Constructing an Orthonormal Basis for Image Decompression and Recognition

Adam Hoerger
Theo Lincke
Kyle Zhou

April 2019

Abstract

An image represented, in its initial state, contains a large amount of redundant information. In a time where millions of images and objects are stored in memory and even transported across networks at any given point in time, a need for an efficient compression method is crucial. This paper focuses on the deconstruction of an image matrix into its singular value decomposition, often abbreviated as SVD. By decomposing an image matrix, we are able to neglect a large portion of the image's underlying structure by removing a heuristically chosen amount of the singular values, reducing the amount of information stored by a significant proportion while allowing for a fairly accurate reconstruction of the image. We were able to compress an image to 15% of its original size, while still maintaining the image almost equivalent to the original.

A Singular Valued Decomposition, however, is not only limited to image compression. We extend the SVD to facial detection software that is able to fit an input face into an orthogonal "eigenface" basis. This can be extended to any pattern recognition because the method examines the covariance in a data set. Thus, other applications include the recognition of individuals' signatures or written letters, or even that of trends in a dataset. We analyze facial composition based on the covariance between a set of data faces, building a program that successfully found 4 out of 10 faces in a large image. We split this paper into two primary sections:

- Singular value image / data matrix decomposition techniques for image compression
- How SVD decomposition can be used to recognize images (primarily faces) using the Fourier coefficients as an input to the orthogonal basis, as well as reconstruct objects based only upon the basis created in the decomposition.

1 Introduction

1.1 The SVD and Image Compression

The singular valued decomposition is a very powerful form of matrix representation as it allows any linear transformation (even a singular or non square one) to be represented by a diagonal matrix with unitary transformation matrices. We will briefly describe the process we took in order to obtain the singular valued decomposition of an image matrix.

The final SVD of a (not necessarily square or nonsingular) matrix $A \in \mathbf{R}^{mxn}$ is

$$A = U\Sigma V^* \tag{1}$$

Where

1. U_{mxm} is a Unitary matrix composed of the left singular vectors of A, which are also the eigenvectors of AA^*. We frequently refer to the row space of U as the "Eigen Basis" or "Eigen Face Basis".
2. Σ_{nxn} is a Diagonal matrix with the singular values of A along the diagonal.
3. V_{mxn} is a Unitary matrix composed of the right singular vectors of A, which are also the eigenvectors of A^*A.

For the sake of generality, we include the complex conjugate transpose, however, for simple image manipulation, this is not the case. However, it should be noted that for practical application, a complex conjugate transpose may be necessary (electrical circuits for example).

The decomposition of the matrix into a diagonal matrix Σ, even when the matrix is singular or rectangular, is a very powerful tool, as it allows us to extract the magnitude of the matrix A, or more formally, the induced 2-norm,

$$\|X\|_2 = \underset{\|x\|_2=1}{max} \|Xx\|_2 = \sigma_1 = \sqrt{\lambda_{max}} \tag{2}$$

Where λ_{max} is the largest eigenvalue to X^*X

The magnitude has many interpretations, including the length of the major semi axis of an ellipse in a coordinate system (from the principal axis theorem). It also measures the squared standard deviation of a dataset in a distribution.

To obtain these three matrices, we consider the eigenvector / eigenvalue pairs for the matrix A^*A and AA^*. Note that the eigenvalues for these two matrices are the same, they are the squared singular values to the matrix A. The eigenvectors of these matrices are the singular vectors.

The rank of the matrix A is also the number of non-zero diagonal elements in Σ. Thus, if $rank(A) = r$, then we consider the compact SVD

$$A = U_+ \Sigma_+ V_+^* \tag{3}$$

such that Σ_+ is a square $r \times r$ diagonal matrix only containing the nonzero singular values.

It can easily be shown (Appendix B) that the zero singular values in Σ have no significance to the matrix A. Thus, in taking the singular valued decomposition of an image, neglecting the zero elements will have no effect. Now realize that, if k is the number of nonzero singular values, decomposing an $m \times n$ matrix produces a matrix of size $m \times k$ (U), another of size $n \times k$ (V), and a final one of size $k \times k$ (Σ). Though the last one can be stored as an array of length k, due to its diagonal nature, the SVD decomposition still results in a massive amount of memory storage.

We can, however, take advantage of the fact that the largest magnitude singular values are associated with the most significant singular vectors of A. See that, for an SVD decomposition, the singular values can be organized by principality, decreasing in size along the diagonal:

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 ..... \geq \sigma_r$$

This means we can keep k elements along the diagonal of Σ as well as k left and right singular vectors because, again from Appendix B, the singular value magnitude shows the effect of removing a singular value / singular vector triplet. Ignoring sufficiently many terms results in a representation of the original image that requires less space to store, while still maintaining the most critical pixel relationships that make the image recognizable. It should be evident ignoring more terms will provide for a more compact image at the expense of accuracy, as more significant singular values begin to be excluded.

We express error in two ways in this investigation. The first is the euclidian distance between the origional matrix A and the truncated matrix $\tilde{A}$ using the frobenious norm:

$$error = \left\| A - \tilde{A} \right\|_F$$

We also look at the actual and practical error, or the image compression ratio. For an image that is stored as n bytes in memory and $\tilde{n}$ bytes after decomposition, the compression ratio is

$$error = \frac{\tilde{4n}}{n}$$

The first gives a mathematical representation of the accuracy of the transformation. It is reasonably stated that at higher singular values, this will be smaller. while the second, due to memory error, shows the actual memory loss of the compression. We multiply $\tilde{n}$ by 4 due to memory organization of floats vs integers, described later.

1.2 SVD and Covariance

The previous definition of the SVD does enough to show the decomposition technique for singular valued factors, however, we approach the SVD in a different point of view in the second part of this paper. Imagine now, that we have a collection of vectors x_1 through $x_n \in \mathbf{R}^{mx1}$. We subtract the average vector from all of these vectors to center the data set about the origin.

$$a_{mx1} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$$x_i = x_i - a, i = 1, 2...n$$

Let the vector u denote the direction of distribution of data in the collection of vectors $x_1 - x_n$ and μ denote the magnitude (variance) in the direction of u. Written formally,

$$\mu = \underset{\|u\|_2=1}{max} \frac{1}{n} \sum_{i=1}^{n} (u^* x_i)^2 \tag{4}$$

By definition, the covariance matrix C is

$$C = \frac{1}{n}\sum_{i=1}^{n}(x_i x_i^*) \tag{5}$$

It can easily be proven (Appendix A) that μ is the eigenvalue of the covariance matrix C and u is the corresponding eigenvector and that each $\mu_i < \mu$ is the next most prominent variance in the x data set. So, each eigenvector u_i measures the direction orthogonal to u_{i-1} (orthogonal to all u for that matter) and with $\mu_i < \mu_{i-1}$. Further more, each μ is the standard deviation for that direction in the data.

