Facial Recognition Using the Eigenfaces and Fisherfaces Algorithms

Naomi Chiu and Vienna Scheyer

April 5, 2018

1 Abstract

In this paper, we discuss how linear algebra techniques are used in two facial recognition algorithms, the Eigenfaces algorithm and the Fisherfaces algorithm. Facial recognition is the process of comparing a given test image to a predetermined set of training images and identifying the image from the training set which matches the test image most closely. The math behind the two algorithms is explained and our own implementation of these algorithms is described.

2 Introduction

In the world today, facial recognition has a variety of applications and uses, from security systems to fun social media add-ons. Facial recognition can be used for a variety of applications, from security systems to fun social media add-ons. Despite being a prevalent feature used in the world around us, there is a general misconception that it is a complex and highly technical process but in actuality it's processes can be explained using linear algebra.

A facial recognition algorithm compares a given image to a set of training images and determines which of the training images is most similar to the given image. The Eigenfaces algorithm essentially creates vectors that represent the data with reduced dimensions and then compares the test image to the training data set to find the corresponding training image with the smallest difference from the test image. The Fisherfaces algorithm takes this a step further by using multiple images for each person sorted into a "class" in the training data set. Rather than comparing image to image, the algorithm compares the test image to the classes in the training set to find the closest match.

In section 3 of this paper, we describe the math behind each of the algorithms in this paper, we describe how we implemented the Eigenfaces and Fisherfaces algorithm. how we represented the images with matrices, explain the eigenfaces and fisherfaces algorithms on a conceptual level, dive further into the mathematical formulas behind each algorithm, and then give an overview of how we implemented these algorithms for our facial recognition program.

3 Algorithms and Justification

3.1 Concepts

In the Eigenfaces algorithm, we represented the images as matrices, compressed the image data by using Principal Component Analysis (PCA) and choosing a limited number of eigenvectors, calculated weight vectors to represent each image, and finally found the training image that had the least difference from the training image. In the Fisherfaces algorithm, we added on to the Eigenvectors algorithm. In this case, instead of comparing individual images, we used multiple images for each person in the training data set

and grouped the images into classes containing images of people. Then we calculated the class with the least difference from the test image to find the matching person. Before we could implement the Eigenfaces and Fisherfaces algorithms, we needed to represent the images numerically as matrices. We started with a set of images that we imagined to be stacked on top of one another, and we created a three dimensional matrix. Since Eigenfaces and Fisherfaces use vector operations, we needed to arrange our data into linear sets so we reshaped our three dimensional matrix into a two dimensional matrix. We converted each image from a square matrix into a column vector, reducing our three dimensional matrix into a two dimensional matrix where the rows contained the values for each of the pixels and each column represented a different image from our training set as seen in Figure 1.

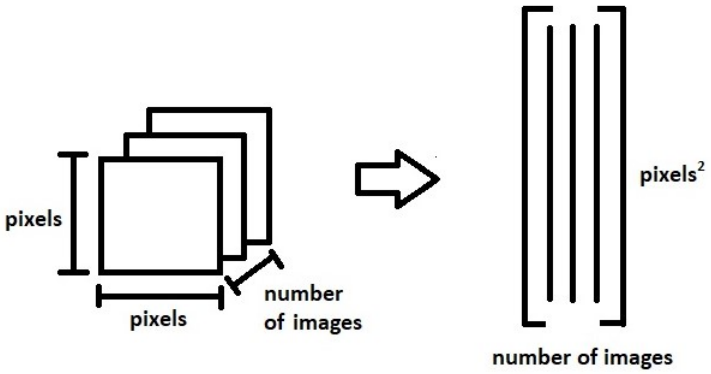

Figure 1: *Given a 3D matrix containing each image and all of its pixels, the matrix can be reshaped into a 2D matrix where the number of rows corresponds to the number of pixels per image and the number of columns is equal to the number of images within the data set.*

3.2 The Eigenfaces Algorithm

For the Eigenfaces algorithm, we compressed the data into fewer dimensions and then analyzed the data to find the directions in which individual images vary the most from each other. This method is called Principle Component Analysis (PCA) as the most important directions are taken and used to represent the whole image as seen in Figure 2. The fewer principal components we choose the less computational space it takes up but the less accuracy it retains, so we wanted to take an appropriate number of components to adequately represent the data within our limits.

