A Face in the Crowd: Exploring Facial Recognition

Kevin Zhang
Quantitative Engineering Analysis I
Olin College of Engineering Spring 2016
Needham, MA

I. Introduction

The concept of facial recognition is an intriguing one. The idea itself is intuitive, but when thinking of how to match one face with another, the field broadens to a wide spectrum of different options and possibilities. There is always a discussion to be had on the method of recognizing faces and the performance of each approach with regards to how well it can match images. One could attempt to match them based on physical features, such as eyes and cheeks, or also by shading and lighting on the surface of the face, most likely with different outcomes in terms of how well the faces are matched. Mathematically speaking, there are various venues through which matching images can be done. In this paper, we will explore different mathematical algorithms for facial recognition and consider the results of the different algorithms with respect to particular standards. Specifically, we will be looking to investigate the wide range of possible algorithm approaches, thus we will look at the simplistic Euclidean Distance algorithm, the complex Image Correlation algorithm, and the efficient Eigenfaces algorithm. We hypothesize that out of the three algorithms, all will perform quite well in facial recognition, but the Eigenfaces algorithm will perform more effectively with regards to speed, space, and accuracy.

We will first begin by an examining different approaches to facial recognition, going over key concepts and background information and presenting the three algorithms under evaluation: the Euclidean Distance algorithm, the Image Correlation algorithm, and the Eigenfaces algorithm. Then we will proceed to the experiments, first looking into the standards upon which performance was evaluated on, and then going over the data that explains the performance of each algorithm under different conditions. The experiments will test the algorithms under different data set sizes and different lighting conditions. Each of the three standards - speed, memory, and accuracy - will be scrutinized within the four experiments for the three different algorithms, and the results will be collected and interpreted. Based on the evidence, it is apparent that all three algorithms performed quite well when lighting conditions were the same for the images, but their accuracy dropped considerably when presented with images of different lighting conditions. Our hypothesis was proven correct, but in an interesting way. While the Eigenfaces algorithm was indeed more efficient in memory and either equal or stronger in terms of accuracy, it seemed to be lacking significantly in speed. However, upon closer evaluation, the evidence trends suggest that the Eigenfaces algorithm can be beaten by more simplistic algorithms when the data set is small enough, but when the data sets become larger and more complex, the Eigenfaces algorithm becomes more and more efficient.

II. Approaches to Facial Recognition

Before we present the algorithms being evaluated, various key concepts and considerations that contribute to the results of facial recognition algorithms must be understood. We will outline the most important ideas behind each algorithm and introduce each algorithm as we progress along.

A. Images as Matrices of Data and Space

When comparing two images to determine their similarity, as in facial recognition, images are thought of as a giant matrix of pixels. Each pixel could be a value of between 0 and 255, where 0 is black and 255 is white, or they could be normalized to be between 0 and 1. As matrices of data in their raw form, these images can already be compared. For instance, a simple method would be to compare each individual pixel with its respective pixel on the other matrix and seeing how that relationship compared to other images. Another way might be to centralize the individual pixel values around an average, and then examining the deviation from that average. In addition, since images can be converted to matrices, this explicitly means that all operations that apply to matrices are usable on image matrices. This means that determinants, inverses, and down sampling operations can all be performed on images to manipulate them in a way that is more easily computed on or examined.

B. Brightness and Contrast

A special concept that needs to be addressed early on is the idea of brightness and contrast in an image. It is completely within the realms of practicality and actually quite often that two images will not be taken with the same lighting intensity or angle, thus two images that visually look exactly the same could actually differ quite a bit mathematically. To prevent this, there are some solutions to pre-process images before they are operated on to ensure that these factors minimally impact the facial recognition program. To address brightness, the simplest method is just multiply the entire matrix of pixels for an image by a scalar such that the scale of the values is more similar to that of other images. Contrast is more tricky, as it is a range of values and a measurement of how wide or narrow the range of a picture is, with the wider range resulting in higher contrast in an image. To normalize contrast, the best way is to determine the upper and lower bounds of intensity values and then use a linear system of algebraic equations to solve for the appropriate operations needed to manipulate the data to fit those boundaries. Doing these steps will ensure that facial recognition programs are much more accurate and are most representative of the actual faces and not the environment in which the images were taken.

For the purposes of this project, we considered both conditions where two images had the same brightness and contrast as they were taken in the same place at about the same time, and also where two images had different brightness and contrast as they were taken in different times and locations. This is to evaluate robustness and rigor in the algorithms when placed under different situations. Pre-processing is never used for any portion of the experiments.

C. Euclidean Distance

Another way of thinking of about images is as a vector of data in space, where each pixel value corresponds to a dimensional value in the pixel space. This means that images can be represented as points in a very high dimension. It would be then intuitive to believe that two images in the image space could be evaluated based on their distance from each other. This idea provides an interesting method through which to determine how closely two images are related by looking directly at the pixels in a physical space. Figure 1 is a visual of what pixel space might look like for two images, and provides a view of what the distance between two images could be envisioned to be. In the figure, the two images are represented as two points in a some arbitrary high dimension space, Point 1 and Point 2, denoted in red. The distance between the two is e, which is found by the Pythagorean Theorem of c and d, and d is found by the Pythagorean Theorem of a and b.

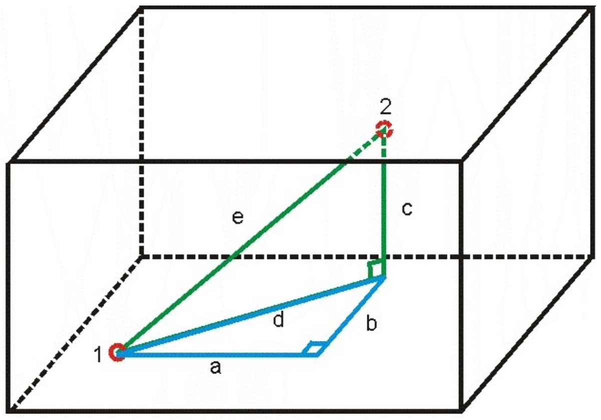


Fig. 1. A visual representation of pixel space and the distance between two images. The two images are represented here as two points, Point 1 and Point 2. The distance between them is e, which can be calculated as the hypotenuse of the right triangle cde, of which d is the hypotenuse of the right triangle abd. Note that the distance is essentially Pythagorean theorem, denoting its ease of usage and direct approach to looking for image similarity.