Theorem 1.1 (Covariance of a Set of Random Vectors)**.** *The direction of maximum variance in a set of m normalized random variables of multiple dimensions (random vectors) is equal to the eigenvector u_1 of the covariance matrix $C = XX^*$ where $X = \frac{1}{n}[x_1|x_2|...|x_m]$ and the magnitude is equal to the eigenvalue σ_1 that corresponds to u_1*

So for any randomized data set, such as a set of vectorized face images or handwritten digit images, the data tends to point towards the principal eigenvector of the covariance matrix.

We use this fact to demonstrate that a set of images can be vectorized and turned into a large matrix that one can then extract all of the necessary information from using the SVD decomposition as a 'eigen basis' (we use the row space of U in the above decomposition) for the object that the images represent. One could then apply an external image to this basis and find the orthogonal distance from the given basis. This would give a practical object detection software algorithm for finding specified objects and a single unitary matrix would be all that is stored in memory.

From the Closest point theorem, we can then find the magnitude of the error of a projection onto this matrix by using the norm of the orthogonal projection onto our 'eigenbasis'.

Theorem 1.2 (Closest Point Theorem)**.** *The unique vector in a subspace M of the inner-product space V that is closest to a vector $b \in V$ is $p = P_M b$, the orthogonal projection of b onto M*

2 Lossy and Lossless Image Decomposition

An image can be represented as three *nxm* matrices for each of the three image colors red, green and blue. For this investigation, we chose to stick to a gray-scale image, however, this can be translated directly to a colored image by repeating the process on each of the three color matrices, see [5] for a use of the colored image decomposition.

Thus, our end goal is to compress a black and white image into its SVD factors in order to condense the amount of information in the image.

A naive approach would be to compress the image by factoring the image matrix into its two orthogonal matrices and one diagonal matrix using the singular value decomposition. However, we established in section 1.1 that using a compact SVD would represent the original matrix exactly, while requiring less memory (as it excludes unnecessary columns in U and V, as well as the zero terms in Σ). A compact SVD compression is thus executed on the image, factoring the image into a diagonal matrix with all of it's zero values removed (the bottom half of the diagonal matrix) and then removing the eigenvectors in U and V that correspond to $N(A - 0I)$, or the eigenvectors associated with the eigenvalue 0. This technique is completely lossless, as it stores all the necessary image, representing the original matrix exactly (assuming insignificant floating point errors).

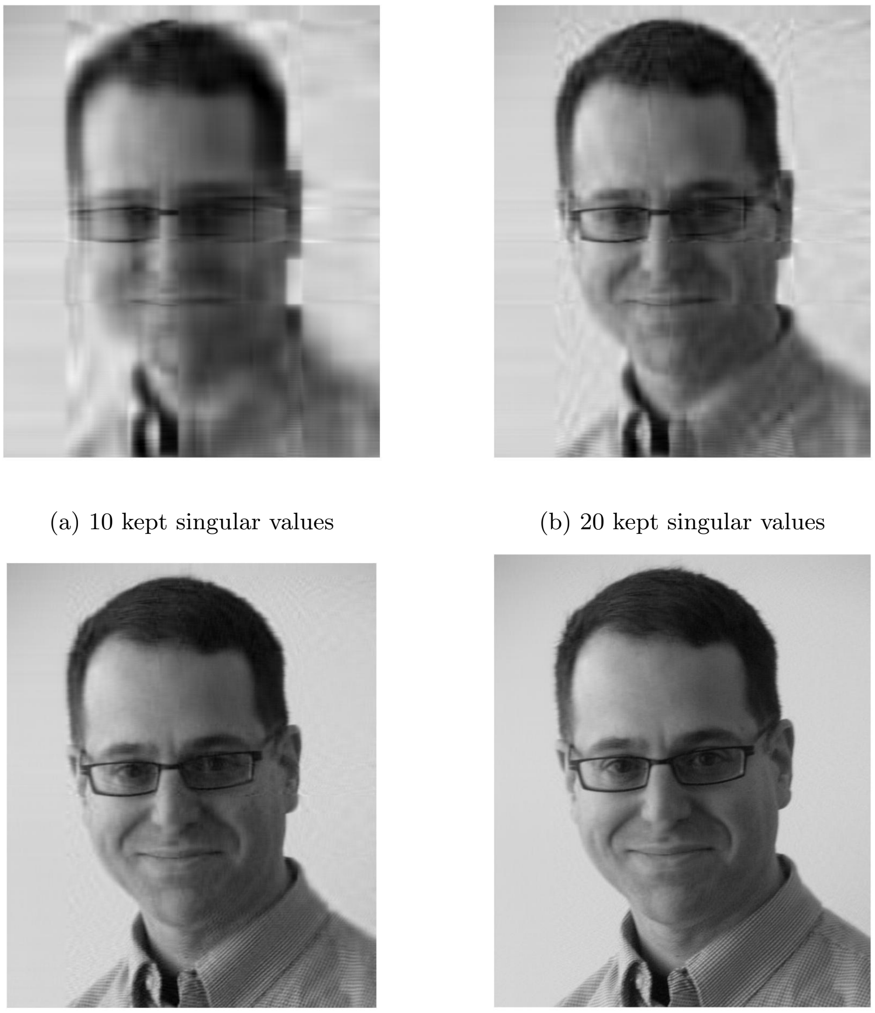
In this investigation, we compressed an image matrix, composed of 8 bit unsigned integers in memory, into three separate matlab arrays of single precision floating points (32 bits each in memory). Thus, for every value we stored in our SVD decomposition, we were storing 4 times as much memory as one value in the original due to this type conversion. This change in memory did not affect the final outcome as drastically as one may suppose.

Figure 1 shows an image compression with 10, 20, 50, and 100 kept singular values.