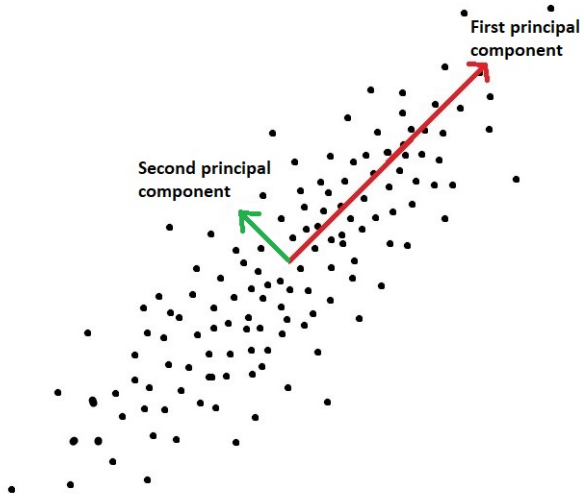

Figure 2: *Principal Component Analysis (PCA) is achieved by using the directions in which the greatest variance in a cloud of data occurred to represent the data with fewer components. By decreasing the amount of components used to represent each data point, reduced dimensionality is achieved allowing for the data to be processed by computing programs with limited computing power.*

In order to determine the directions, first we calculated the average face of the training data matrix, M_{train}, by finding the mean value of each pixel from the training data. We subtracted this average face from all of the faces in the training data to normalize the data and leave behind features that make each face unique to get Φ.

$$\Phi = M_{train} - mean(M_{train}) \tag{1}$$

From this matrix Φ, we created the covariance matrix $\Phi\Phi^T$ to analyze the relationships between independent pixels which are indicated by the eigenvectors of the matrix. However, the covariance matrix $\Phi\Phi^T$ is very large and has some eigenvalues of zero which are not helpful for PCA, so we consider the covariance matrix $\Phi^T\Phi$. The covariance matrices $\Phi\Phi^T$ and $\Phi^T\Phi$ have the same eigenvalues but different corresponding eigenvectors which we calculated using the Singular Value Decomposition or SVD of Φ as seen in Equation 2

$$\Phi = U\Sigma V^T \tag{2}$$

where Σ is a matrix of the non-zero eigenvalues of $\Phi\Phi^T$ and $\Phi^T\Phi$, U is a matrix containing the eigenvectors corresponding to $\Phi\Phi^T$ and V is a matrix containing the eigenvectors corresponding to $\Phi^T\Phi$. The eigenvectors in U are the eigenfaces which we need to represent all of the training faces. The matrix U has as many eigenvectors as the number of images in the training set but only the eigenvectors corresponding to the greatest eigenvalues are needed for a reasonable representation of the individual training images. While it is true that the more eigenvectors we use the more accurate the representations will be, it is important to remember that the point of finding these eigenvectors is for dimensionality reduction so not all of them should be used. The set of the K best eigenvectors we choose becomes the eigenfaces that we use in linear combinations to reproduce the data. We took the dot product of the eigenfaces and the normalized image vectors as shown in equation 3.

$$\vec{w} = W_{eig}\Phi \tag{3}$$

The resulting matrix contains the corresponding weight vectors for each of the images within the training set. Where the original image vectors had components equal to the number of pixels in each picture, the weight vectors have components equal to the number of eigenfaces chosen thus achieving reduced diensionality. Once we calculated the weight vectors of the training images, the training section of the algorithm was complete and we were ready to try it with test images. For the algorithm to compare between the test image and the training images, the test image must also first be reshaped into a column vector, normalized by subtracting out the average face and projected into the eigenspace with the dot product, in the same way that we projected the training images. Once we calculated the eigenface weight vector for the test image, we found the euclidian distance, d, between the weight vector for the test image and the weight vectors of the training images as shown in equation 4.

$$d = \sqrt{(\vec{w_{test}} - w_{train})^2} \tag{4}$$

The euclidian distances of all of the training images from the testing image are stored in the resulting matrix, and the training image with the minimum euclidian distance is the one that most closely matches the testing image.