Note that the usage of Pythagorean Theorem to find the distance between two points and using that as a metric of similarity is rather straightforward. This distance between two images in pixel space can be categorized under what is known as Euclidean Distance, and it is an intuitive but useful approach to matching up two images.

D. The Euclidean Distance Algorithm

This brings us to the first algorithm under consideration, the Euclidean Distance algorithm, one of the most straightforward approaches to measuring similarity between two images. This algorithm is very simple and is based on the idea that all images are points in N dimensional space, and a measure of similarity can be found by determining the distance between the image points in pixel space. The principal equation used to determine distance is:

$$d = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{M}(x_{ij} - y_{ij})^2} \tag{1}$$

where N and M represent a NxM image matrix, x_{ij} and y_{ij} represent corresponding coordinates in the dimensional space, and d is the distance between the points in pixel space. The equation is merely finding the distance between every single dimension in the space between the two images and summing them up to find the overall distance in the space between the two ”points”. This algorithm utilizes an extreme case of Pythagorean theorem, and basically brute forces the comparisons with a direct one-on-one measurement between every single pixel in an image with every corresponding pixel on the other image.

The Euclidean Distance algorithm is meant to be fast and direct, as in terms of coding there are few for loops to be created and very few lines of code. The simplicity of the algorithm is what allows it to be fast. Another aspect in which it is quite strong in is the accuracy, as it compares every pixel with another and finds the measurement of similarity. This allows for a pretty close approximation for the actual similarity between the two faces.

A serious issue with this algorithm is the fact that it only work well for small data sets, when computation is not the main limiting factor. Given that it uses every single pixel in its comparison, data sets that become increasingly large cause the algorithm to become ridiculously slow, and this could potentially scale exponentially. Another major issue is the space involved. This algorithm uses a lot of space to be able to perform its raw calculations. While there are few variables, the few variables that are there contain massive amounts of information, and it is totally in the realm of possibility that a data set could become too big and overflow the program’s memory storage. Space and scalability are two things that would have to be scrutinized when testing this algorithm.

E. Correlation

Diving deeper into the idea of images as matrices, we can use a more complex method of evaluating similarity by using the idea of correlation. Image correlation is looking mathematically at how related two images are. In this case the relation between two items is defined as their deviation from the norm. Two standards to define that deviation are the mean of the data set and the standard deviation of the data set. The mean of a data set is the average of all the values in a sample or in this case the average of all the pixel values in an image. It is defined by the equation:

$$\mu_d = \frac{1}{N}\sum_{i=1}^{N} d_i \tag{2}$$

Where μ_d is the average of the samples in a particular set of data, N is the number of elements in the data, and d_i is the ith element of the data. The average provides a reference point to see how far each element deviates from. Another useful metric is the standard deviation, which is the spread of the data around this average. It is defined by the equation:

$$\sigma_d = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - \mu_d)^2} \tag{3}$$

Where σ_d is the standard deviation of the samples in a particular set of data, and the other variables are as stated above. The standard deviation is finding the accumulation of all the deviations in the data. With these two piece of statistical data, one can find the correlation between two data sets by looking at how they compare in terms of their means and standard deviations. The most common and most effective way to measure the correlation between two pieces of data is through the Pearson Correlation Coefficient, which is defined as:

$$r_{xy} = \frac{\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)}{(N-1)\sigma_x\sigma_y} \tag{4}$$

where r_{xy} is the correlation coefficient between the two data sets x and y, x_i and y_i are the individual elements of the data sets, μ and σ represent the mean and standard deviation of the data sets respectively, and N is the number of elements in the data sets. The Pearson Correlation Coefficient is finding the sum of all the deviations between two data sets with respect to their means and then normalizing that with the standard deviations. In this way they can find how far the two images deviate from each other, generally on a scale from -1 to 1, with 1 being the same and -1 being the complete opposite. Image correlation is useful in that instead of just giving the direct approach of computing numbers, the Pearson Correlation Coefficient takes into account the various forms of deviation and compares similarity after having minimized variation due to statistical margins. It is a different approach that could perhaps yield more accurate results in facial recognition.

F. Image Correlation Algorithm

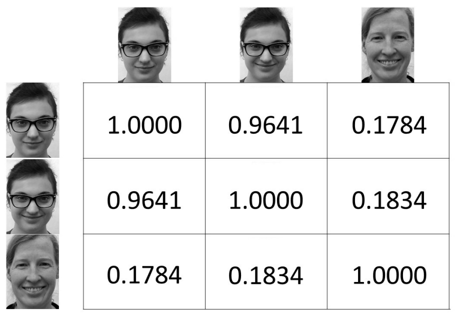
This allows us to introduce the Image Correlation algorithm. The algorithm is more complex than Euclidean Distance, and it is founded on the basis of the Pearson Correlation Coefficient. The idea that the image closest to the target image would have a correlation closest to one, and by using the Pearson Correlation Coefficient one could determine the two images that had the highest correlation, which would be an accurate representation of the highest similarity. Given a matrix A that contains N elements, the Pearson Correlation Coefficient is calculated as follows:

