ECE 420: Embedded DSP Laboratory
Lab Assigned Project Lab
Eigenfaces for Recognition
Paper Summary

Jonathan Wang & Kristian Lauszus
NetID: JYWANG6 & LAUSZUS2

October 18, 2016

1 Introduction

The face acts as the primary identifier in our social intercourse. The human ability to recognize faces is therefore an essential skill for almost everything we do. However, this often-times assumed skill is much more difficult to accomplish quickly and accurately in a digital world. In the paper **Eigenfaces for Recognition**, the authors Matthew Turk and Alex Pentland evaluates this 3D facial recognition problem as a 2D recognition problem involving a small set of 2D characteristic views. This is done by presenting the face in constrained environments such as an office or a household, and treating it not as a complex 3D geometry, but rather a 2D shape with a small subset of points of interests, or "features". By simplifying the problem, the algorithm can be performed in near-real-time with high accuracy in the given constrained environment.

2 Methods

2.1 The Eigenface Approach

In order to reduce the complex facial recognition problem, the significant local and global "features" of faces must be found and evaluated. These features do not necessarily correlate to the intuitive notion of facial features such as eyes, nose, hair, etc. Instead, they should be extracted from the comparison of a large variety of faces as the features that vary the most from one face to another. Mathematically speaking, the procedure is to find the principal components of the distribution of faces. In other words, if each face is treated as a point/vector in a very high dimensional space, we wish to find the eigenvectors of the covariance matrix of a set of said face points/vectors. Each eigenvector corresponds to a single mathematical feature, and can be displayed as an *eigenface*. A linear combination of all the eigenfaces produces a unique face. Furthermore, we can define an M-dimensional subspace - the "face space" as the span of the best M eigenfaces. Each face can then be approximated as a linear combination of these M eigenfaces. With this initialization, we can reduce each face down to a set of M dimensional weights, by which the original face can be approximated. To summarize, we perform the following steps to initialize the system:

1. Acquire the training image set.
2. Calculate the eigenfaces from the training set, keep M highest eigenvectors and define the *face space*. (This step can be repeated when necessary to update face database)
3. Calculate the weight space spanning M-dimensions for each of the face in the database by projecting their face images onto the *face space*.

With the initialized system, we can then recognize new face images by:

1. Calculate a set of weights based on the input image and M eigenfaces by projecting the input image onto each of the eigenfaces.
2. Calculate the face's distance to the *face space*, determine if it is within a preset boundary (determine whether or not it is a face).
3. If a face is identified, classify it as known or unknown.

2.1.1 Calculating the Eigenfaces

1. Obtain $I_1, I_2, ..., I_M$ (training faces, centered and cropped to the same size)

2. Represent every image I_i as a vector Γ_i
   This Γ is an $N^2 \times 1$ vector, corresponding to an $N \times N$ face image I.

3. Compute the average face vector Ψ:

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$$

4. Subtract the mean face:

$$\Phi_i = \Gamma_i - \Psi$$

5. Compute the covariance matrix C:

   Define: $A = [\Phi_1\ \Phi 2\ \ldots\ \Phi_M]$ $N^2 \times M$ matrix

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T\ (N^2 \times N^2 \text{ matrix})$$

6. Compute the eigenvectors u_i of AA^T
   However, notice that AA^T $N^2 \times M$ matrix is very large, and findings its eigenvectors is not practical in near-real-time. Therefore we consider a simpler approach:

   (a) Compute the eigenvectors v_i of the matrix A^TA

$$A^TAv_i = \mu_i v_i$$

   The relationship between the eigenvalues u_i and eigenvectors v_i is given by:

$$A^TAv_i = \mu_i v_i$$
$$AA^TAv_i = \mu_i Av_i$$
$$AA^Tu_i = \mu_i u_i$$

   AA^T and A^TA have the same eigenvalues, their eigenvectors relate by: $u_i = Av_i$. This reduces the burden of finding N^2 eigenvalues and eigenvectors down to only M eigenvalues and eigenvectors. These M eigenvectors/values represent the M largest eigenvectors/values of AA^T.

   (b) Compute the M best eigenvectors of AA^T using aforementioned method: $u_i = Av_i$

   (c) Normalize u_i ($||u_i|| = 1$)

7. Keep K eigenvectors (manually set threshold, K largest eigenvalues and their eigenvectors that are needed to represent the initial data with minimal inaccuracy)

In this K basis, each face (minus the mean) in the training set can be represented as a linear combination of the best K eigenvectors:

$$\hat{\Phi}_i = \sum_{j=1}^{K} w_j u_j\ (w_j = u_j^T\Phi_i)$$

The u_j's, or eigenvectors, are also referred to as the eigenfaces. Each normalized training face Φ_i can be represented in this K basis by the vector:

$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ \ldots \\ w_K^i \end{bmatrix}\ i = 1, 2, ..., M$$

2.1.2 Face Detection

Given an unknown image Γ, we want to know whether or not it is a face. To do so we calculate the **distance from face space** ϵ_d.

