Facial Recognition Design Report

Willem Thorbecke and Izzy Harrison

January 16, 2017

1 Executive Summary

This report was written for the purpose of explaining how to implement two different facial recognition methods with the use of linear of algebra. Additionally, we will compare the effectiveness of these two algorithms; one which is very simple and another which produces comparatively better results, but still leaves much to be desired. The simpler method compares the 'distance' between two images, using Euclidean distance, in order to find a match. The more advanced method uses Eigenfaces.

In order to better understand our explanations of these algorithms we have provided the Background and Terminology section that explains vocabulary and concepts vital to understanding these algorithms. Following the background and terminology section we have provided the reader with a thorough description of our implementation of the two different algorithms in addition to the reasons why the Eigenface method may be superior to the Euclidean distance method. This information can be found in the Algorithms and Justification section of this report. After explaining how our two algorithms work, we will explain their trade-offs. This will be done through quantitative engineering analysis by interpreting various tests. This quantitative analysis will help explain the implications of each method's test results. This can be found in the Comparison and Performance section of this report. A final analysis of this report can be found in the Conclusion section where we will discuss the key takeaways of this report and call it a day.

2 Background and Terminology

In order to understand our algorithms, it is necessary to understand the linear algebra concepts behind them. Below we will define the key terms and concepts that we will reference throughout this paper.

Facial Recognition:

A facial recognition program is able to take in an input image and figure out which image in its database the input image is most similar to. This can be done using many different approaches, some pay attention to specific features (such as eyes, nose, and mouth), where as others (such as Eigenfaces and Euclidean distance) deal with the image as a whole.

Mathematical Notation:

$n \times m$: a matrix that is n rows by m columns

Covariance matrix:

A covariance matrix is a matrix in which each element is centered and averaged, followed by this resultant matrix being transposed and then multiplied by the pre-transposed result. This creates an $n \times n$ which makes it possible to find the eigenvalues.

Eigen Values and Vectors:

Consider the expression $Av = \lambda v$, where A is a matrix, λ is a scaler, and v is a vector. If this expression is satisfied, λ is considered an eigenvalue, and v is an eigenvector.

Principle Component Analysis (PCA):

PCA is a method we will use as part of our Eigenfaces method. PCA can be though of as factoring but for matrices. it allows you to represent a matrix (or in our case, a face) as a linear combination of Eigenfaces.

Euclidean distance:

the Euclidean distance is the ordinary distance between two points, essentially the same as the Pythagorean Theorem, but applied to n dimensions

Training Set:

In order to perform facial recognition you need a database of baseline images to match the input image to. The Training set is the set of images used to create eigenfaces and perform facial recognition with. Our training set uses seven different expressions of 43 people. All of the images were taken under the same conditions, are grey scaled, and are 360x256 pixels. see Figure 1 for an example of some of the faces we used.