$$A = \frac{1}{\sqrt{N-1}}\begin{bmatrix} \frac{x_1-\mu_x}{\sigma_x} & \frac{y_1-\mu_y}{\sigma_y} \\ \frac{x_2-\mu_x}{\sigma_x} & \frac{y_2-\mu_y}{\sigma_y} \\ \vdots & \vdots \\ \frac{x_N-\mu_x}{\sigma_x} & \frac{y_N-\mu_y}{\sigma_y} \end{bmatrix} \tag{5}$$

$$I_{xy} = A^T A \tag{6}$$

$$P_{xy} = AA^T \tag{7}$$

where A is the matrix A after being centered with the mean and standard deviation, and I_{xy} is the image to image coefficient matrix for data sets x and y, and P_{xy} is the pixel to pixel correlation between the two data sets. Essentially the correlation matrix is found by centering the data and then taking the dot product of the matrix with itself, in matrix multiplication form. There are two types of correlation: Image to Image and Pixel to Pixel. Image to Image is looking specifically at comparing the images to the other images as entire matrices of pixels. Pixel to pixel compares images on an individual pixel level, comparing pixels across images for the entire image. An example of an image to image correlation matrix can be seen in Figure 2. Note that the diagonal is 1, which represents that the image is being compared to itself. The non-diagonals are symmetrical because it is the image being compared to the other in a different order.



	1.0000	0.9641	0.1784
	0.9641	1.0000	0.1834
	0.1784	0.1834	1.0000

Fig. 2. A visual of an image to image correlation matrix taken from three images in a data set. Note that the diagonal is merely the image being compared to itself, thus the number 1. The non-diagonals are symmetrical because the images are being compared to each other in a different order. For the two images that are very similar, the correlation is very high, almost 1, and for the two images that aren't very similar at all, the correlation is very low, almost 0. This shows how correlation relates to the similarity between two images. The objective of the Image Correlation Algorithm is find the non-diagonal that give the highest number for any image and a database.

As can be seen, when the two images being compared are very similar, the correlation coefficient is very high, almost 1. When two images that are not similar at all are being compared, the correlation coefficient is very low, in this case almost 0. This empirically shows how correlation coefficients relate to and can be used to find matches with images. The key here is to find the image from the database that will find the highest non-diagonal correlation coefficient, which will represent the image that is most heavily correlated with the target image.

The Image Correlation Algorithm is more complex and uses a different, more refined method to find its similarity measurement. In this manner, by accounting for deviations such as the mean and standard deviations, the Image Correlation Algorithm can find a more accurate match to the target image. Image Correlation can actually be done in two different ways, Image to Image Correlation and Pixel to Pixel Correlation. The Image to Image correlation is faster but is less accurate when compared to its sister. Pixel to pixel correlation is much more computationally intensive, but gives a much more accurate answer since it's comparing images down to their individual pixels rather than as general holistic images. In both instances, Image Correlation are most likely more accurate than Euclidean Distance.

The problem lies with the massive amount of computation required to perform Image Correlation. Image to Image Correlation, like Euclidean Distance, is already memory intensive, and Pixel to Pixel correlation is only possible with small data sets. Because of the massive amount of computation and space required to make the algorithm run, it is very slow. For larger and larger data sets, Image Correlation becomes less and less feasible, due to enormous amount of memory usage and the massive computational burden that could take hours to complete. The trade off that Image Correlation makes is additional accuracy at the cost of speed and memory.

G. *Eigenfaces*

The last approach we will examine is the Eigenfaces. Eigenfaces is a more sophisticated method of identifying images than the previous two, and thus will require more extensive explanation. By definition, "Eigenfaces" is the name given to a set of eigenvectors that are used in facial recognition to form a basis of vectors upon which images can be compared based on the coefficients of their linear combinations with the Eigenface basis. We will discuss the two main components of the definition first: eigenvalues and eigenvectors, and bases and spans of vectors.

Eigenvalues and eigenvectors are among the most popular and most powerful tools in the realm of facial recognition. To explain eigenvalues and eigenvectors, recall from linear algebra that a matrix of unknowns can be solved with the following formula:

$$Ax = b \tag{8}$$

where A is a matrix of coefficients, x are the unknowns, and b is the resultant vector. The unknown vector x could be found either with elementary algebra or by multiplying both sides by the inverse of A. Eigenvalues and eigenvectors are defined in a similar way:

$$Ax = \lambda x \tag{9}$$

where λ is a scalar eigenvalue and x is now the eigenvector. The equation shows that for every matrix A there is an equivalent scalar λ that can produce the same result for a corresponding v. In other words, eigenvectors v are the directions in which applying a matrix operation to v could have the same result produced by a scalar, and eigenvalues are the scalars that are multiplied onto the eigenvectors to achieve that same result. This is an extremely important idea because it means that there exists a way to simplify data and operations on that data by finding the eigenvalues and corresponding eigenvectors that allow for operations to be reduced to simply scaling the data, potentially insinuating the idea that there exists a coordinate axes upon which we could project our data such that all operations would be reduced to scalars.

To find the eigenvalues and eigenvectors of a matrix A, we can use a version of the quadratic formula known as the characteristic equation:

$$\lambda = \frac{tr(A) \pm \sqrt{tr(A)^2 - 4det(A)}}{2} \tag{10}$$

Where tr(A) refers to the trace of A, defined as the sum of the diagonal, and det(A) is the determinant of A, defined as the difference in the product of the diagonals. Then to find the eigenvectors, just use the eigenvalues and solve Equation 9 for x:

$$(A - \lambda I)x = 0 \tag{11}$$

Where I is the identity matrix, making the subtraction operation on the diagonal of A. This will provide a method of finding both the eigenvalues and the eigenvectors for any matrix A.