1. Compute: $\Phi = \Gamma - \Psi$
2. Compute: $\hat{\Phi} = \sum_{i=1}^{K} w_i u_i\ (w_i = u_i^T\Phi)$
3. Compute: $\epsilon_d = ||\Phi - \hat{\Phi}||$
4. Define a detection threshold T_d. If $\epsilon_d < T_d$, then Γ is a face.

2.1.3 Face Recognition

Given an unknown face image Γ, we want to know whether or not it is in the training database. We first crop and center the face the same way as the training images. Then, we define a **distance within the face space** ϵ_r. The following procedures are taken:

1. Compute: $\Phi = \Gamma - \Psi$
2. Project the normalized face onto the face space (eigenspace):

$$\hat{\Phi} = \sum_{i=1}^{K} w_i u_i\ (w_i = u_i^T\Phi)$$

3. Represent Φ as:

$$\Omega = \begin{bmatrix} w_1^i \\ w_2^i \\ \ldots \\ w_K^i \end{bmatrix}\ i = 1, 2, ..., M$$

4. Find $\epsilon_r = min_l||\Omega - \Omega_l||$
   Here we find the closest known Ω_l, or weight vector in the eigen face basis, from the new Ω in order to find the closest candidate for a match.

5. Define a recognition threshold T_r. If $\epsilon_r < T_r$, then Γ is recognized as face l from the training set.

Note that when a new image is presented and analysed, there are four possible outcomes, they are tabulated in Table 1.

3 Our Attempt

3.1 Preface

The steps described earlier were those proposed and performed by the authors of the paper. Based on our understanding of the algorithm presented, we first recreated the eigenface initialization and recognition program in Matlab. The paper uses a database of 16 distinct faces as a training set. In order to more thoroughly test the algorithm, we used the "Our Database of Faces" created and distributed by AT&T Laboratories Cambridge. This database contains the faces of 40 individuals, each with 10 different images taken with varying lighting, facial expression, and facial details (glasses / no glasses). This gives us a training set of 400 images in the PGM format, with 92x112 pixels in 8-bit greyscale.

3.2 Singular Value Decomposition

The 6th step in section 2.1.1 describes the process of Principal Component Analysis (PCA), where we find the eigenvectors of the covariance matrix. An efficient and popular way of performing PCA is through Singular Value Decomposition (SVD). Putting the covariance matrix (or the smaller version of it) through SVD yields $\boldsymbol{U}\,\boldsymbol{S}\,\boldsymbol{V}^*$. For our purposes, we take $\boldsymbol{V}^*$, which contains the eigenvectors sorted from high to low variance, and $\boldsymbol{S}$, which contains the singular values corresponding to the eigenvectors. Since the singular values indicate the importance of the corresponding eigenvectors, we can use it to choose the number of eigenfaces to keep.

With the AT&T Facedatabase, we find that the 10-16 largest eigenfaces cover the majority of the variance between different faces, as seen in the plot of the singular values obtained from SVD in Figure 1. Therefore, the optimal balance between speed/memory usage and reliability for our case lies in a choice of $K \in [10, 16]$. Choosing $K = 16$, we generate and display the top 16 eigenfaces (shown in Figure 2), corresponding to the 16 most significant "features" determined by PCA. While it is difficult to relate the eigenfaces to human-distinguishable features, it is generally applicable to view the areas with high contrast (within each eigenface) as the "features" represented.