A few visual observations of the image show that that the higher compression image (10 kept singular values) has a somewhat grid like grain feel to it. This is because, as mentioned before, the SVD is eliminating unnecessarily small variances between bordering pixels. So a neighboring pixel that has small covariance with its neighbors is expressed as a new value expressed by the decomposed matrix. Thus, all the loss in information is due to the neighboring pixels, resulting in a “Grid like” grain feel.

For a quick reference, the image compression ratio (using actual memory storage rather than element storage) respectively for 10, 20, 50 and 100 singular values was:

- 10 kept terms: 3.5%


(a) 10 kept singular values

(b) 20 kept singular values

(c) 50 kept singular values

(d) 100 kept singular values

Figure 1: Decompressed images using 10, 20, 50 and 100 original singular value / vector triplets

- 20 kept terms: 7.00%
- 50 kept terms: 17.50%
- 100 kept terms: 35.00%

Note that in the above ratios, our images were stored as data matrices in binary files in Matlab. Some memory tricks could be used, such as storing each value of the matrix as an integer (only storing 8 bits of a floating point number) that we then convert to a floating point number, however, even with the change in data type, the compression of the image is evident with a small loss in image content.

When actually choosing the amount of data to keep in the image, it is helpful to see the magnitude of the effect of each singular value. Figure 3 shows the magnitude of each singular value along the diagonal of Σ. Note that we stop at 100 for brevity.

We also show the Frobenious norm ratio between the original and the compressed image over the origional image to normalize it in figure 2 to show the loss of reward for including more singular values. From a visual standpoint, the image appears to maintain its general structure between 90 and 100 singular values. The actual value we compute is

$$\epsilon = \frac{\|Original - Compressed\|_F}{\|Original\|_F}$$

The general interpretation for the singular value image compression method is that it removes the unnecessary noise produced from external stimuli such as random error in the camera, or random noise

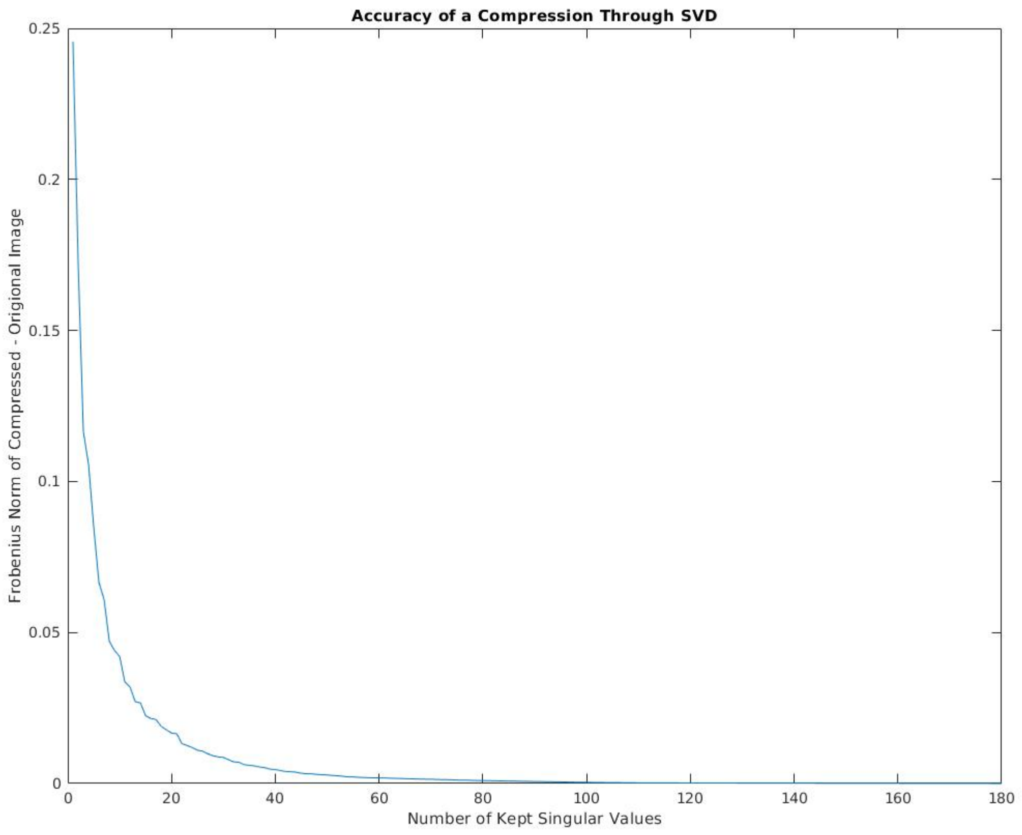


Figure 2: A plot of the Frobenius norm error vs the number of kept singular values

in the surrounding image which is interpreted as the covariance between neighboring pixels. For any image matrix, it is helpful to visualize the data in its most prominent sub components, quantified by the singular values. Of course, this same process can be carried out on many more types of vectorizable data sets.

2.1 Image Memory Data Structure

We touch briefly on the use of memory manipulation in order to store massive matrices. Image matrices can be stored easily and efficiently using a sparse matrix, or a meta data matrix. Such matrices would store only the indices and values for each non zero elements in the images. Further, the indices themselves could be removed by storing the matrix in a compact array in one block of memory. This not only collapses the amount of memory for each image, but it also offers an easy to extract string of values. origional image to normalize it in figure 2 to show the loss of reward for including more singular values. For the number storage system, an integer conversion is proposed. Depending on the magnitude of the singular decomposition, we can shift the bits of the stored floating point number so that it is represented as an 8 bit integer. To do this efficiently, we can multiply each eigenvector by a rough magnitude in order to store the necessary 8 bits, then through data extraction, we divide by the same number and convert the 8 bits to a floating point number. This would reduce the above percentages to a fourth, however, it would result in a small loss in memory from the rounded integer conversion.

2.2 Further Optimization of Image Compression

The SVD technique itself is very efficient at storing information in compressed form. With a combination of simplified data structures to store the data in memory could drastically reduce the amount of memory usage in an image file format. For cases in which images can be reconstructed, or when images don't rely on complete accuracy, an SVD keeping only a small amount of singular values, storing the image as 3.5% of actual memory is very efficient.