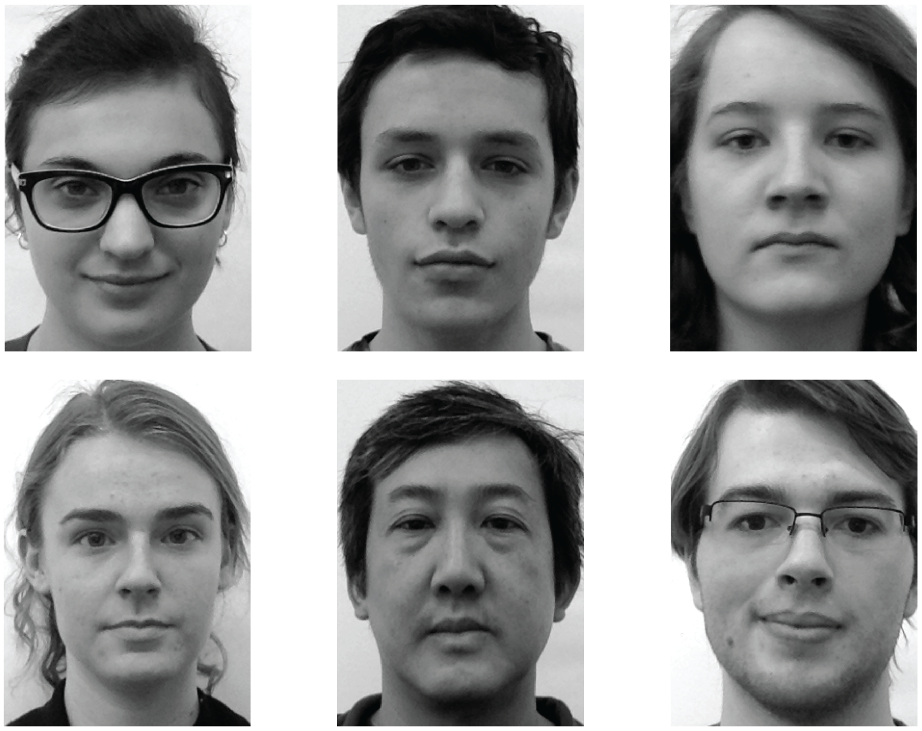

Figure 1: six of the faces from our training set

Eigen Faces:

Eigenfaces are the set of faces that are created from the Eigenvectors of the training set images, these images are then used for Eigen decomposition. See Figure 2 for an example of 6 of our eigenfaces. The top ones are some of the first and most importance faces (correspond to the larger eigenvalues) whereas the bottom three are less significant.

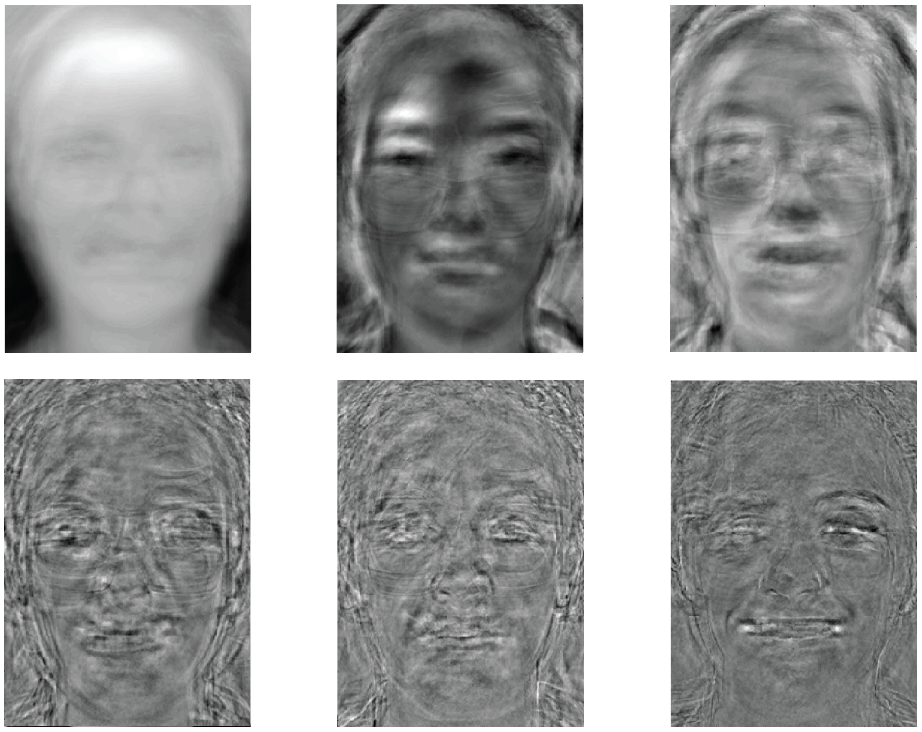

Figure 2: six of our 301 Eigenfaces, the top ones are some of the first and most importance faces (correspond to larger eigen values) whereas the bottom three are less significant. we will explain this in more detail in our Algorithms section

Eigen face decomposition:
Eigenface decomposition is used to represent an image as a sum of different amounts of the Eigenfaces, which then makes it possible to do facial recognition using this method.

Face Space:
A coordinate system in which each component is defined as an Eigenface.

3 Algorithms and Justification

Having introduced you to terminology, we will now layout the approaches we used to produce each facial recognition algorithm. Each algorithm takes an input image and finds an image within the training set that is a plausible match to said input image.