3.3 The Fisherfaces Algorithm

The Fisherfaces algorithm relies on the Eigenfaces PCA algorithm for image dimensionality reduction. However, Fisherfaces incorporates an additional step in which the training images are sorted into classes, which represent individuals and the given testing images are compared to the classes rather than to individual images. This is achieved as the algorithm finds the directions among the data that maximizes the differences between classes or the between class scatter and minimizes the differences within each class or the within class scatter. The scatter matrices function similarly to covariance matrices, however the data is not normalized and the magnitudes of the directions indicated by the scatter matrix are approximations. This is illustrated in Figure 3 to show that data can be reduced to vectors pointing in perpendicular directions to achieve these effects. This is useful because, when we maximize between class scatter, we can represent all the compressed data points along a single vector while still maintaining distinction between classes.

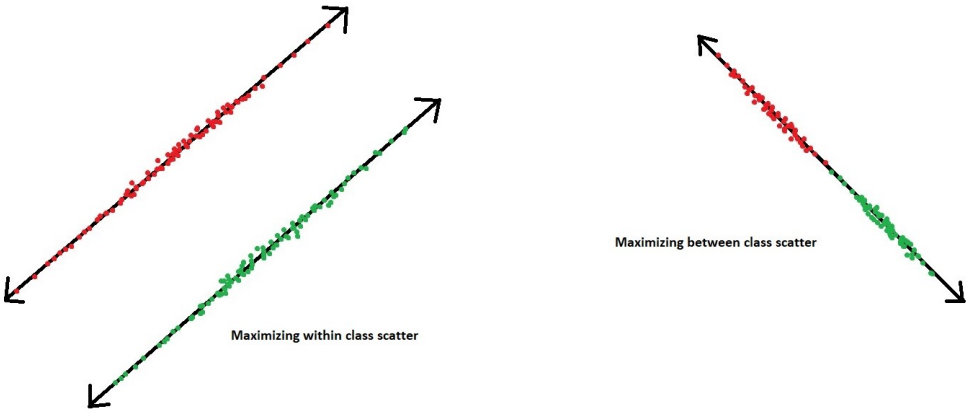

Figure 3: *On the left side, the data is compressed along vectors that maximize the within class scatter where the red points belong to Class 1 and the green points belong to Class 2. On the right, the data is compressed along a vector that maximizes the between class scatter, separating different classes as much as possible from one another.*

To account for this additional step, the training set for the Fisherfaces requires more pre-processing as several images of each individual must be included to form classes. Once the images are sorted by into

classes, we found the average face vector from each class in addition to the overall average face of the entire data set. Then we projected the training images into eigenspace through the same steps that we used in the Eigenfaces algorithm. Using the eigenface weights to represent the images, we calculated the between class scatter matrix, S_b, using the following equation

$$S_b = \sum_{i=1}^{c} N_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{5}$$

where c is the number of classes within the training set, N_i is the number of images within class i, μ_i is the eigenface weight vector representing the average face of class i and μ is the eigenface weight vector of the overall average face across all of the classes. The resulting matrix from this operation should be a square matrix with dimensions equal to the number of eigenfaces chosen to represent the data. Next, we calculated the within class scatter matrix, S_w, using this equation

$$S_w = \sum_{i=1}^{c} \sum_{x_k \in x_i} (x_k - \mu_i)(x_k - \mu_i)^T \tag{6}$$

where x_k is the eigenface weight vector for each image within class i and μ_i is the eigenface weight vector of the average face for class i. The resulting matrix should be of identical dimensions as S_b. From these two matrices, S_b and S_w, we found the generalized eigenvectors with linear algebra by using Equation 7

$$S_b w = \lambda S_w w \tag{7}$$

where w is a matrix of the generalized eigenvectors that correspond to the eigenvalues stored in λ. The eigenvectors w are the ones that we used to project images from the eigenspace into the fisherspace. Since the comparison of images using the Fisherfaces algorithm occurs within the fisherspace, the fisherfaces are defined as the dot product of the eigenfaces calculated at the beginning, also called W_{PCA}, and the w eigenvectors, which are also called W_{FLD} as seen below. W_{FLD} stands for Fishers Linear Discriminant, which is the type of linear algebra we are using here.

$$W_{fish} = W_{PCA} * W_{FLD} \tag{8}$$

Using the fisherfaces, the training images can all be projected into the fisherspace and the training of the algorithm is complete. For testing, the testing image is also projected into the fisherspace as a vector of weights for each of the fisherfaces and the euclidian distance is once again calculated between the test image and the training images. The matching face is returned as the testing image which has the minimum distance from the test image.

