Image Compression with Singular Value Decomposition

Chris Baroni and Nick Bernstein

I. INTRODUCTION

Say you are watching whatever TV show you are currently binging on Netflix at a resolution of 1080p. Each frame is an image with a height of 1080 pixels and a width of 1920 pixels. That means there are 2,073,600 pixels on each image. Note that unless the images are in greyscale, there also is color information; for this scenario we will consider the the RGB colorspace. Each pixel has 3 values that range between 0 and 255 for each color: red, blue, and green. Each of these values require 1 byte to store, meaning each pixel requires 3 bytes to store, which, in turn, increases the estimated size of the image file to 6,220,800 bytes or about 5.93 GB. This is a massive image before we consider further that this is just a single frame. Video usually plays at minimum of 24 frames per second. That means we are now at 149,299,200 bytes of data flashing in front of your eyes every second. This is roughly 142.4 GB per second. It must also be considered that, since you are streaming the show, Internet speed is measured in bits per second and there are 8 bits in a byte. So we now have 1,194,393,600 bits per second or about 1.11 Tb per second minimum required to watch Netflix.

With all of these calculations, it should be impossible to stream videos over the Internet. In the United States, the average download speed for a fixed broadband connection is 64.17 Mbps (Speedtest). To reach the necessary speed from above, speeds would need to be increased by a factor of about twenty. So, how is it possible that Netflix, Hulu, Amazon Prime, and so many other streaming services are not only able to exist, but are so successful, that many users are canceling their cable or satellite television services? The simple answer is image compression. It is possible to condense data into a more manageable size to store and transfer. There are two main types of compression: lossy and lossless. A determination must be made as which one best suits the needs for the problem at hand. Lossy compression, as the name implies, loses information that is deemed unnecessary, causing it to be unrecoverable. Lossless compression, also as the name implies, doesn't lose information, but reformats how it is stored so that it can be unpackaged when it reaches its destination. Since the human eye doesn't require much information to see the whole picture, lossy compression is appropriately used in the scenario of video streaming.

There are many different methods of lossy compression with regard to image compression but the subject of this paper is Singular Value Decomposition. SVD is a method in Linear Algebra in which allows matrix of any size to be manipulated to yield an acceptable approximation of itself. Computationally, this requires less data to be stored or sent than the original image. Discussed in this paper are the history and mathematics behind SVD as well as a visual example of how it works to compress an image.

II. HISTORY

The history of Singular Value Decomposition goes back to the late 1800s. It was originally viewed as a mathematical curiosity, but with the contributions of five men over forty years it became the basis for many incredible tools today. The first few individuals who discovered SVD looked at it through the lens of linear algebra.

The first individual to publish work on SVD was Eugenio Beltrami in 1873. He published his work in the Journal of mathematics for the Use of the Students of the Italian Universities and used it to encourage students to familiarize themselves with bilinear forms.Beltrami's derivation was succinct, but did not cover degenerate cases. He showed a certain relaxed attitude toward the problem, showing he probably had not thought it fully through. Though, this was most likely done to easily explain it to his students.

Camille Jordan is considered the co-discoverer of SVD. His work was published a year after Beltrami, but is known to be clearly independent work. "Memoire sur les formes bilineaires" is the name of his published work on the topic, and is described as is important because it tackles the challenges that arrive in Beltramis work. Jordan goes from problem to solution with more grace by using deflation and reducing the problems from bilinear form to diagonal form using substitution.

Jame Joseph Sylvester is another person who discovered SVD independently, but in 1889. Sylvester wrote two papers and a footnote on the topic of SVD. He solved by reducing a quadratic form to a diagonal form solves SVD by reducing a quadratic form to a diagonal form.

Beltrami, Jordan, and Sylvester built up the groundwork for this interesting problem, but it wasnt until 1907 that Ernhard Schmidts work showed that SVD could be used for more. Schmidt began solving SVD with integral equations instead of linear algebra in 1907. Schmidt showed how SVD can be used to obtain optimal, low rank approximations to an operator. This use of optimization became the first step in showing the usefulness of SVD as a computational and theoretical tool. Schmidt's work became the basis for the fundamental theorem of SVD moving forward.

The last individual was Hermann Weyl who worked on SVD in 1912. Weyls work advanced on Schmidt's work and made it more usable. These advancements matured SVD into what is now known as the theory of the singular value

decomposition . The work of these men laid the groundwork for Singular Value Decomposition and its many uses today.

III. MATH

A. REVIEW OF LINEAR ALGEBRA

To understand SVD, the reader must be aware of a few fundamental aspects of Linear Algebra. If that is already the case, feel free to skip over to subsection B.

Vectors: A vector in mathematics is an object with direction and magnitude. In this paper, the notation used for vectors will be a boldfaced letter. The vector $\mathbf{v}$ is a matrix in three directions, or dimensions. For the sake of consistency, three dimensional vectors will be used for the overview of the fundamentals.

