Facial Recognition Report

Charlie Weiss

I. Introduction

In this report, I explore two methods of facial recognition. The first, Eigenfaces, is a fascinating method of reducing a training set of images to a basis set of eigenvectors (which can be represented as images), which in turn can be used to recreate various images in the dataset. This is fascinating from a mathematical, conceptual, and visual perspective, and even obtains an impressive amount of accuracy when applied to facial recognition. It compares well to the second algorithm, Pixel to Pixel Correlation, an elegant application of matrix operations to achieve an even higher level of accuracy in facial recognition.

II. Background and Terminology

Facial recognition has many interesting applications. Among more obvious ones, such as surveillance and security, it has applications in dating sites, pet adoption, social media and animal and plant identification.[1] While these may seem unrelated, all incorporate aspects of a computer's representation of visual comparison and identification, which is the core of the field of facial recognition.

Facial recognition struggles to perform under certain conditions. The most common struggles stem from comparing images with poor lighting, facial obstructions such as sunglasses and long hair, differences in facial position, skin tone, and low resolution images. Expressions can also complicate the facial recognition process, which is why many governments require subjects to have neutral expressions in their identification photos.[2]

The Eigenface method is particularly interesting, because of its unique use of linear algebra to represent images. While it may not be the most effective method in existence, the concept itself is intriguing, and represents some surprising applications of linear algebra.

III. Algorithms and Justifications

I use two algorithms for facial recognition. The first is called 'Eigenfaces' and the second is called 'Pixel to Pixel Correlation.'

Eigenfaces

Eigenfaces is a useful method for facial recognition. It consists of deriving eigenvectors from the covariance matrix of the probability distribution over the high-dimensional vector space of a set of images, a description which will have meaning by the end of this paper. The eigenvectors (or eigenfaces) form a basis set of all the images used in the covariance matrix, allowing a smaller set of images to represent the original training set. After this, facial recognition can be implemented by simply comparing how different faces are represented by the set of eigenfaces.

C. Weiss is with Olin College of Engineering

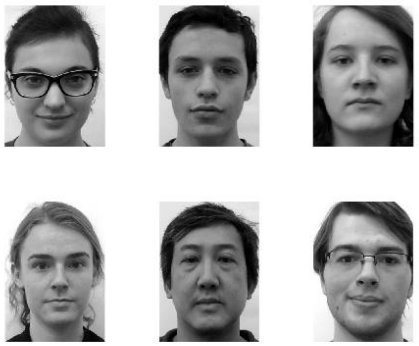

Fig. 1. The first six images from a training image set of 43 faces. Each face has a neutral expression with similar lighting conditions and head positions. Each person corresponds to a greater image set of 8 different expressions per person, which will be used for comparison and recognition.

The Eigenfaces algorithm can be described as a series of steps. First, it relies on a set of s training images which includes a picture of each person one hopes to identify, with the same resolution ($n \times m$). These images are more effective if taken under similar conditions (lighting, resolution, head position, etc.), as we shall see later. In my training set, I take 43 photos of a classroom of college students and teachers with neutral expressions, where each image has a resolution of 360×256. Some examples of these are shown in Figure 1.

These training images must be transformed into vectors, with the rows of the original image concatenated to create a $1 \times (nm)$ row vector such that

$$x = [a_{1,1}, \ldots, a_{1,m}, a_{2,1}, \ldots, a_{2,m}, \ldots, a_{n,1}, \ldots, a_{n,m}]$$

where x is the image vector and a_{ij} is the entry in the ith column and jth row.

One can now find the "mean face" of all these images, by finding the average value of each pixel across all images. Figure 2 shows the resulting vector, reconstructed back into an image matrix. Subtracting the mean image from each image vector gives us a set of centralized data, which we can pack into an $nm \times s$ matrix of training set data T, where

$$T = [x_1, x_2, \ldots, x_s]$$

where x_1 is the transpose of the first training image in the set, and x_s is the last.