4 Comparison of Performance

After learning about the two algorithms, we implemented them ourselves to observe the behavior and performance of each algorithm and to assess and compare the performance between the two. In our own implementation of the Eigenfaces and Fisherfaces algorithms, we used identical data sets composed of 172 images, each with a 256 x 256 pixel resolution, that were divided into 43 classes with 4 pictures per person. For the Eigenfaces algorithm, the data was divided so that the odd numbered pictures were in the training set and the even numbered pictures were in the testing set. For the Fisherfaces algorithm, the fourth picture

from each class was removed for the testing set and the other three remained for the training set. Given these training and testing sets taken from the same original data set, our results, as seen in Table 1, indicated that the Eigenfaces algorithm outperformed the Fisherfaces algorithm in every aspect. The training time indicates the time that the MATLAB script of the algorithm took to process the training set and calculate the corresponding eigenfaces or fisherfaces. The testing time indicates the time that the algorithm's MATLAB script took to identify a single test image and return the matching image from the training set. The accuracy of each algorithm was calculated using string comparisons of the corresponding names for each picture in MATLAB.

Table 1: *Experimental data collected from our implementation of the Eigenfaces and Fisherfaces algorithms. The training time indicates the amount of time that the algorithm took to process the training set and form the eigenfaces and fisherfaces, respectively. The testing time indicates the amount of time the algorithm took to recognize a single test image and the accuracy is the percentage of the images from the testing set that the algorithm correctly identified.*

	Eigenfaces	Fisherfaces
Training Time	0.74 seconds	0.90 seconds
Testing Time	0.05 seconds	0.07 seconds
Accuracy	95.35%	93.02%

While the Fisherfaces algorithm was expected to perform better, further research indicated that the conditions in which the algorithm exceeded the Eigenfaces algorithm was not met in our implementation as our training set was not as deep as preferred. We found that the Fisherfaces algorithm worked best when there were at least four images per class in the training set, but in ours we only had three pictures for each class. As the Fisherfaces algorithm compares test images to the classes defined within the training set, deeper classes would allow for a more wholistic representation of the class which may not have been achieved with the limited training images that we had at our disposal. The Eigenfaces algorithm did not encounter this problem as it simply compared the test image to each of the training image, looking for the closest picture instead of the closest individual. If our data set had included a greater number of images per class with more variety between each image, we believe that the Fisherfaces algorithm would have out performed the Eigenfaces algorithm. In terms of the algorithmic run times, it is understandable that the Fisherfaces algorithm would take longer as it performs the same operations as the Eigenface algorithm with additional steps afterwards to account for the data classes.

5 Conclusion

Overall, we learned how to apply linear algebra concepts such as matrix reshaping, data dimensionality reduction, PCA, and SVD. In the Eigenfaces algorithm, we compared a test image to a set of training images and then in the Fisherfaces algorithm we extended that further to compare a test image to classes of training images. Based on our research, we learned that Eigenfaces is very useful to find a specific image in the training set while Fisherfaces is useful when you want to match the test image to a person. In other words, fisherfaces is capable of accounting for more variance in lighting or facial expression than eigenfaces. However, in our implementation it turned out that we didn't have enough data to prove that fisherfaces can have a higher accuracy than eigenfaces. In the future, we could try to run fisherfaces with more images of each person and see at what point fisherfaces surpasses eigenfaces in accuracy. Additionally, we could obtain a data set that includes images with more variation in lighting and observe whether our fisherfaces algorithm can handle that lighting variation better than our eigenfaces algorithm. Researching and implementing these algorithms was an effective process for us to implement our linear algebra knowledge in a practical way.