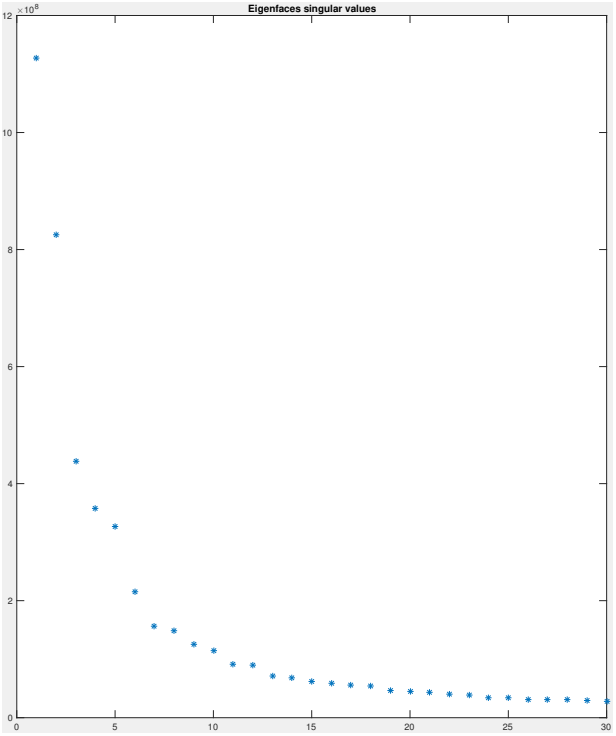

Figure 1: Singular values from AT&T Facedatabase

3.2.1 Calculate K from cumulative energy

There is a more elegant way of automating the finding of the optimal K value. If we view each singular value as the "energy content" of the corresponding eigenvector, then we can find the cumulative distribution function of the singular values, and choose the cut-off mark K as the point where 90 % of data variance is achieved. The following code snippet , which is als implemented in our code, illustrates this technique.

```
% Calculate the cumulative sum of the singular values
cumulativeEnergy = cumsum(diag(S));
% Convert to percentage
cumulativeEnergy = cumulativeEnergy/cumulativeEnergy(end);
% Find the index where the cumulative energy is above or equal 90 %
K = max(2, find(cumulativeEnergy >= .9,1,'first'))
```

4 Result Analysis and Going Forward

Between Figure 3 and Figure 4 we see that the former, which is computed using $K = 30$ and the latter, which is computed using $K = 10$, a large increase in the K value doesn't necessarily guarantee a higher recognition success rate. In fact, a lower

Face Detection	Face Recognition	Implication
$\epsilon_d < T_d$	$\epsilon_r < T_r$	Image is a face, face is recognized
$\epsilon_d < T_d$	$\epsilon_r > T_r$	Image is a face, face is unrecognized
$\epsilon_d > T_d$	$\epsilon_r < T_r$	Image isn't a face, recognition returns false positive
$\epsilon_d > T_d$	$\epsilon_r > T_r$	Image isn't a face, face is unrecognized as expected

Table 1: Possible Outcomes

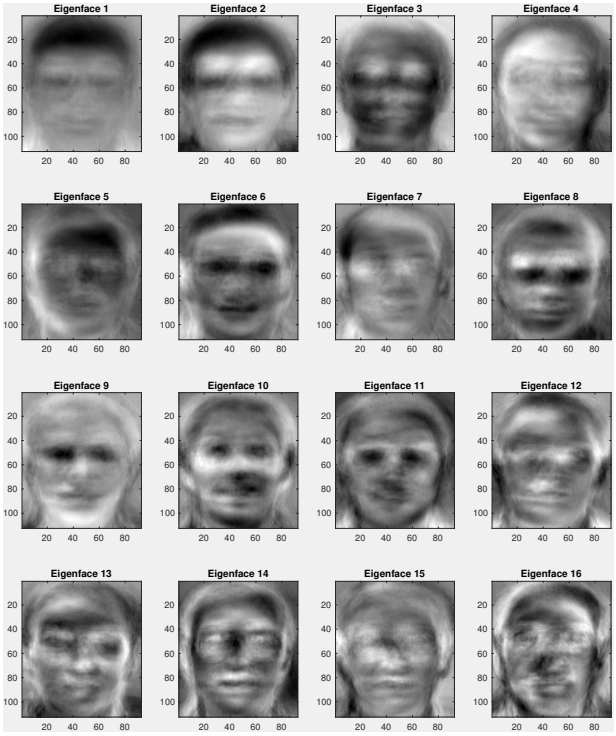

Figure 2: 16 Most Significant Eigenfaces

K value actually performed better for the input picture in Figure 4 than it did in Figure 3, as it identified no mismatches for all 9 images. This phenomenon highlights a shortfall of the Eigenfaces method. While PCA maximizes the scatter between different face classes, it also maximizes the scatter within a single face class. In fact, the concept of a class containing multiple images of the same face doesn't exist. Rather, each training image is treated as a new face, and the projection and calculations simply find the closest face in the *face space*.

Therefore, by using PCA we end up with a recognition system that tests well against the original images, but fails more frequently when faced with variation in poses and lighting. As suggest by Moses et al., lighting and expression variations make up for the largest source of variance between different faces[1]. A proposed remedy to this problem is to discard the largest three principal components after performing PCA. However, there is no way to be sure that for every training database presented, the top three principal components correspond to these large yet insignificant variances. Furthermore, it is very likely that we would lose other important "features" in the meantime.

Additionally, the database of eigenfaces have to be retrained from the beginning with every addition of a new face. This makes the algorithm less practical as a real-time adaptive system, but more suitable as a system with a static database that only performs recognition.

