create the best basis as described by Proposition 2.8. Please see our explanations throughout the code (denoted with the % symbol). Before considering the code, however, we must note it is necessary to define two other functions which were used within this image loading function. The first is defined as "linecount" and its purpose is to count the number of lines within the filelist to determine how many images we will be loading. The code is as follows.

```
function lc=linecount(filename)

fid=fopen(filename,'r'); if fid < 0
    lc=0;
else
    lc=0;
    while 1
        ln=fgetl(fid);
        if ~isstr(ln) break; end;
        lc=lc+1;
    end;
    fclose(fid);
end;
```

The second function which is needed to execute the "load images" function is known as "sorteig". This function simply sorts the eigenvalues of our covariance matrix in descending order and then sorts the corresponding eigenvectors in the same order.

```
function[V1,L1]=sorteig(V,L)
lambda=diag(L);
n=length(lambda);
[val,idx]=sort(lambda);
val=val(n:-1:1);
idx=idx(n:-1:1);
L1=diag(val);
```

```
V1=V(:,idx);
```

Now, with all the previous m-files loaded into the work folder of *MATLAB* we can define our "load images" function.

```
function [Images, w, h, Meanimage, Meanface, Meansub, V, S, Basis,
SmallBasis, SmallV, Percent] =load_images(filelist,b)
numimgs=linecount(filelist); fid=fopen(filelist,'r');

for i=1:numimgs
  imgname=fgetl(fid);
  if ~isstr(imgname)
      break;
  end;

  fprintf(1, 'loading JPG file %s\n', imgname);

  Img=imread(imgname);

  Img=double(Img)+1;

  if i==1
      [w,h]=size(Img);
  end;

  Images(1:w*h,i)=reshape(Img,w*h,1);
   end;
  fclose(fid);

  fprintf(1, 'Read %d images.\n', numimgs);
%All of these lines up to now have simply loaded the images,
%converted them to the proper format, reshaped them into
%vectors and printed out a running list as it loaded each file

  Meanimage=mean(Images');

  Meanface=reshape(Meanimage, w, h);
  imagesc(Meanface); figure(gcf);
  colormap(gray); %Displays the average face

  Meanimage=Meanimage';

  Meansub=Images-repmat(Meanimage, 1, numimgs);

  Covar=Meansub'*Meansub; %Calculates the covariance matrix

  [V, S]=eig(Covar);
```

```
  [V, S]=sorteig(V,S);

  Basis=Meansub*V; %Calculates basis from eigenvectors

  for g=1:b
      SmallBasis(:,g)=Basis(:,g);
      SmallV(:,g)=V(:,g);
  end

  for t=1:b
      Per=sum(diag(S(1:t,1:t)));
      Percent=(Per/(sum(diag(S))))*100;
%This calculates the percent of the variance retained by
%our b basis vectors
  end
```

B Code for Testing Image Outside the Database

This code begins in basically the same way as the database formation by loading the outside image into the *MATLAB* workspace. The test image is then mean subtracted and projected onto the subspace spanned by our chosen number of basis vectors. The distance from each of the images in the database is then calculated. These vectors, which are the database images, are then sorted from smallest to greatest distance. The final aspect of the code displays the test image alongside the top three closest matches from within the database.

```
function [RTest, Coeff, Dist, B, Index, best1, best2,
best3]=outside_image(testimage,SmallBasis, Meanimage, SmallV, Image,
k) numimgs=linecount(testimage); fid=fopen(testimage,'r');

for i=1:numimgs
  imgname=fgetl(fid);
  if ~isstr(imgname)
      break;
  end;

  Img=imread(imgname);

  Img=double(Img)+1;

  if i==1
      [w,h]=size(Img);
  end;

  RTest(1:w*h,i)=reshape(Img,w*h,1);
   end;
  fclose(fid);
```

```
  RTest=RTest-Meanimage; %Mean-subtract the test image

  for i=1:k
     Coeff(1,i)=(dot(SmallBasis(:,i),RTest))/(dot(SmallBasis(:,i),SmallBasis(:,i)));
  end
%Projects the test image onto the space spanned by the chosen basis vectors

  [r, c]=size(SmallV);

  for i=1:r
      Dist(i,1)=norm(SmallV(i,:)-Coeff);
  end
%Calculates the distance of this projection from all the database images

  [B,Index]=sortrows(Dist);
%Sorts the database images from least to greatest distance from the test image

  m1=Index(1,1);
  m2=Index(2,1);
  m3=Index(3,1);

  best1=Image(:,m1);
  best2=Image(:,m2);
  best3=Image(:,m3);

  best1=reshape(best1, w, h);
  best2=reshape(best2, w, h);
  best3=reshape(best3, w, h);

  figure(1);
  imagesc(Img);
  colormap(gray);

  figure(2)
  imagesc(best1);
  colormap(gray);

  figure(3);
  imagesc(best2);
  colormap(gray);

  figure(4);
  imagesc(best3);
  colormap(gray);
%Displays the test image and three closest matches from the database
```

References

[1] R. Chellappa, C.L. Wilson, and S. Sirohey. "Human and Machine Recognition of Faces: A Survey." *Proceedings of the IEEE.* Vol. 83, No. 5. May 1995. Pp. 709-40.

[2] L. Sirovich and M. Kirby. "Low dimensional procedure for the characterization of human faces." *Journal of the Optical Society of America A, Optics and Image Science.* Vol. 4, No. 3. March 1987. Pp. 519-24.

[3] M.A. Turk and A.P. Pentland. "Face Recognition Using Eigenfaces." *International Conference on Pattern Recognition.* 1991. Pp. 586-91.

[4] M. Turk and A. Pentland. "Eigenfaces for Recognition." *Journal of Cognitive Neuroscience.* Vol. 3, No. 1. 1991. Pp. 71-86.

[5] D. Hundley. *Chapter 6: The Best Basis.* 2004. Whitman College. 20 June 2005. <http://marcus.whitman.edu/∼hundledr/courses/M350/Ch5-Ch7.ps>.

[6] R.A. Horn and C.R. Johnson. *Matrix Analysis.* New York: Cambridge University Press, 1992.

[7] D.C. Lay. *Linear Algebra And Its Applications.* 3rd ed. Boston: Addison Wesley, 2003.

[8] S. Dutch. "Capturing File Lists." 28 March 2002. University of Wisconsin-Green Bay. 5 July 2005. <http://www.uwgb.edu/dutchs/CompTips/FileLists.HTM>.