I. ***Euclidean Distance*** The first and most basic algorithm we tested was Euclidean distance. To understand this method, you can think of each $m \times n$ input image as a vector in mn dimensional space, as well as thinking of each image in the training set as a vector in mn dimensional space. From there you just need to find the distance between the input image and each of the training set images using the euclidean distance formula:

$$||q - p|| = \sqrt{(q - p) \otimes (q - p)}$$

where q is the input image, and p is a training set image, and both are reshaped into $mn \times 1$ vectors. the algorithm returns the image within the training set that is "closest" to the input image in nm dimensional space.

II. ***EigenFaces***

The more advanced algorithm that we tested was Eigenfaces. This algorithm involves taking a training set of images where each image is $m \times n$ pixels and then representing each image as a point in an mn-dimensional vector space. Thus each image will be represented as a $mn \times 1$ vector.

The Eigenface algorithm involves plotting each image in mn-dimensional vector space and then finding the principal components of the distribution of these points. The principal components are known as Eigenvectors in a traditional sense, but in the case of the Eigenface algorithm they are known as Eigenfaces. Refer to Figure 2 for an example of some Eigenfaces. In order to find the Eigenfaces of our training set of images we need to construct a covariance matrix. This can be constructed by first finding the mean image vector of all the image vectors in our training set and then subtracting that from all the image vectors in the training set. A matrix of all the centered image vectors can then be created as such:

$$A = [r_1, ..., r_f]$$

where each r is an image vector from the training set of f images with the mean image of the training set subtracted out.
The covariance matrix is then:

$$C = A^T A$$

The eigenvectors of the covariance matrix define the eigenfaces of our image training set. The eigenvectors associated with the largest eigenvalues are the most important eigenfaces because they correspond with the largest variation in our training set of images. For this reason it's possible to accurately represent our image set using only the most important (largest) eigenvectors/eigenfaces. Doing so is a form of image compression and can be useful for a larger training set of data. In the case of this project, our training set consists of 301 images and compression would only be necessary for a much larger training set of images.

After the eigenfaces are defined, a facespace can be created using each eigenface as a principal component. This is demonstrated in figure 3 with only two eigenfaces, but in our case we used 301 eigenfaces.

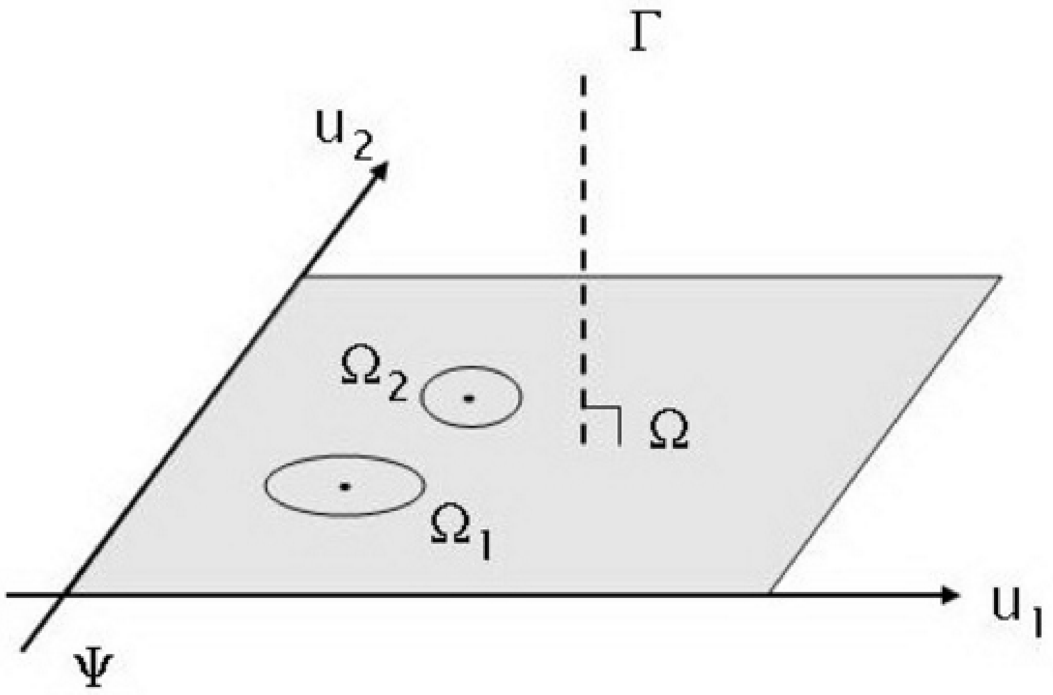

Figure 3: u_1 and u_2 correspond to an eigenface component and form a facespace. The origin is the mean eigenface, and each Ω represents an image of a face being projected onto the facespace defined by u_1 and u_2 (source: http://www.scholarpedia.org/article/Eigenfaces)