The next idea to examine is the bases and spans of vectors. These are useful concepts in linear algebra and help define the function of Eigenfaces. Given a set S of vectors $x_1, x_2, ..., x_n$, the set is considered linearly independent if

$$c_1x_1 + c_2x_2 + ...c_mx_m = 0 \tag{12}$$

is only true when $c_1, c_2, ...c_m$ are all 0. In general this definition is meant to find vectors such that the set S can encompass all points in a given space. The capability of reaching all points in a given space is known as the span of the set S. This is defined as all the linear combinations of the vectors in the set:

$$c_1x_1 + c_2x_2 + ... + c_mx_m \tag{13}$$

Where c is a constant that is multiplied onto the vector v. This culminates in the idea of bases. A basis of a vector space V is defined as a set of vectors S such that the vectors in S are linearly independent and all points in the vector space V can be expressed as linear combination of the vectors in S. The Eigenfaces is a basis for an image space defined by a set of eigenvectors where all the eigenvectors are linearly independent and all points in the image space can be expressed as a linear combination of the eigenvectors. Matching is done by looking at the images as a linear combination of the eigenvectors and comparing the corresponding coefficients.

Eigenfaces are based on the idea of principal component analysis, or PCA. PCA is a technique used to find the principal components of a set of data. In this manner we can find the data that contains the most information and view it or use it in that way. To find the principal components of a set of data, we turn to eigenvalues and eigenvectors. When finding the eigenvalues and eigenvectors of a covariance matrix for some data set, the eigenvectors represent a direction of the data in some high dimensional space, and its corresponding eigenvalue represents the amount of variation that occurs in that direction. Thus the eigenvector with the highest corresponding eigenvalue is the direction of the highest variation, the principal component that holds the most information. Figure 3 shows a graphical representation of an arbitrary set of data and the eigenvectors associated with the three highest eigenvalues. As can be seen, the red eigenvector associated with the highest eigenvalue is along the dimension of the data set with the highest variation once projected onto the eigenvector axis. The other two eigenvectors are the orthogonal axes that correspond to lower eigenvalues, thus representing direction that preserve less variation and thus less information.

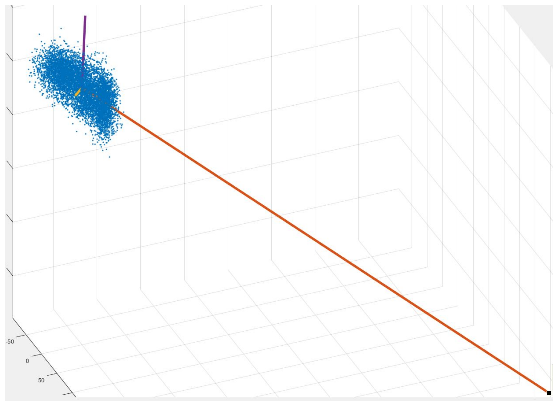


Fig. 3. A 3D graph showing an arbitrary cluster of data and the associated eigenvectors of that data with the highest eigenvalues. The red eigenvector has the highest eigenvalue and thus runs the length of the data set, preserving the most variation and allowing for a projection onto a coordinate system that is much easier to work with but also retains the most information.

The fact that principal eigenvector represents the direction of most variation in the data is very important because it means there exists a vector space upon which we can project our data into such that the most important information can be viewed more easily.

To find the principal eigenvector, we can use a matrix operation called Eigendecomposition which breaks a data matrix down into its eigen components. Given a matrix A, its Eigendecomposition is defined as:

$$A = Q\Delta Q^{-1} \tag{14}$$

Where Q is a matrix whose columns are the eigenvectors of the data matrix A, and Δ is a diagonal matrix that holds all the corresponding eigenvalues. This method makes it really easy to find the highest eigenvalue and the corresponding eigenvector. With this you can find the principal eigenvector. However, the catch with Eigendecomposition is that it only works with square matrices. To find the principal eigenvector of a rectangular matrix, a more generalized formula can be used. It is known as the Singular Value Decomposition, and it is defined as follows:

$$A = UEV^T \tag{15}$$

Where A is the original data matrix, U is the eigenvector matrix whose columns are eigenvectors of AA^T, E is the diagonal matrix that holds the eigenvalues for either eigenvector matrix (they're the same) along the diagonal, and V is the eigenvector matrix whose columns are the eigenvectors of A^TA. It is important to note that U and V have the same number of eigenvectors which correspond to the eigenvalues. The Singular Value Decomposition is more useful in that E contains all the eigenvalues in descending order, so you can easily tell which ones are the principal eigenvalues and eigenvectors.

The reason in finding the principal eigenvectors and the reason why Eigenfaces is such a powerful algorithm is because PCA can be used in a technique called Dimension Reduction, where a set of data can be projected onto a lower dimensional space while retaining as much information as possible. Once the Eigenfaces are found, one can select the principal eigenvectors in the Eigenface matrix to be used for actual image comparison. Since the eigenvectors whose eigenvalues are the highest contain the most information, that means that the ones with the lowest eigenvalues contain the least, and thus they can be discarded because most of the information is retained within the principal eigenvectors. Then after selecting the strongest eigenvectors, we can project the data onto that matrix to effectively decrease the dimensions of the data. Given a matrix A whose columns are the data samples, and a matrix B whose columns are the principal eigenvectors, dimension reduction is produced by the following:

$$D = B^TA \tag{16}$$

Where D is the reduced matrix that contains the matrix A projected onto a lower dimensional space B by using the dot product. Principal eigenvectors essentially find another coordinate axes to project the data onto such that the information is retained but transformed into a smaller, more manageable version. One caveat to this approach is that there will be loss of data, however minimal, so it's not necessarily the most optimal. The idea, however, is to lose the least important data such that the rest of the data can function and be operated on much faster. In this manner, we can significantly reduce the amount of space and the amount of time required to compute facial recognition through elimination of negligible dimensions in our data.