This method would involve drastically compressing an image (only keeping 1 - 20 singular values), then using an image reconstruction technique such as Zhu, Liu, Cauley, and Rosen's proposed Image reconstruction by domain-transform manifold learning [2]

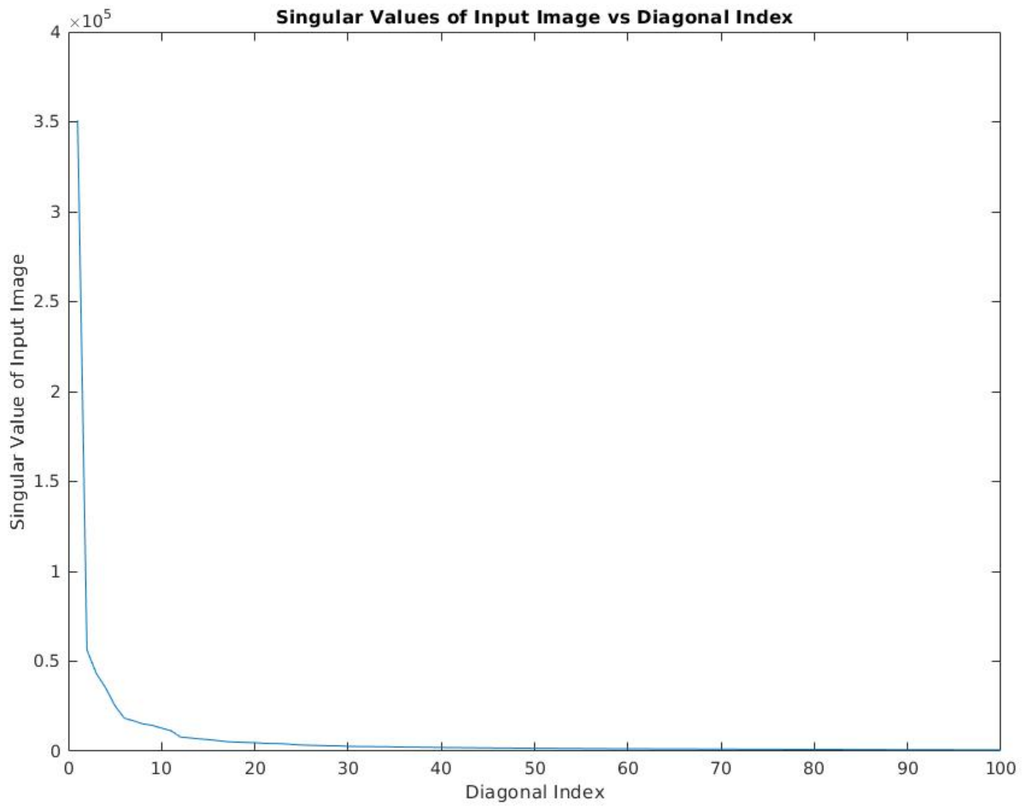


Figure 3: The singular values of an image at the diagonal index, showing a general magnitude of each singular value

2.3 SVD Compression drawbacks

In the end, the SVD technique is a potentially lossless technique, though lossy in practice. A dataset that must obtain all of its original information with no loss in accuracy will not stand well for the singular valued decomposition. A database of words, for instance, is subject to exactness in the end and would not benefit from a lossy compression technique. However, pairing this lossy technique with a reconstructing algorithm, one could possibly deconstruct a set of exact data and reconstruct it using reconstruction algorithms already known.

3 SVD and Covariance

Given a set of eigenvectors to the covariance matrix of a vectorized set of data, rebuilding or matching an input image to an orthogonalized basis could offer extremely accurate object detection, useful for facial recognition, signature validation and any software that requires object detection.

The problem for how to recognize objects based on their image data has hindered many programs from recognizing faces, objects and numbers. A useful program to have, image recognition can be achieved by using the covariance matrix found in section 1.

3.1 The Eigen Face

Introduction

A classic example of image detection is that of the human face. What does it take to detect whether a given image is distinctly a human face or contains a human face within? In order to answer this question, we go back the the covariance matrix found in section 1.

For our set of faces, we used the AT&T and Bell Labs normalized face database [1]. This database contained 2000 normalized faces, 10 for each individual and 200 unique individuals. If we vectorize each image so that each column in the image matrix is stacked to form a long vector (if an image is $p \times q$ we stack each column so that we obtain a $pq \times 1$ vector), we can then compose one massive matrix with the rows being the individual face vectors. From section 1, we could come up with a set of singular vectors and singular values that tell us the direction and variance of this distribution of faces. In an ideal data set, we would have relatively similar faces. So each face is centered at eye level and facing perfectly forward. It is also worth noting that for SVD to be practical, massive data sets aren't entirely necessary. We are decomposing our matrix into the general features for the average face, so a set of 30 - 40 accurately normalized face vectors could suffice. We chose to utilize a large amount of face vectors in order to visually see each eigen face and show the trend as we display less important singular vectors.

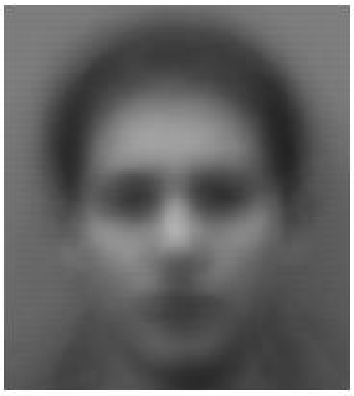
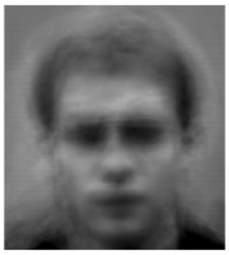


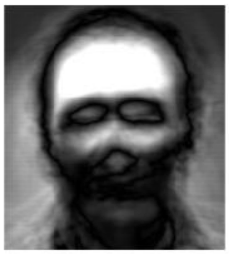
Figure 4: The Average Face

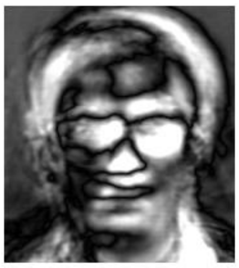
The end goal for a supposed image detection software is to form an orthogonal matrix U that contains a set of eigenvectors for the covariance of our dataset (or the singular vectors for our dataset) in decreasing principality. This way, we can extract individual vectors from the orthogonal matrix and project our image onto our "face basis".