Once a facespace is defined comparisons can be made between an image in the training set and a new, user inputted image. This can be done by projecting a new image onto the defined facespace and comparing the euclidean distance of this vector between the image vectors of each of the training set images projected onto the same facespace. The projection vector for any image can be defined as:

$$\Omega = U^T(\Gamma - \Psi)$$

Where U is defined as a matrix of the eigenvectors of the covariance matrix, Γ is defined as a $nmx \times 1$ vector of a given image, and Ψ corresponds to the mean vector image of the training set images. Thus Ω is an $nmx1$ image vector.

The Euclidean distance between two image vector projections can be defined as the distance between two vectors in Euclidean space, as explained previously. This can be found using the Euclidean distance formula:

$$||q - p|| = \sqrt{(q - p) \otimes (q - p)}$$

Where q and p are two image projection vectors.

4 Comparison of Performance

In principle we expected the Eigenface algorithm to work much better than the Euclidean Distance algorithm due to the fact that the Eigenface algorithm was heralded as the first facial recognition algorithm ever, and a great baseline facial recognition algorithm. We did little to no background research on the viability of Euclidean distance being used as a facial recognition algorithm and didn't expect much out of its performance.

In terms of speed of each algorithm, the Euclidean distance algorithm was about five times quicker than the Eigenface algorithm. This was to be expected since, although the Euclidean distance algorithm was used as its own form of facial recognition, a Euclidean distance calculation was also used as the final step in the Eigenface algorithm.

In order to compare the performance of the algorithms we did two tests involving two sets of image data. The first test involved taking the initial image training set (containing eight images of all the students and instructors) and then removing a single image of each person to create two separate image sets – one containing all the initial training set images minus the removed images and one containing all the removed images. Then using the removed images as input images and the stripped data set as the new training set, we test the accuracy of each algorithm. As stated initially the Euclidean distance formula was generally five times faster than the Eigenface formula. In terms of accuracy, surprisingly, the Euclidean distance algorithm was correct 97% of the time whereas the Eigenface algorithm was correct 81.4% of the time. We will discuss the implications of these results after we go over the results of the second test.

The second test involved using the original image training set to create the Eigenfaces. We then used an all new image data set consisting of head shots of all the students from the original image training set and used this new image data set as input images with the hope of finding a match from within the original image training set. In this test the speed was the same as the first test, but the accuracy dipped significantly for both algorithms. The Eigenface algorithm was correct 34% of the time and the Euclidean distance algorithm was only accurate 5% of the time.