Fig. 2. Mean image of the test image set. It is convincingly blurred and average looking.

The next step is to create a covariance matrix C out of matrix T and its transposes, with $C = TT^T$. However, this will result in an $nm \times nm$ matrix, which here is 92160×92160. This is not desirable, because we intend to perform an eigen decomposition on this matrix, and it will not be computationally efficient.

Luckily, there is a clever workaround, utilizing a modified Principal Component Analysis (PCA) method of determining the eigenvectors. First, the eigenvector decomposition of C is given by

$$Cv_i = TT^T v_i = \lambda_i v_i$$

where C and T are as previously defined, v_i is the ith eigenvector of the covariance matrix, and λ_i is the ith eigenvalue. However, we can take an alternate eigenvalue decomposition

$$T^T T u_i = \lambda_i u_i$$

multiply both sides of the equation with T to obtain

$$TT^T T u_i = \lambda_i T u_i$$

and notice that we have defined

$$T u_i = v_i$$

so that we can find the eigenvectors u_i of $T^T T$, a manageable $s \times s$ or 43×43 matrix, and multiply them by T to get the first s eigenvectors of the covariance matrix and their corresponding eigenvalues. This method returns the eigenvectors with the largest eigenvalues in the original covariance matrix C, so while some data is lost, the most important information is retained.

Now we have the first s eigenvectors and eigenvalues of our hypothetical covariance matrix of training images, which happen to be the largest eigenvalues and their corresponding eigenvectors. These eigenvectors are what are called eigenfaces, for when they are reshaped into an image matrix, they form the haunting faces seen in Figure 3. Each face tells you a pattern of variation between images in the training set, and each corresponding eigenvalue signals how important that variance is.

Fig. 3. Examples of eigenfaces. The patterns of the faces shows certain qualities of the test image set.

Now we can begin to use the eigenfaces to reconstruct other images.

We can utilize this quality for facial recognition. Begin by taking the dot product of a face and each eigenface, essentially projecting the face onto s-dimensional space made up of the eigenvectors. Mathematically, this is very simple, as one can simply perform:

$$P_s = V_s^T t^T$$

where V_s is an $nm \times s$ matrix whose columns are the first s eigenfaces, t^T is the transpose of the image as a row vector, and P_s is the resulting $s \times 1$ column vector of the scalar result of each projection.

Now the original image can be reconstructed with the following matrix operation:

$$r_s = V_s P_s$$

where r_s is the reconstructed image from s eigenfaces.

It is much easier to visual this operation in a simple case. Consider when the s-dimensional space is only two-dimensional, as in only two eigenfaces are used to reconstruct the original image. Thus, V_s is made up of only two eigenvectors, which can be represented as two-dimensional axes, and P_s is a 2×1 column vector, which can be said to contain coordinates along the eigen-axes. This point represents the projection of the original image onto the eigenfaces. Figure 4 shows a scatterplot of each of the training images projected onto the eigenfaces.

However, when the images are projected onto only 2 of the 43 eigenfaces available, much data is lost. All the information in the last 41 dimensions is neglected. Figure 5 displays the accuracy to which an image may be reconstructed when different amounts of eigenfaces are used.

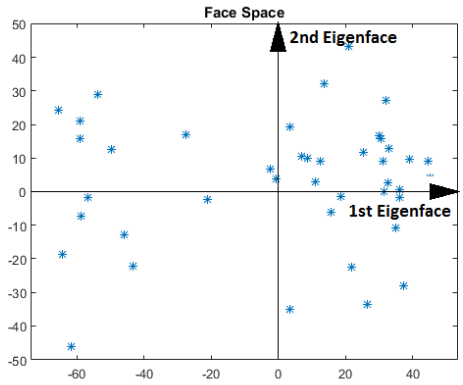

Fig. 4. A graph of two-dimenstional space represented by the two most significant eigenfaces from the dataset. Each blue dot represents the scalar projection of a training image onto the eigenfaces.