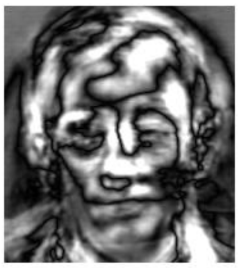
Process: Looking first at a single vector $\mathbf{x}$, we can use the matrix $\mathbf{xx^T}$ to find the covariance between each element in $\mathbf{x}$, given that the mean of all elements is zero. Specifically, the entry $[\mathbf{x}]_{\mathbf{ij}}$ would represent the covariance between x_i and x_j (also note that the matrix is symmetric).

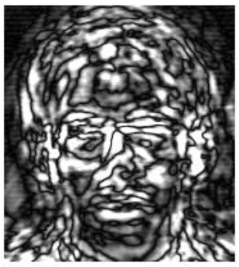
Now, if we take a set of normalized p x q face images (same size, angle, lighting, background, position, etc.) and reshape each image to a column vector m of size pq, we can construct a representation of the entire set as a large matrix A. Now define $\mathbf{a} = \mathbf{\frac{1}{n}\sum_{i=1}^{n}[A]_{\star i}}$. This produces a column vector, each of whose elements is the average of the corresponding row elements in A. As such, this represents the "average" face. Now, consider a set of vectors $\{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$ where $\mathbf{x_i} = [\mathbf{A}]_{\star \mathbf{i}} - \mathbf{a}$. We subtract off the average face, as this allows us to better focus on the differences between faces, which helps in creating a better basis. Additionally, this causes the mean of each vector become zero, which is helpful when creating the covariance vector. Now, defining $X = \frac{1}{\sqrt{(n)}}[\mathbf{x_1}\,\mathbf{x_2}\,...\,\mathbf{x_n}]$ the matrix XX^T would then represent the covariance matrix of the rows in X, which would then represent the relationship between each individual pixel in the average-subtracted image. Note that performing an SVD decomposition on X can be used to find the singular values and corresponding left-hand singular vectors of X. Because these also represent the eigenvalues and eigenvectors of XX^T, they can be used to form a basis for the covariance matrix of the pixels, which in turn gives us a set of "eigenfaces".


(a) 1^{st} Eigenface


(b) $2nd$ Eigenface


(c) 10^{th} Eigenface


(d) 20^{th} Eigenface


(e) 100^{th} Eigenface

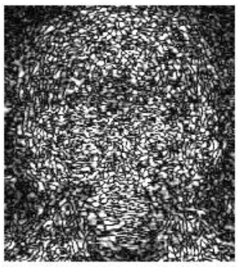
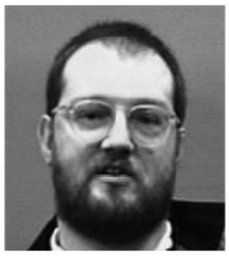

(f) 1000^{th} Eigenface

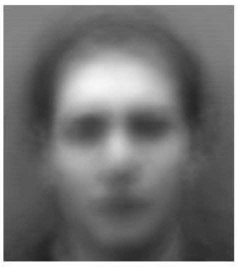
Figure 5: Various eigen faces contained in the orthonormal basis U

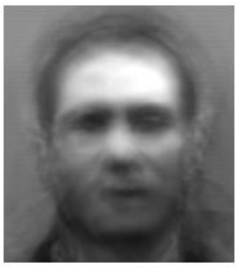
Using this technique on a set of faces produces the three desired matrices U, Σ and V. Figure 5 shows the individual columns of U, our orthogonal matrix whose column space is the basis for the set of faces. We show the 1st, 2nd, 10th, 20th, 100th and 1000th column vector (that we refer to as a single eigenface). These eigen faces are not meant to be show as an image, as they may contain negative or extremely low values. The basis U is used as a transition to transform an input face. Thus, each of these images is scaled by a scalar that has no significance. It is, however, interesting to look at these images.

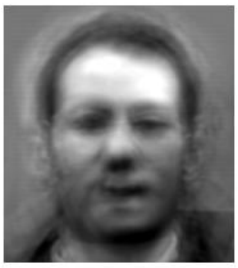
A few key observations show that the images increase in detail as you approach higher singular values. The first image for instance, is the most blurry and non-detailed eigenface, while the 1000th face is quite intricate.

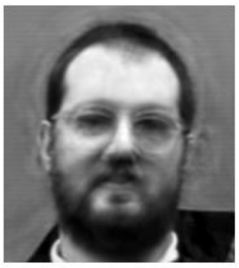
Each face captures general characteristics of the human face as a whole, and with a more normalized and accurate and well thought-out data set, a more accurate eigen face basis could be constructed.


(a) Original Image


(b) 1 Eigenface


(c) 10 Eigenfaces


(d) 50 Eigenfaces


(e) 100 Eigenfaces

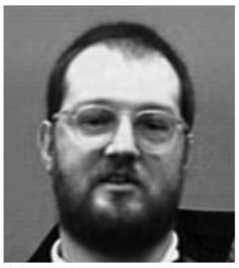

(f) 500 Eigenfaces

Figure 6: Various eigen faces reconstructed by the orthonormal basis U

3.2 Facial Recognition / Detection

After finding U Σ and V, we can project any relatively normalized face onto the orthogonal space spanned by $R(U)$ in order to capture the input image in the basis for eigenfaces. Let y be a reshaped vector from an input image or dimension pxq such that $y \in \mathbf{R}^{pqx1}$. We first subtract a to obtain our matrix that we want to project onto U. Thus, we attempt to solve

$$Ux = y - a \tag{6}$$

This is clearly a least squares problem, since **y - a**, or the face after removing the average, is not guaranteed to be in the range of U (in fact, after removing singular vectors, it shouldn't be). But since U is composed of orthogonal columns, we can now say

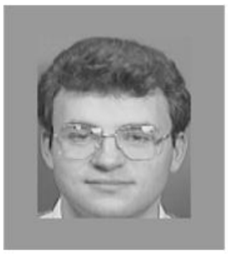
$$x = U^T(y - a) \tag{7}$$

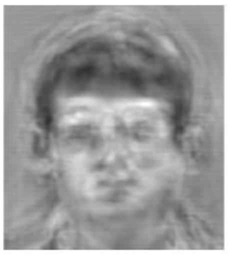
Our reconstructed face, $\widetilde{y}$ is

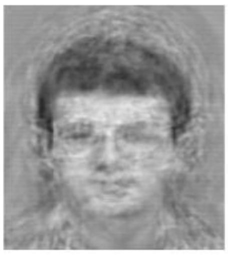
$$\widetilde{y} = U(U^T(y-a)) + a \tag{8}$$

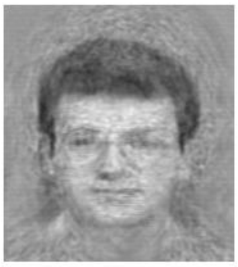
In summary, we just found a set of principal directions in which the overwhelming majority of our images are distributed (the eigenvector of the covariance matrix), gathered them up into matrix U as orthogonal singular vectors to our input matrix X and projected an input matrix using U and found the smallest distance between our original vector and the span of our face vectors in U using the closest point theorem.