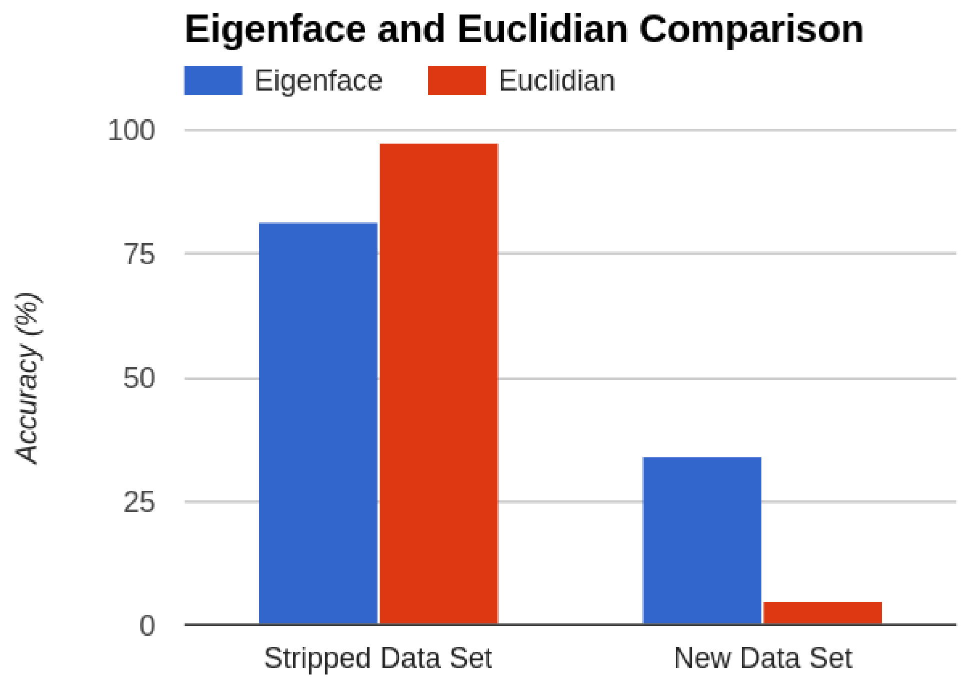

Figure 4: shows the accuracy of our algorithms. Note that the Euclidean algorithm performs better using images taken under the same conditions as the training set, and the Eigenface algorithm performs better using the input images taken under slightly different conditions. Note that both perform significantly worse when using the second set of input images, due to discrepancies in lighting and positioning.

In our third test, we experimented with the number of Eigenfaces used in our algorithm. This could have an impact on our results because often the Eigenfaces corresponding to the largest eigen values correspond more to lighting changes than facial features. We found that the accuracy resembled a bell curve, and that it is best to take out the first 15 Eigenfaces in order to optimize the accuracy. see figure 5 for a visualization of the data.

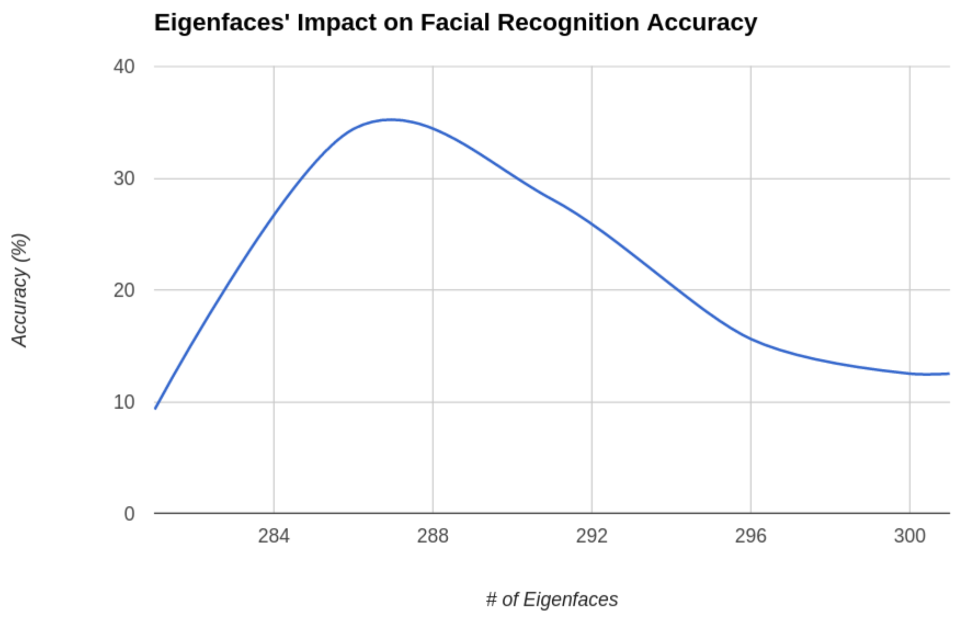

Figure 5: There is an optimal amount of Eigenfaces to use in order to get the best accuracy. It is best to ignore the Eigenfaces corresponding with the 15 largest eigen values.

Our 4th test experiments with the number of facial expressions from each person included in our training set. see figure 5. We tested algorithm accuracy by including either 7, 5, 3, 2, or 1 expressions per person in our training set. This test used images taken under the same conditions as the training set images. As expected, the accuracy went up as we used more expressions, but overall, the euclidean distance algorithm was more accurate.