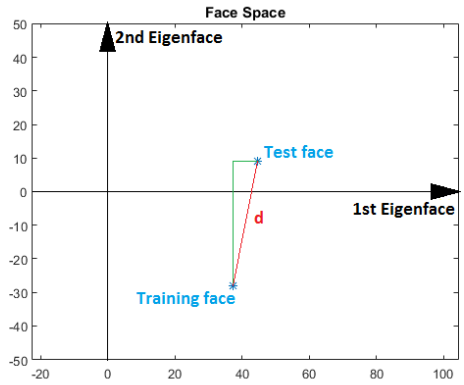

Fig. 6. A graph of two-dimensional eigen-space. This demonstrates the method of facial recognition using eigenfaces, wherein training faces and test faces are projected onto eigenfaces, and the distance is found between them. The minimum distance is the best match.

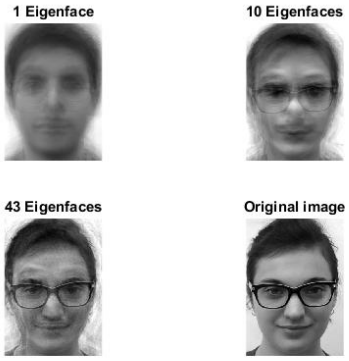

Fig. 5. Reconstruction of a test image using various eigenfaces. The maximum number of eigenfaces calculated, 43, does not convincingly look like the original image, which demonstrates the loss of data using the PCA method.

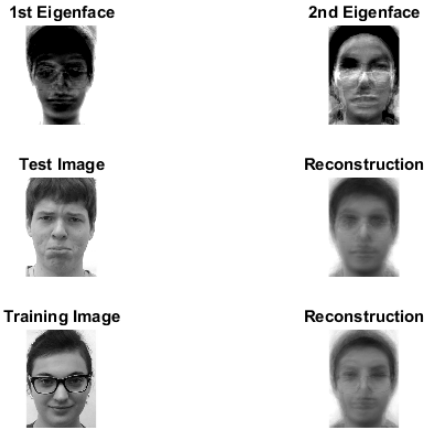

Fig. 7. The eigenfaces, test image, training image, and their projections onto the eigenfaces (labeled 'Reconstruction') that correspond to the graph in Figure 6.

Applying these concepts to facial recognition is quite simple. Since each image can be represented as a point in eigenspace, comparing the similarities of two images is simply a matter of finding the Euclidean distance between two points, as Figure 6 shows. (Incidentally, Figure 7 shows the corresponding images to the diagram in Figure 6). The figure shows the case of a 2-dimensional eigenspace, but the math remains quite simple for higher-order spaces, so that

$$d = \sqrt{(P_{1x_1} - P_{1x_2})^2 + (P_{2x_1} - P_{2x_2})^2 + \cdots + (P_{sx_1} - P_{sx_2})^2}$$

where P_{sx_1} is the sth scalar coordinate in vector P for training image x_1,P_{sx_2} is the same for the test image, and d is the Euclidean distance between the two image projections.

Once one has the data for the scalar projections of the training images onto the eigenfaces, one needs only to perform the same dot product operation on a test image, and find the distances between the test image and each of the training set. The training image the minimum distance away is the best match!

Pixel to Pixel Correlation

The second algorithm used is pixel-to-pixel correlation using the Pearson Correlation Coefficient and a correlation matrix C. For a pair of datasets $X = x_i$ and $Y = y_i$, with the same number of n elements, the Pearson Correlation Coefficient is defined as:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{N\sigma_x\sigma_y}$$

where μ_x,μ_y,σ_x, and σ_y are the means and standard deviations of the datasets. If we take the transposed row vectors of the training images x_i and put them in a matrix B such that

$$B = [x_1, x_2 ... x_i]$$

we can create the matrix A with the mean of each image subtracted from each element in that image along with its standard deviation divided out:

$$A = \frac{1}{N}[\frac{x_1 - \mu_1}{\sigma_1}, \frac{x_2 - \mu_2}{\sigma_2} \dots \frac{x_i - \mu_i}{\sigma_i}]$$

Now we can create the correlation matrix

$$C = A^T A$$

which contains elements of the correlation between the datasets ranging from -1 (anticorrelation) to 1 (positive correlation), with 0 as the non-correlated middle ground.