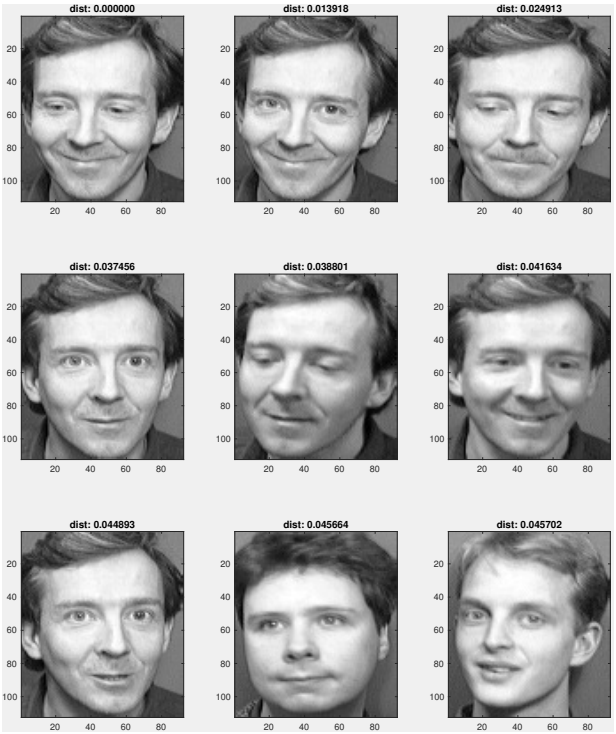

Figure 3: Facial Recognition using K = 30

4.1 C++ Implementation

In the source code attached we have also implemented the aforementioned Eigenfaces method, along with the edits and improvements, in C++, using the fully templated "Eigen3" and "RedSVD" libraries. The goal in doing this is to increase cross-compatibility of the program with different platforms. However, we discovered that by foregoing the overhead that Matlab provides with built in matrix manipulation, we are able to see a runtime speed improvement of 50% with the AT&T database, from an average of 0.8s to 0.4s. Another interesting phenomenon we observed was the sign ambiguity in singular value decomposition. When we calculated the SVD in our C++ implementation, we find that while the singular values and the absolute values of the eigenvectors are consistent, the signs of the eigenvectors are often flipped between the Matlab and the C++ solution. As a result, the greyscale plots of the eigenfaces are sometimes inverted in terms of the "color". Since the following analysis only requires the distance measurement, the sign ambiguity is unimportant. If desired, the C++ code can be implemented to follow the sign of the majority for each eigenvector, as that is the standard Matlab uses. The Matlab and C++ code is available at the following link: `https://github.com/Lauszus/Eigenfaces`.

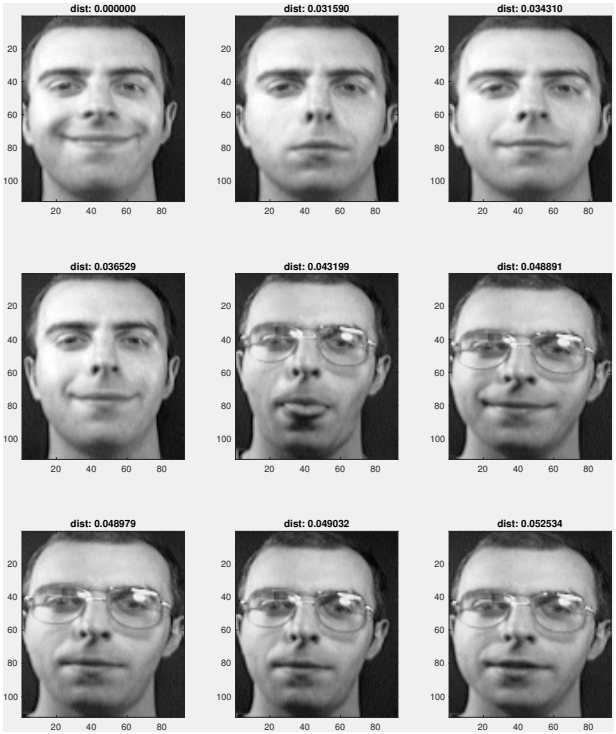

Figure 4: Facial Recognition with Glasses, K = 10

References

[1] Y. Adini, Y. Moses, and S. Ullman. Face recognition: The problem of compensating for changes in illumination direction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):721–732, 07 1997. ISSN 0162-8828. doi: 10.1109/34.598229.

[2] Philipp Wegner. Eigenfaces. `http://www.bytefish.de/blog/eigenfaces`, 07 2011. Visited: 2016-10-17.

[3] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 01 1991. ISSN 0898-929X. doi: 10.1162/jocn.1991.3.1.71.