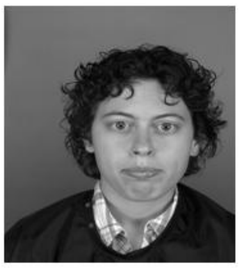
Figure 6 shows a reconstruction of a face that was previously in the dataset. After 500 eigen faces, it is almost indistinguishable from the original image. Figure 7 shows a reconstruction of a somewhat normalized face not found in the dataset as well as a reconstruction of a face that is off center and not positioned correctly to be reconstructed properly.


(a) Origional Image


(b) 100 Eigenfaces


(c) 500 Eigenfaces


(d) 1000 Eigenfaces


(e) Origional Offset Image

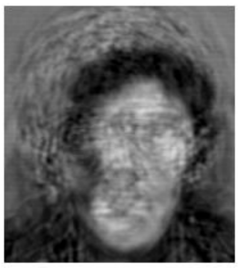
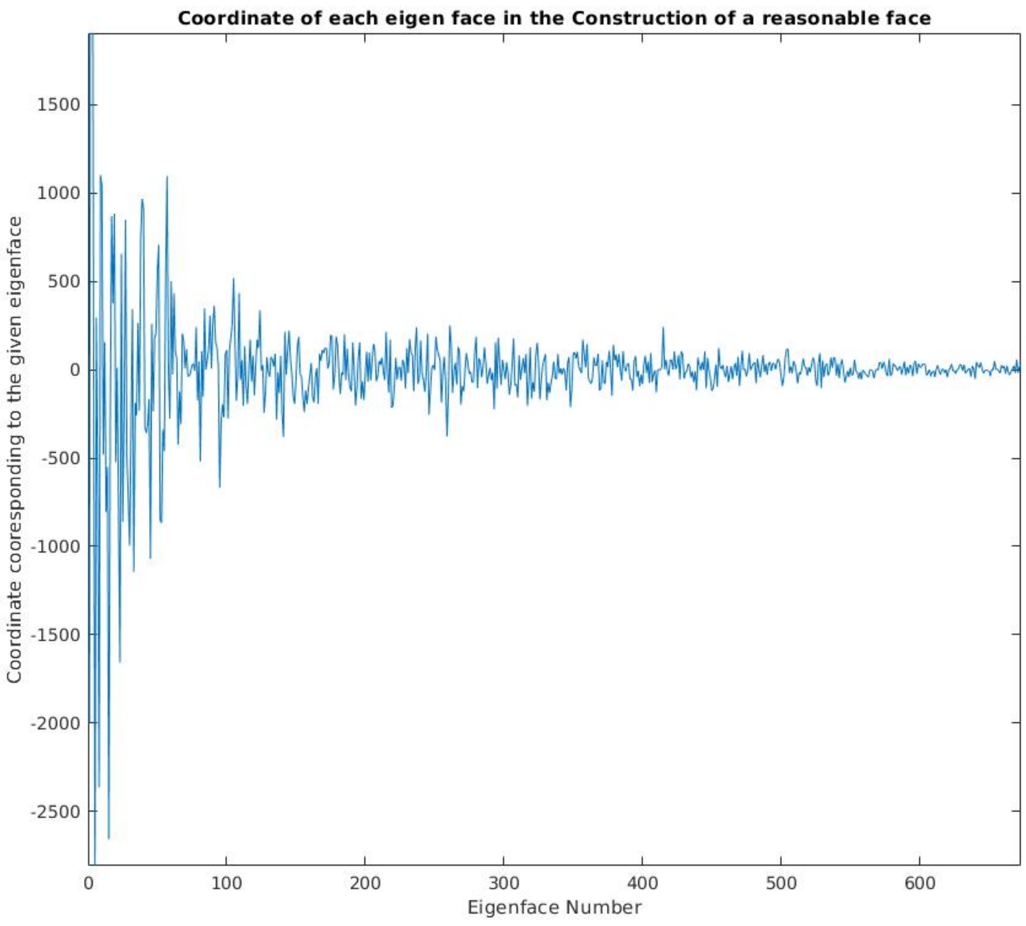

(f) 200 Eigenfaces Applied to Offset Image

Figure 7: An Eigen Face reconstruction using an image not in the dataset that is off center

So for a facial detection mechanism, we would construct a single eigen face matrix U. For a technical standpoint, we would store this matrix in the machine recognizing faces, so a camera running a live stream face detection algorithm would contain this matrix in memory and constantly sweep the image with this matrix in order to recognize when and where faces are in its field of view. For every object it detects, it would need a scalar error magnitude that it would compare the transformed image to the original.



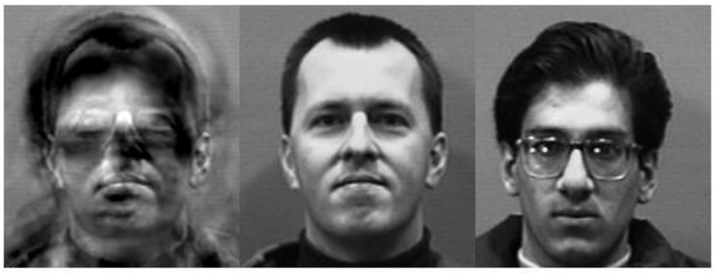


Figure 8: Attempting to morph the middle image into the right hand image by alternating the weight of each eigenface. The plot shows the coordinates of the right most face in the Eigen Basis.