Finding the best match is now a matter of determining the maximum value in each column of the covariance matrix, and finding the index of the images which created it (the first image will simply be the number of the column). Of course, this requires that one first subtract out the indeces which contain an image's correlation with itself, which will always return 1.

IV. Comparison of Performance

The Eigenfaces method of facial recognition works reasonably well, particularly as more eigenfaces are used for identification. Figure 8 shows a graph of the accuracy of the algorithm as the number of eigenfaces used increases. The slope is very interesting, as the accuracy increases very quickly with only a few additional eigenfaces. In fact, it takes only 3 eigenfaces to achieve an accuracy of 78 percent within this dataset, ten faces to achieve 87 percent accuracy, and a maximum accuracy of 95 percent with all 43 eigenfaces. This suggests a tradeoff between accuracy and efficiency, which has decent results even at higher levels of computational efficiency.

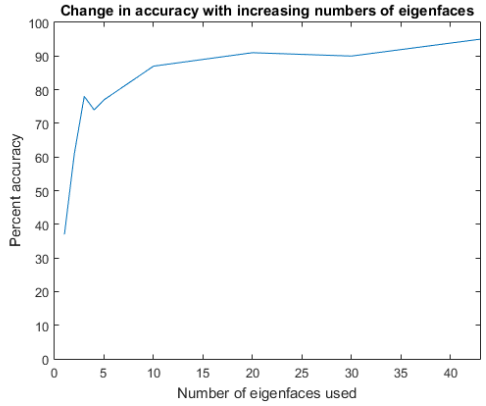

Fig. 8. A graph of the accuracy of the algorithm as the number of eigenfaces used increases. The slope is very interesting, as the accuracy increases very quickly with only a few additional eigenfaces.

Interestingly, this amount of accuracy was achieved despite different expressions and head positions, though all the pictures have similar lighting conditions and resolutions. Figure 9 shows a surprising success, and Figure 10 includes an example of two faces which were mismatched, where one can see the similarities between the eigen-reconstruction and the two compared faces.

Pixel to pixel correlation results in an incredible 97 percent accuracy with this dataset. This is likely due to the similar lighitng conditions and facial positions of the images, so the algorithm would not work well when scaled to different

Fig. 9. An example of a complex successful recognition. The head tilt and differing expression between the two images makes this an impressive achievement.

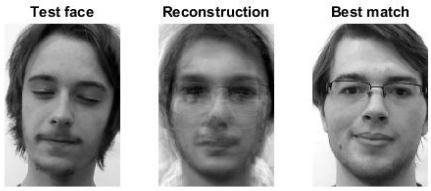

Fig. 10. Failure case for eigenfaces algorithm. Notice how the reconstruction looks visually similar to both the test face and the result face, so while the recognition was inaccurate, it is at least justified.

situations or more diverse datasets. However, for this set, it is an elegant solution requiring very little setup.

V. Conclusion

Eigenfaces is an interesting algorithm which demonstrates some of the more interesting powers of eigenvalue decomposition. Its accuracy is acceptable for demonstration, with the benefit that this particular algorithm is decent at recognizing people even with different expressions, a classically difficult problem in facial recognition. It would be interesting to further explore its performance with a training set that includes different lighting conditions or other obstructions. An exploration of the different variances present in each eigenface would also be intriguing, as the data contained in specific eigenfaces could potentially affect the performance of the algorithm.

References

[1] http://www.popularmechanics.com/technology/how-to/g1731/8-weird-ways-people-are-using-facial-recognition-software/

[2] https://en.wikipedia.org/wiki/Facial$_r ecognition_s ystem$