H. Eigenfaces Algorithm

The last algorithm is the Eigenfaces algorithm. This algorithm utilizes the ideas behind PCA and eigenvectors to formulate a set of vectors that encompass the vector space in which the images can be projected into. To find the eigenfaces for a data set A, we must first find the covariance matrix of the data set. This accomplished by centering the data set around the mean and then reshaping the images as column vectors:

$$A = \frac{1}{\sqrt{N-1}} \begin{bmatrix} x_1 - \mu_x & y_1 - \mu_y \\ x_2 - \mu_x & y_2 - \mu_y \\ \vdots & \vdots \\ x_N - \mu_x & y_N - \mu_y \end{bmatrix} \tag{17}$$

And then multiplying the new A by its transpose to find the covariance matrix with respect the to image to image correlation:

$$C = A^T A \tag{18}$$

Where C is the covariance matrix of A with respect to the image to image correlation. Next we will find the eigenvalues and eigenvectors of the covariance matrix, using EigenDecomposition:

$$C = Q \Delta Q^{-1} \tag{19}$$

The reason we choose not to use Singular Value Decomposition directly is because we are looking for the eigenvectors of the covariance matrix with respect to the pixel to pixel correlation, which will be the much more accurate version of the Eigenfaces. The problem is that computation to directly find that is too intensive for practicality, so we will first find the eigenvalues and eigenvectors of the easier image to image covariance matrix, and then use the principles of Singular Value Decomposition to find the Eigenfaces.

Once the eigenvectors of the covariance matrix have been found, we will apply an interesting rule of Singular Value Decomposition. Recall that in the formula of Singular Value Decomposition, it is important to note that the eigenvalues of U and V are the same, and they also have the same number of eigenvectors which correspond to the eigenvalues. Thus if you can find the eigenvectors for one matrix, you can determine the other one. We currently have the eigenvectors for the matrix $A^T A$, which is based on an image to image comparison. To calculate the pixel to pixel comparison, we must find the eigenvectors of AA^T. To do so, if we assume that v is the eigenvector matrix and λ is the eigenvalue matrix for $A^T A$, then:

$$(A^T A)v = \lambda v \tag{20}$$

$$A(A^T A)v = A\lambda v \tag{21}$$

$$(AA^T)Av = \lambda Av \tag{22}$$

This proves that the eigenvectors of AA^T given the eigenvectors v of $A^T A$ is Av. We can now apply this idea to our eigenvectors, by multiplying the matrix A which contains the centered images as vectors with the eigenvectors of the image to image correlated covariance matrix:

$$E = Av \tag{23}$$

Where E is the set of eigenvectors that contain the pixel to pixel comparison and can be used to find images as a linear combination of the eigenvectors. These are the Eigenfaces.

The power of Eigenfaces lies in the fact that most of them don't even have to be used in the facial recognition process. Figure 4 looks at the Eigenfaces of a set of faces, with the left being the first Eigenface with the highest eigenvalue and the right being the last Eigenface with the smallest eigenvalue. As can be seen, the first resembles a face with different intensities to represent different correlations and similarities points of various aspects of the images. The last image is almost synonymous to noise. This shows how the eigenfaces with the higher eigenvalues contain more data and thus can represent the image more clearly. The lower eigenfaces with smaller eigenvalues don't even need to be accounted for because the information they contain is negligible.

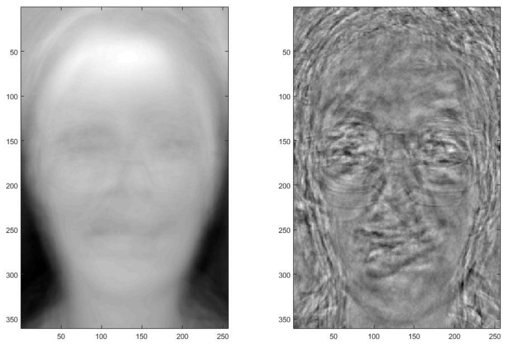


Fig. 4. A comparison of the strongest Eigenface (left), and the weakest Eigenface (right). Note how the strongest Eigenface looks like a face, whereas the weakest Eigenface resembles noise. This shows how in the Eigenface space, only the higher eigenfaces need to be preserved and utilized in image comparison because they contain the bulk of the information. We can decrease computational time tremendously by tossing aside eigenfaces that we don't need because it's less comparisons and operations that the program has to do.

This allows us to simply toss aside the eigenfaces that contain little information because they aren't needed in the image comparison. Computational time is decreased significantly because there is less data to compare and less operations to for the program to do. Thus eigenfaces should be considerably faster than other algorithms because they inherently use less space and less comparisons to accomplish the same objective.

The only concern with eigenfaces is the potentially slight dent to accuracy because the algorithm is throwing a bit of information away by compressing the data into a lower dimension. However, it can be assumed that data lost is negligible and shouldn't be a deciding factor when looking at the performance of image comparison with the algorithm. The trade off of this algorithm is a massive boost to speed with about equal or stronger accuracy, and a much smaller usage of space.

III. Performance

Having presented the algorithms under consideration, we now turn to experimentation and an examination of the performance of each algorithm under different conditions.

A. *Standards for Performance*

Performance is graded on three different categories: speed, accuracy, and space. Four experiments were conducted: one where a 301 image training set and a 43 image testing set was used, one where a 43 image training set and a 301 image testing set was used, one where a 301 image training set and a 32 image testing set under different lighting conditions was used, and one where a 43 image training set and a 32 image testing set under different lighting conditions was used. For the purposes of this experiment, space wasn't necessarily an issue as the data volume wasn't high enough, but it will still be accounted for by using an analytical comparison. Speed and Accuracy are the most important considerations in these experiments, and they weighted about equally.

B. *Experiment - Speed*

Testing was ran for the four experiments and data was collected for speed. Figure 5 shows a bar graph that contains the data on the four different experiments with respect to the time it took for each algorithm to complete its computation. The experiments are arranged from least to greatest amount of data. As can be seen, with increasing data, Euclidean Distance and Image Correlation (in yellow and blue respectively) increase constantly in time. However, Eigenfaces appears to be more logistical, increasingly quickly in the beginning and then reaching a cap, even decreasing in time.