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{1}$$

A group of vectors is said to be linear independent if none of the vectors are a linear combination of the others. In the example below $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ are linear independent while $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$ are linear dependent as $\mathbf{u}_3 = \mathbf{u}_1 + \mathbf{u}_2$

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{v}_2 = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} \quad \mathbf{v}_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$\mathbf{u}_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{u}_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{u}_3 = \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The length, or magnitude, of the matrix can be calculated using an version of the Pythagorean Theorem that is expanded to more dimensions. The magnitude of a vector $\mathbf{v}$ is denoted as $\|\mathbf{v}\|$. If $\mathbf{v}$ has n elements, the magnitude is square root of the sum of the squares of each element. A vector is said to be a unit vector if it's magnitude is 1.

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_1^2 + v_1^2 + \cdots + v_n}$$

Matrices: A matrix is a rectangular array of numbers. A matrix defined as m by n will have m rows and n columns. In this paper, the notation for a matrix will be an upper case letter and the elements of that matrix will be the letter lower case letter with subscripts to represent its row and column. For example, a_{ij} would be an element of matrix A where i is the row number and j is the column number. Below is a 3 by 3 matrix for the reader's further understanding.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

To transpose a matrix, the elements are mirrored along the diagonal. This is represented with a superscript T.

$$A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

It becomes much easier to understand matrices if the reader considers a vector to be an k by 1 matrix where k is the number of elements. Take the vector $\mathbf{v}$ from equation (1) and consider it a 3 by 1 matrix. Below is the transpose of this vector, or a 1 by 3 matrix.

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \tag{2}$$

The rows and columns of a matrix can be represented as vectors. Take the first column and first row of matrix A from above. It is visually simpler to always consider row vectors as transposes. This also helps with the matrix multiplication that will be covered later in this section.

$$\mathbf{a}_c = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} \qquad \mathbf{a}_r^T = \begin{bmatrix} a_{11} & a_{12} & a_{13} \end{bmatrix}$$

The rank of a matrix is the maximum number of linearly independent column or row vectors in the matrix. The below matrix A is rank 2 since the third column is the sum of the first two and the third row is the second row minus the first.

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 1 & 0 & 1 \end{bmatrix}$$

Next we have the identity matrix which is notated with I_n. The identity is a square n by n matrix with 1s along the diagonal.

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When a matrix is multiplied by the identity matrix, it stays the same, much like multiplying by 1 in arithmetic.

$$AI_n = I_n A = A$$

The inverse of a matrix is denoted with a superscript -1. The product of a matrix and it's inverse is the identity matrix. Not all matrices are invertible. *

$$AA^{-1} = A^{-1}A = I_n$$

*For a matrix to be invertible it must be square and full rank, that is the number of rows, the number of columns, and the rank must all be equal.

Vector and Matrix Addition: Addition of vectors is simple. The elements in the same relative positions are added together. This requires both vectors to be of the same dimension. Since the same principle can be applied to adding together the row or column vectors of two matrices, matrix addition is similar.

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad \mathbf{v}+\mathbf{u} = \begin{bmatrix} v_1+u_1 \\ v_2+u_2 \\ v_3+u_3 \end{bmatrix}$$

Multiplication with Vectors: The term scalar is used to describe a number that is not a vector. It has magnitude and not direction. If a vector is multiplied by a scalar each element is multiplied by that magnitude. Multiplication between two vectors, on the other hand, is a little more difficult. There are two types of vector products, dot and cross, but this paper will only focus on the dot product. To calculate the dot product of two vectors, that must be of the same dimension, the products of each corresponding element are added together, resulting in a scalar.

$$\mathbf{v} \cdot \mathbf{u} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = v_1u_1 + v_2u_2 + v_3u_3 \tag{3}$$

Multiplication with Vectors and Matrices: Now let us consider multiplying a matrix by a vector. To do this, each element of each row is multiplied with each element of the vector to yield a new vector. In order for this to work, the number of rows in the matrix has to be the same as the dimension of the vector.

$$\begin{aligned} A\mathbf{v} &= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \\ &= \begin{bmatrix} a_{11}v_1 + a_{12}v_2 + a_{13}v_3 \\ a_{21}v_1 + a_{22}v_2 + a_{23}v_3 \\ a_{31}v_1 + a_{32}v_2 + a_{33}v_3 \end{bmatrix} \end{aligned} \tag{4}$$

Let's take this idea and look back at multiplying vectors again. However this time, consider $\mathbf{v}$ to be the 1 by 3 transpose matrix from equation (2). Notice that this result is the same as vector multiplication with the dot product in equation (3).

$$\mathbf{v}^T\mathbf{u} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = v_1u_1 + v_2u_2 + v_3u_3$$

Thinking of dot products of vectors in this way makes for a much clearer understanding of multiplication of two matrices. However the drawback is that it is not commutable like the dot notation.

$$\mathbf{v} \cdot \mathbf{u} = \mathbf{v}^T\mathbf{u} \quad \& \quad \mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T\mathbf{v}$$

$$\mathbf{v}^T\mathbf{u} \neq \mathbf{u}^T\mathbf{v} \quad \& \quad \mathbf{v}^T\mathbf{u} \neq \mathbf{u}\mathbf{v}^T \quad \& \quad \mathbf{u}^T\mathbf{v} \neq \mathbf{v}\mathbf{u}^T$$