Another sample application for the singular valued decomposition is actual facial construction. We demonstrate this crudely in Figure 8. Instead of starting from scratch, we start with the coordinates of a face already in the eigenbasis and move the coordinates around to try and recreate the second face using only the pre-existing eigen faces as a basis for our face space. This method for facial construction must be done somewhat heuristically. In order to form the constructed face, we examined each eigen face (basis vector in our Eigen Basis) and noted what characteristic the vector correspond with the most. For example, the 10th eigenface from Figure 5 shows distinctly the relationship between glasses and face and the 2nd eigenface shows the relationship between for head and under eye hue and the rest of the face. By tweaking each of the weights contained in the coordinate vector $\tilde{y}$ in $\tilde{y} = U^T(y-a)$, we were slowly able to reconstruct a face starting at an initial face. Of course, this is the exact same process as just projecting the desired face onto the eigen basis, however, the point in the end is that we could potentially create a face from scratch. Had we just projected the right face onto the eigen basis, we would have the most accurate representation of the face.

More efficient techniques could be used to find the perfect combination of coordinates to make an image more "human like". More notably, an accurate software system would examine the nonlinear trend in the coordinate plot (shown in Figure 8) and it would keep all eigen faces aligned on this curve.

3.3 Facial Detection Model and Error

To complete our model for a theoretical facial detection software, we consider the error of the projection of our matrix. When we projected our input image using the unitary matrix U, we found the minimized distance (due to the closest point theorem) from our projected space and our projecting space. If we were to put this minimized distance in terms of a single scalar, we would be able to get a percent error of our system in telling whether a given input is indeed a face.

In order to find the error of the system, we consider the euclidean distance the projected image lies from the average face. This error margin is practical in image scanning and facial detection. So for an input image matrix y, we compute $\tilde{y} = U_+(U_+^T(y-a)) + a$ and compare the euclidean distance, or:

$$error = \|y - \tilde{y}\|^s \tag{9}$$

To put this idea to the test, consider the image in figure 9.

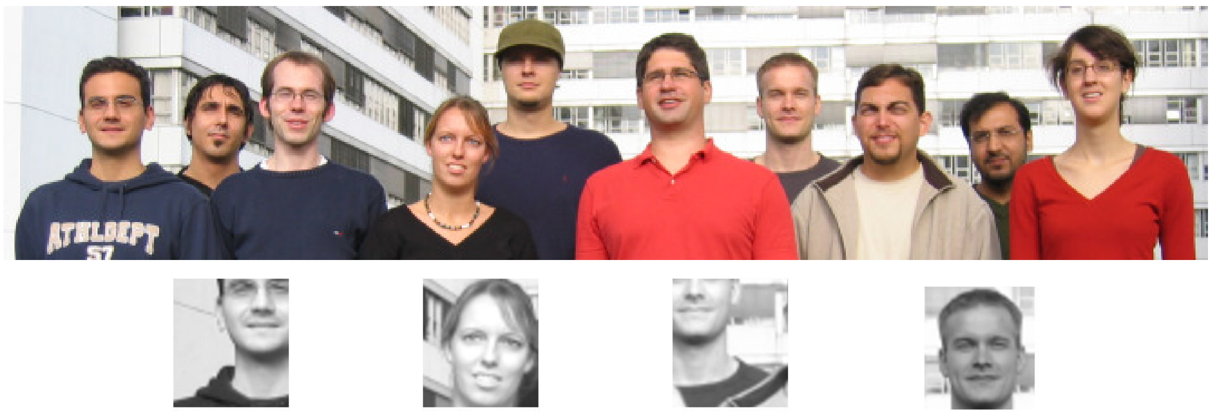


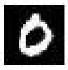
Figure 9: An image containing a set of faces

We were successfully able to detect four out of the ten faces in the image. Although our method for detecting the faces was not optimized (we took a kernel and applied it to every point on the matrix and found when the error was less than a certain bound determined heuristically), it shows that the SVD technique could be used for use with face detection in a large set of faces.

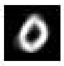
Figure 9 demonstrates the ability for an svd scanning image detection system to detect a set of faces from a larger image. The cropped faces at the bottom are the faces our svd image detection system was able to recognize. Although not fully optimized, the system was able to distinguish quite clearly a few prominent faces.

3.4 Further Application of the Covariance Method

The covariance matrix measures any covariance in a dataset. This method is not only limited to facial detection. We ran similar tests on digits, attempting to find handwritten digit eigen basis (figure 10). The svd transformation is a very practical object detection system.


(a) Original testing 0

(b) Original testing 8


(c) Digit 0 transformed by the 0 Basis

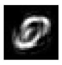

(d) Digit 8 transformed by the 0 Basis

Figure 10: Digits undergoing svd recognition

However, outside the realm of images lies data analysis in more complex systems. A fascinating controls application could be to detect the final outcome of a set of randomized variables in an automated systems. For example, in a theoretical network system for self driving cars, we could input the coordinates of each car in a traffic intersection as well as their relative speed and acceleration, we could decompose this data matrix and examine the optimal combination of position, speed and acceleration to allow for high speed intersection traversal. The opportunities for the SVD extend to any system that we are not able to analytically calculate the necessary covariance between surrounding values.

The drawbacks for such a system, however, result from the amount of leeway we had for each setting in the system. For example, for an accurate face detection software, we would need to distinguish faces (apply a training aspect to the dataset) so that the software would know which error is in fact, a minimum to be classified as a face. Similarly, when trying to reconstruct the face, we could use some optimization techniques such as extrapolation based on the nonlinear trend of the data, but this is still just an estimate. In the end, to practically approach the SVD, a certain amount of outside influence is necessary.

4 Conclusion

We examined the benefits of decomposing a matrix of variant random variables into its singular values and singular vectors. In doing so, we were able to measure the variance of a data set and the direction of distribution. This was practical in being able to compress images and recognize objects based on their SVD components. The SVD is truly a powerful tool that has uses in any profession that requires the analysis of continuous and random data. Its application is not even limited to object detection and compression. It is not even limited to images. The SVD factors can be used to analyze any complex system of multiple interacting variables.

For future work on the subject, we would like to investigate more complex and accurate reconstruction models. For example, we proposed a software that would fit the coordinates of a face with respect to the Eigen Basis within the distribution shown in the graph in figure 8. We also propose an extension to the data structure. Our method for compressed image storage was not optimized and it would be interesting to examine the data system/structure to represent such a sparse matrix. Generally, we would like to formalize our results in software and actually create a functional image recognition prototype. Our face scanning algorithm was not entirely optimized and a more heuristic approach could be used using a software framework like opencv.