Surprisingly, the data shows that Eigenfaces was not as fast as we imagined, and rather is slower than Euclidean Distance, and about as slow as Image Correlation, which is known to have long computation times. There is, however, a trend where the experiments that are dealing with larger data sets show a faster or equivalent time for Eigenfaces and longer times for the other algorithms. This could potentially speak to the idea that Eigenfaces are indeed fast, but for smaller data sets the simpler algorithms can be comparatively faster. Perhaps looking at experimentation with a much larger data set would be able to shed more light on this.

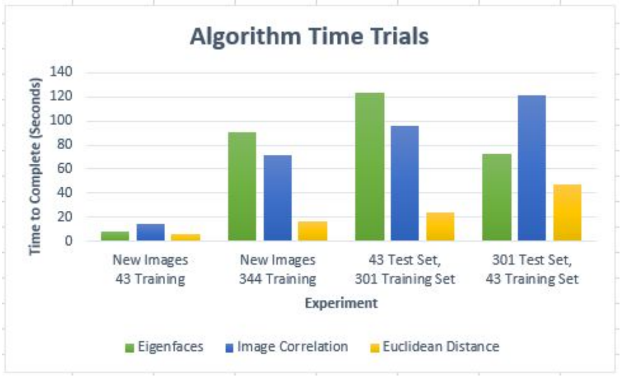


Fig. 5. A bar graph showing the time trials of the three different algorithms under different conditions. They are arranged from the smallest amount of data to the largest amount of data. Note that with increasing data sets, while Euclidean Distance and Image Correlation increase in time constantly, the Eigenfaces Algorithm actually decreased in time for the largest data set. This suggests that while Eigenfaces was slower for the smaller data sets, it is actually faster for larger data sets when dimension reduction becomes much more useful and apparent.

C. *Accounting for Space*

When looking at space, an analytical approach will suffice, especially since empirical data collection wasn't possible and the data sets weren't particularly large. When looking at space, we assume that an algorithm takes up memory through the data set that it holds and uses in its comparisons. We assume that this will primarily be in the form of matrices, and that each element in a matrix will take up 8 bytes of memory.

Examining the Euclidean Distance algorithm, it is clear that the it's usage of space is quite high. The space required for Euclidean Distance can be modeled by the equation:

$$E = 8XNM \tag{24}$$

Where E is the space taken by the algorithm, X is the number of images in the data set, and N and M are the dimensions of an image. The algorithm does a direct comparison of all pixels with all the pixels of the other image. This means that the algorithm will constantly be holding a data set of NxM images. Then given the number of images, that data set will increase linearly.

The Image Correlation algorithm uses an equally high amount of space for its program. The space required for Image Correlation can be modeled by the equations:

$$I = 8(4X^2 + XNM) \tag{25}$$

$$P = 8(X^2 * (NM)^2 + XNM) \tag{26}$$

Where I is the space taken by image to image correlation, P is the space taken by pixel to pixel correlation, and X, N, and M are the same as above. The algorithm can be relatively moderate using an image by image correlation, as it would create a 2 by 2 correlation for every single image with another image in the data set in addition to holding the entire data set itself, but even then this computation exhausts quickly the more images are used in the database and for testing. The pixel to pixel comparison, which is more accurate, is near impossible to scale with space, as it creates a NxM correlation matrix for every single image with another one in addition to holding the entire data set. The scalability of the this algorithm is probably the worst, as any increase in the size or the number of images will lead to a massive increase in the size of the correlation matrix and the amount of computation involved.

The Eigenfaces algorithm is by far the most efficient in usage of space. The space required for Eigenfaces can be modeled by the following equation:

$$V = 8(\frac{X}{A}NM) \tag{27}$$

Where V is the space taken by the Eigenface algorithm, A is a constant that is based on the number of eigenfaces the user chooses to use, and the other variables are the same as above. Using all 301 images to create the eigenface matrix only results in the usage of about 100-120 eigenfaces, as any more past that adds a negligible amount of information to the eigenfaces. This means that the last 200 eigenfaces can be unused and save 200 NxM images from the data set, possibly more. Dimension reduction allows for a division by some constant, determined by the number of eigenfaces one chooses to eliminate. This means that the program only needs to store a few pieces of data, namely the principal eigenfaces, to compute, thus making it much more efficient terms of space than the other algorithms.

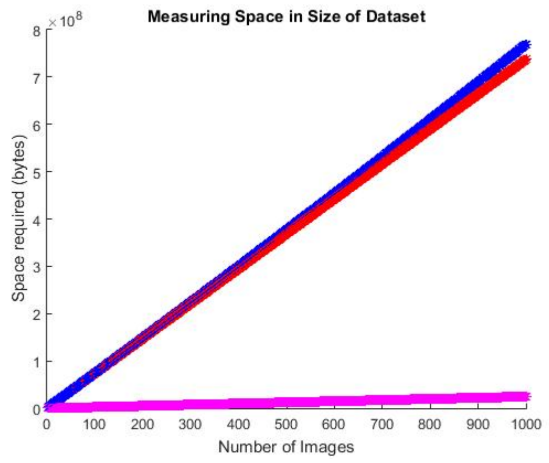


Fig. 6. A graph showing the number of bytes taken up by the three algorithms as a function of the number of images in a data set. The Eigenfaces algorithm is in magenta, the Euclidean Distance in red, and the Image Correlation in blue. Note that the Euclidean Distance and Image Correlation algorithms both use a large amount of data that climbs quickly as the number of images increases, while the Eigenfaces algorithm consistently uses a small amount of space that isn't nearly as memory intensive as the other two. This is attributed to the fact that Eigenfaces only uses a certain number of images in the data through PCA, so it's less affected by scalability. Pixel to Pixel correlation isn't even being shown due to the fact that it's off the charts.