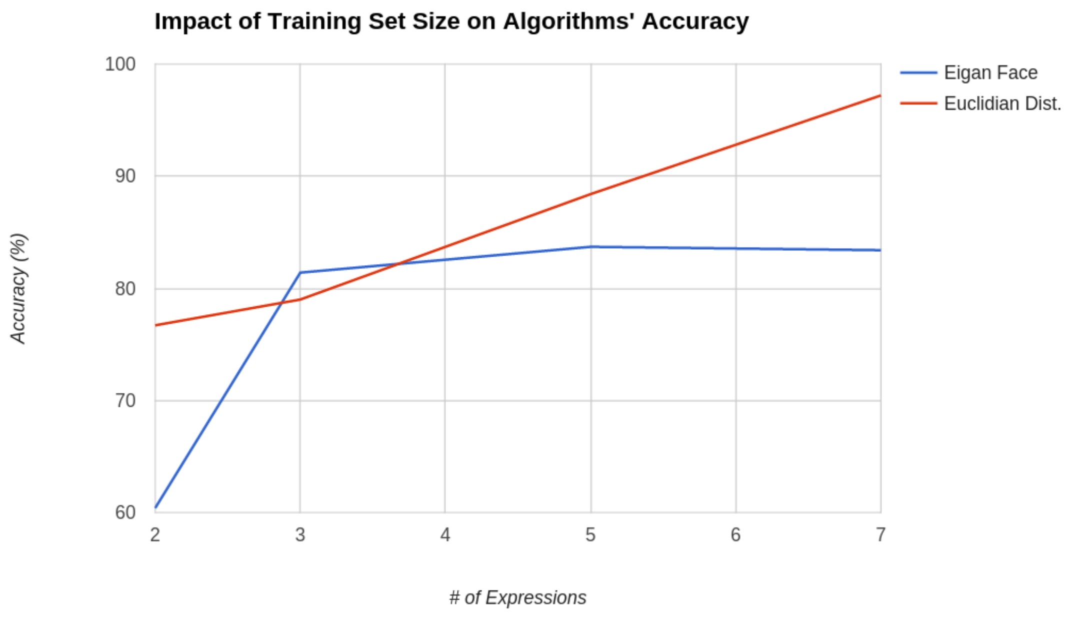

Figure 6: accuracy using 7, 5, 3, 2, or 1 expression per person included in our training set. This test used images taken under the same conditions as the training set images. As expected, the accuracy went up as we used more expressions, but overall, the euclidean distance algorithm was more accurate.

The tests imply different things about each algorithm. One thing to note was that that all the images in the initial training set data were taken at the same time under the same lighting conditions, with the same camera. Thus in the first test, when a comparison was being made between a removed image and the stripped training set, the lighting, contrast, and image quality were all being controlled thus allowing for the maximum variation to appear because of differences between the faces. Variation would increase when two of the same faces had different facial expressions, or if each face wasn't precisely centered through each photo. This is different than the second test in the sense that the training set images and the input images were taken at different times under different light conditions. Thus a lot of variation came from lighting and contrast alone. While both algorithms did very poorly under these test conditions, the Eigenface algorithm did much better under conditions with higher variance. This was due to the fact that the Eigenface algorithm broke down the training set into Eigenfaces and then calculated the Euclidean distance between images using a Eigenface defined face space (the Euclidean distance algorithm skipped this step), this allowed for the Eigenface formula to overcome the larger variations in the second test when compared to the Euclidean distance algorithm alone.

5 Conclusions

The Eigenface formula takes longer but is better for cases in which there is a larger variation in lighting and contrast between an input image and a training set, this is because these large changes in variations can be overcome by not including the most important Eigenfaces. The Euclidean distance algorithm is faster and performs better when the input image and training set images were taken under the same conditions. So, if the images you are trying to recognize were taken under the same settings as your training set images, it might not be worth it to use the Eigenface algorithm since it is less accurate and slower.

The next steps in this process would be to develop an algorithm that can process an image taken under different conditions and correct it so that it has similar contrast values to the training set images, and then apply that using the Euclidean distance algorithm. This would more accurately assess how the performance of the Euclidean distance algorithm compares to the Eigenface Algorithm for any input image.