The mathematical formulation of the SVD is a powerful tool and there is truly an unbounded amount of work to be done to utilize such a tool.

5 Acknowledgments

We would like to thank the affiliations in charge of supplying the provided datasets such as the AT&T face database[1], and mnist digit database [6]. We would also like to credit the 2004 paper Singular Value Decomposition, Eigenfaces, and 3D Reconstruction written by Neil Muller, Lourenco Magaia, and B. M. Herbst, [4] [5] from which we received inspiration and a lot of credits and information from. We would also like to Acknowledge [5] Sunny Verma, Jakkam Phanindra Krishna on their paper Image Compression and Linear Algebra, which showed us the SVD compression technique and demonstrated the ability to compress not only a gray scale image, but an 3 colored image into sub-components. Finally, we worked through problems in Matrix Analysis and Linear Algebra, by Carl D. Meyer, [3] which helped us conceptualize the SVD accurately for this paper.

Appendix

A Proof of Theorem 1.1 (See page 3)

Theorem 1.1 (Covariance of a Set of Random Vectors)**.** *The direction of maximum variance in a set of m normalized random variables of multiple dimensions (random vectors) is equal to the eigenvector u_1 of the covariance matrix $C = XX^*$ where $X = \frac{1}{n}[x_1|x_2|...|x_m]$ and the magnitude is equal to the eigenvalue σ_1 that corresponds to u_1*

Proof. In order to analyze the maximum variation of this set of vectors centered around the origin (from removing the average), we look for the vector u and the magnitude μ that shows us the direction of variance in the set of vectors.

$$\mu = \underset{\|u\|_2=1}{max} \frac{1}{n} \sum_{i=1}^{n} (u^* x_i)^2$$

For the collection of data points, the covariance matrix C can be defined as

$$C = \frac{1}{n} \sum_{i=1}^{n} (x_i x_i^*)$$

We can now rewrite μ as

$$\mu = \underset{\|u\|_2=1}{max} (u^* C u)$$

We know that C is diagonal and positive semi definite. So we can re-write C as $C = PDP^*$ where P is a unitary matrix and D is a matrix representing the eigenvalues of C. Thus,

$$\mu = \underset{\|u\|_2=1}{max} (u^* PDP^* u) = \underset{\|y\|_2=1}{max} (y^* D y)$$

where $y = U^* u$ and $\|y\|_2 = 1$. Because D is diagonal, we can write

$$\mu = \underset{\|y\|_2=1}{max} \sum_{i=1}^{n} \lambda_i |y_i|^2 = 1$$

such that

$$\sum_{i=1}^{n} |y_i|^2 = 1$$

Thus, $\mu = \lambda_1$ and $y = e_1$. Because $u = Py = Pe_1$ the maximum variance u is the right eigenvector that corresponds to λ_1 □

B Why It's Possible to Ignore the Zero Eigenvalues whilst Maintaining the Same Amount of Information

Proof. X is a $rank(r)$ matrix with singular value decomposition

$$X = U\Sigma V^*$$

and compact decomposition

$$X = U_+\Sigma_+V_+^*$$

.

Thus, X can always be rewritten as

$$X = \sum_{j=1}^{r} \sigma_j u_j vj^T$$

Note that this is likened to the spectral decomposition of a square matrix using the left and right eigenvectors and singular values. We can discard the zero singular values, because if we kept this summation to the end, we would result in all elements after r equalling zero

$$0u_{r+1}^*v_{r+1}$$

has no meaning and doesn't contribute to the final form of the matrix X. Thus, we can conclude that we can discard all zero singular values and still result in a fully lossless image compression. As for the smaller eigenvalues being discarded resulting in a lower effect on X, remember that each eigenvector u_i and v_i is part of an orthogonal matrix, so there is no "magnitude" to the normalized eigenvector. Thus, for small singular values, the resulting affect on the matrix X is negligible because the eigenvectors do not vary the magnitude of each element in the above summation.

□

C Software

In order to construct the eigenbasis and alter the images in our dataset, we used custom MATLAB code. You can view the repository at the github repository https://github.com/tlincke125/eigenfaces/.

We will not include any code here in the appendix for the sake of brevity, however, all the code is open source and welcome to be contributed to.

C.1 CreateEigenBasis.m

This is the main script that loops through a given file and creates U, d and $a_{average}$. Note that it destroys V as there is no need. LoopthroughFile loops through a given file and matches all jpg images and resizes them to fit into the large image matrix to be decomposed.

C.2 compressionMain.m

This is the main compression method. The only parameters it takes are numberKeepTerms and the file. This compresses the given image and keeps numberKeepTerms singular values in the re-composition of the image. It also writes to memory a MATLAB binary file, which we use to measure the efficiency of the compression. To finalize this, the final image we reconstruct is the one from memory.

C.3 calculateindividualerror.m

A simple script that calculates the error based on the euclidian distance between our origional image and the final image.

C.4 faceConstruction.m

A script that inputs a face and allows the user to chose which coordinate to adjust to alter the coordinates of a face represented in the eigen basis.

C.5 findfacesinlargeimage.m

A script that uploads a large image, then scans the image length-wise to find supposed faces. Results in the matrix faces, which contains vectorized (double) faces that it thought fit the basis well.

C.6 data.tar.xz

This is the data we used. It contains a set of faces and handwritten digits. The face directory is a pool of images while the digits directory organizes the digits into folders.

References

[1] The database of faces, 1994. URL https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

[2] Stephen F Cauley Bruce r Rosen Matthew S Rosen Bo Zhu, Jeremiah Z Liu. Image reconstruction by domain-transform manifold learning. 2018.

[3] Carl D Meyer. Matrix analysis and applied linear algebra, 2011.

[4] A.Pentland M.Turk. Face recognition using eigenfaces. *Journal of Cognitive Neuroscience*, pages 71–86, 1991.

[5] Jakkam Phanindra Krishna Sunny Verma. Image compression and linear algebra. 2013.

[6] Christopher J.C. Burges Yann LeCun, Corinna Cortes. The mnist database, 2010. URL http://yann.lecun.com/exdb/mnist/.