Figures 6 and 7 simulate the three algorithms based on the their space equations above in terms of the number of images in a data set and the size of the images in a data set. The magenta is Eigenfaces, the red is Euclidean Distance, and the blue is Image Correlation. Figure 6 is a function of the number of images, simulated with 360 x 256 images from 1 to 1000. As can be seen, Eigenfaces consistently maintains a lower memory usage than the other two algorithms by a wide margin. This is because despite the number of images that come in, the Eigenfaces algorithms only needs the first X number to compute, so it doesn't scale as much as with the number of images. In comparison, Euclidean Distance and Image Correlation make full use of all images in the data set, thus taking up way more space.

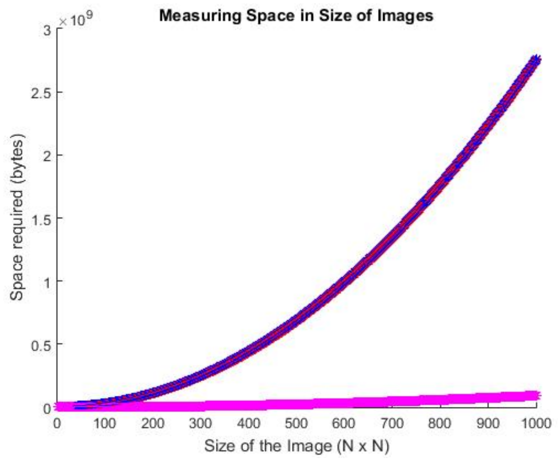


Fig. 7. A graph showing the number of bytes taken up by the three algorithms as a function of the size of the images in a data set. The Eigenfaces algorithm is in magenta, the Euclidean Distance in red, and the Image Correlation in blue. Note that the Eigenfaces algorithm is once again considerably lower than other two. This is attributed to the fact that Eigenfaces only uses a certain number of images in the data through dimension reduction, so it's less affected by scalability than the other two. Once again, Pixel to Pixel Correlation isn't shown because it goes off the charts.

Figure 7 shows the algorithm space equations as a function of the size of the images. The simulation was conducted with 344 images with NxN pixels, from 1 to 1000. The color coding is the same as above. Note that again, Eigenfaces is consistently using less memory than the other two algorithms. This is because Eigenfaces scales linearly very slowly with dimensions, whereas Euclidean Distance and Image Correlation scale linearly much faster. Dimension reduction once again shows that only targeting the principal eigenfaces, the Eigenfaces algorithm utilizes much less space.

D. *Experiment - Accuracy*

Testing was also ran for the four experiments and data was collected for accuracy. The accuracy of the algorithms is determined by a simple percentage calculation of correct recognitions over total number of images, and given that the images tested number in the 30-40 range in general, these measurements can be confident to roughly 2-3 percent. The accuracy for each algorithm on each experiment is shown in a bar graph in Figure 8, displaying Eigenfaces in green, Euclidean Distance in yellow, and Image Correlation in blue. The graph shows a side by side display of the different algorithms each experiments. Note that on the experiments where the images were taken in the same lighting, the algorithms all did much better in comparison to the data sets where images were taken in different lighting conditions.

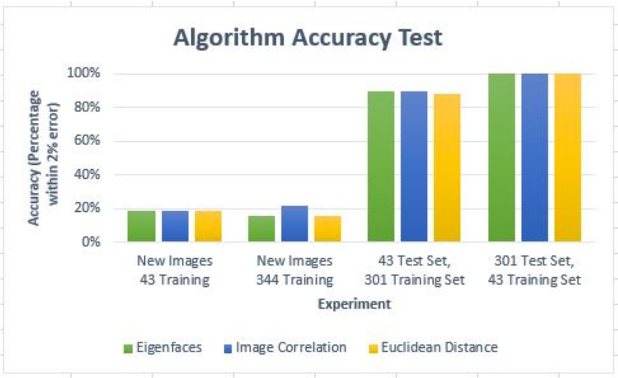


Fig. 8. A bar graph that shows the accuracy of the three algorithms in the four different experiments. It is important to note that the Eigenfaces (green) has about the same accuracy as the Euclidean Distance (yellow) and Image Correlation (blue), if not better. This is interesting because despite using dimension reduction and losing information, the Eigenfaces algorithm still performs at the same level as the other algorithms. In general, it appears that all algorithms performed great on images with the same lighting, but terribly on images with different lighting.

As can be seen, the Eigenfaces have equal to or better accuracy in comparison to the other algorithms. The only time in which they lose slightly in accuracy is in the second experiment with 344 training set and 32 test images taken under a completely different setting and lighting environment. This is significant because especially taking into account that fact that Eigenfaces actually throw away dimensions and lose data during the processes of the algorithm, this shows that they are actually still on par or even slightly better in term of correctly recognizing faces.

Holistically speaking, all three algorithms performed well for the images with the same lighting conditions, and performed poorly for the images with different lighting conditions. This is probably due to the fact that none of the algorithms take into account lighting and are comparing solely based on pixel values, which is skewed by lighting.

E. Results

Overall, it appears that all three algorithms have the same performance in terms of accuracy. However, in terms of space required, Eigenfaces win by a landslide, given the way the algorithm is created with PCA. In terms of speed, it appears the Euclidean Distance algorithm is the fastest, but looking at larger data sets there is evidence to suggest that the Eigenfaces algorithm will perform better as the data sets become larger and larger.

IV. Conclusion

We now conclude the paper with key takeaways.

A. The bottom line on the Algorithms

Given the evidence, it is apparent the three different algorithms perform differently in various ways. In terms of speed, Euclidean Distance computed the fastest, holding true to its simplistic design. Image Correlation was considerably slower, largely due to its additional computations of constantly generating new correlation matrices to do the comparisons. Eigenfaces was technically supposed to be much faster than the other two, but the evidence suggests that it's actually slower. However, upon closer examination it appears the evidence suggests that the Eigenfaces algorithm begins to speed up as the size of the data set grows, which leads to the conclusion that the Eigenfaces algorithm is faster for larger data sets, and it is just that our experimental data was based on data sets small enough such that the other algorithms could still keep up.

In terms of space, it is very clear that the Eigenfaces algorithm is more efficient with its computation. In both the size of the data set and the size of the images, it consistently uses less data than the other two algorithms. In addition, when looking at larger data sets, the Eigenfaces algorithm is the only one that can be maintained feasibly when the size of the data sets or images become extremely large. The Euclidean Distance and Image Correlation algorithms would both eventually become unusable because of memory overflow.

In terms of accuracy, all three were quite similar. Image Correlation was slightly more accurate, most likely due to the additional computation it does to create correlation matrices and thus minimizing deviation more in its results, but the edge is so small that it could be considered negligible. What's important to note is that Eigenfaces still managed to maintain its accuracy despite reducing its dimensions. Even with the loss of information, the Eigenfaces algorithm was still accurate. This proves the idea that Eigenfaces do hold an advantage in their computations, as they perform faster with much less space, while still maintaining the same level of accuracy, if not better.

In conclusion, it is evident that when comparing different algorithms for facial recognition, the Euclidean Distance Algorithm, the Image Correlation Algorithm, and the Eigenfaces Algorithm, all three algorithms seem to perform quite well for known data sets and perform poorly on unknown data sets, but the Eigenfaces Algorithm can do this more efficiently and quickly for larger data sets.

B. Shortcomings of the experiments

The largest shortcoming of the experiment was the usage of such a small data set and then proceeding to measure performance of algorithms which normally should be used in a much higher data volume setting. Especially given the variability of the data and small sample size, it come as no surprise should some values be skewed or contain large errors. Especially when the evidence we had suggested more insight could be garnered in a larger data set, our experiment design of having only about 300 images was a flaw.

Another fallacy in the experiment is the lack of accounting for environmental variables, such as lighting, brightness, contrast, etc. No pre-processing was done on the images, mainly due to a lack of resources to efficiently normalize all the images to the same range of intensity values. This resulted in potential problems with recognizing faces since the algorithms mainly compare based on the values of the pixels in some manner, and having different reference points for the intensity ranges of the pictures could have skewed results.

However, we still believe that these flaws don't significantly impact the results, and that our conclusions still grant insight on the performance levels of the various algorithms.

In the end, we believe we have offered substantive evidence to show that facial recognition is an intriguing subject, and that different approaches to facial recognition algorithms can manifest in vastly different outcomes, both in computation and in results.

V. Letter to the Editor

Thank you for the feedback on our technical paper you provided us with earlier this week. We greatly valued your brilliant insights on our paper and the areas of possible improvement. In response to this feedback, we have edited several portions of the paper. We hope that the changes, as noted below, successfully address the concerns you had.

First, we oriented the entire paper more towards mathematical proofs and less towards analytical explanation. We added several more equations to help aid in the explanation of various parts of the paper, most notably in the correlation section and the eigenfaces section. We also rewrote most of these areas such that they revolved around explaining the equations and explaining their significance so that the reader can better understand what they're looking at. There are still some parts that use analytical explanation, but we felt that these parts didn't have any substantial math behind them worth presenting and were better explained conceptually.

Next, we rewrote the introduction and the conclusion such that they were more in line with the evidence we had collected. Instead of trying to prove the superiority of Eigenfaces, we have structured it such that we are exploring three different approaches to facial recognition, and it appears that all are quite decent in terms of recognizing images, and Eigenfaces seem to have a competitive edge for larger data sets. We decided to keep the small discrepancy where we hypothesize that Eigenfaces will be better in general, but later we find that our evidence suggests that it's better mainly for larger data sets. We feel that these changes should better reflect the contents of the paper as well as the what we personally discovered in our experiments with these different approaches.

In addition, there were substantial changes to the Background Knowledge and Algorithms sections, where we combined both into one and streamlined the explanation of the knowledge needed to understand the algorithms and the algorithms themselves. The paper now progresses by first explaining the conceptual premise that an algorithm is based on, and then introducing and explaining the pros and cons of that algorithm, and then moving to the next. We believe that this provides better flow to the paper and removes redundancy. There is still a little bit of repetition, but we felt that keeping the background information and algorithms separate still provided better organization, so some ideas are briefly repeated to introduce the algorithms.

Furthermore, we also changed the Performance section, adding in new, more refined bar graphs for the speed and accuracy sections and introducing equations and figures to the space section. We felt that the performance is now much more solid and substantial, and it lends a better air of credibility to the paper.

We also completely rewrote the Eigenfaces section, correcting the technical errors from before and introducing a new progression in explaining eigenvalues, eigenvectors, etc. We believe that the new explanation is more robust and easier to understand, so readers who have no idea what eigenvalues are or what PCA is should be able to comprehend what Eigenfaces are.

Finally, there were a number of small changes to the paper, such as updating Figure 1 and 2, rewriting some of the captions, revising some areas of the explanation for smoother flow and easier comprehension, and fixing errors in the document regarding the math and ShareLatex. Thank you once again for your feedback, and see you next semester!!

Acknowledgment

The author would like to thank Mark Somerville, John Geddes, Siddhartan Govindasamy and Rebecca Christianson for their extensive guidance in the progression of this project and the learning of building blocks that happened prior to this experiment. We would also like to thank the NINJAs Victoria Preston and Jong Nam for their help and support in troubleshooting the design and fabrication. Finally a big thanks to the entire Quantitative Engineering Analysis class for its assistance at various stages of the project.