To understand this, let's consider both $\mathbf{v}$ and $\mathbf{u}$ to be matrices. To multiply matrices, each element of each row from the left matrix is multiplied by each element of each column of the right matrix. However, in this example, each row of the vector $\mathbf{u}$ only has one element as does each row of $\mathbf{v}^T$. The result of $\mathbf{u}\mathbf{v}^T$ is a matrix, while $\mathbf{v}^T\mathbf{u}$ is a scalar.

$$\mathbf{u}\mathbf{v}^T = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1v_1 & u_1v_2 & u_1v_3 \\ u_2v_1 & u_2v_2 & u_2v_3 \\ u_3v_1 & u_3v_2 & u_3v_3 \end{bmatrix}^{\dagger} \tag{5}$$

With this new way of looking at it, recall equation (4) and notice that the elements of $A\mathbf{v}$ are the products of the row vectors of A with $\mathbf{v}$. This allows for much cleaner notation when multiplying matrices. Consider the matrices A and B with row vectors $\mathbf{a}_i$ and column vectors $\mathbf{b}_i$.

$$\begin{aligned} AB = \begin{bmatrix} A\mathbf{b}_1 & A\mathbf{b}_2 & A\mathbf{b}_3 \end{bmatrix} &= \begin{bmatrix} \mathbf{a}_1^T\mathbf{b}_1 & \mathbf{a}_1^T\mathbf{b}_2 & \mathbf{a}_1^T\mathbf{b}_3 \\ \mathbf{a}_2^T\mathbf{b}_1 & \mathbf{a}_2^T\mathbf{b}_2 & \mathbf{a}_2^T\mathbf{b}_3 \\ \mathbf{a}_3^T\mathbf{b}_1 & \mathbf{a}_3^T\mathbf{b}_2 & \mathbf{a}_3^T\mathbf{b}_3 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \mathbf{a}_3^T \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 \end{bmatrix} \end{aligned}$$

In order for AB to be possible, the dimension of the row vectors of A must equal the dimension of the column vectors of B. The dimension of the row vectors of A is the number of columns in A. Similarly the dimension of the column vectors in B is the number of rows in B. So if A is an m by c matrix and B is an r by n matrix, it must follow that $r = c$. Additionally, the resulting matrix from this product will be m by n. Similarly for BA to be possible, $n = m$ and the resulting matrix would be r by c. Note that since it is possible for AB to be possible but not BA, we cannot assume that matrix multiplication is commutable. It is also important to note that the rows and columns of each matrix are not necessarily equal, so matrix multiplication does not commute except in specific cases.[‡]

When there are matrices in an equation, it is possible to effectively divide by a matrix by instead multiplying by its inverse. Though, take extra care to ensure that the matrix is applied to the same side for both sides of the equation.

[†]This result is actually an example of lossless data compression. In this example we can represent the nine element matrix as a product of two three element vectors, effectively cutting the required data by one third. At higher numbers, the impact is much greater.

[‡]In order for matrices to commute, both of them must be diagonalizable. A diagonalized matrix is such that $A = X\Lambda X^{-1}$ where X is the matrix with the eigenvectors of A as its columns and Λ is the diagonal matrix with the eigenvalues of A along its diagonal. This is incredibly useful for analysis of certain effects over time. Since the reader is already reading this footnote, it is apparent that their curiosity has been peaked and should consult §6.2 of Strang's book.

$$AB = C \implies A^{-1}AB = A^{-1}C$$
$$I_nB = A^{-1}C$$
$$B = A^{-1}C$$

$$AB = C \implies ABB^{-1} = CB^{-1}$$
$$AI_n = CB^{-1}$$
$$A = CB^{-1}$$

Diagonal Matrices: A concept that is critical to understanding SVD is that of the diagonal matrix. It is exactly how it sounds, a matrix with values only along the diagonal and zeros everywhere else. Diagonal matrices are great when multiplying by another matrix. Let D be a 2 by 3 diagonal matrix and A be a 3 by 3 matrix with column vectors $\mathbf{a}_1$ and $\mathbf{a}_2$. Notice that the resulting of this matrix is 2 by 3.

$$\begin{aligned} AD &= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \\ &= \begin{bmatrix} a_{11}d_1 & a_{12}d_2 & a_{13}d_3 \\ a_{21}d_1 & a_{22}d_2 & a_{23}d_3 \end{bmatrix} = \begin{bmatrix} d_1\mathbf{a}_1 & d_2\mathbf{a}_2 \end{bmatrix} \end{aligned} \tag{6}$$

Orthogonal Matrices: Another type of matrix that is seen in SVD is the orthogonal matrix. Two vectors are said to be orthogonal if their product is the scalar zero: $\mathbf{v}^T\mathbf{u} = 0$. In two dimensional space, orthogonal and perpendicular are interchangeable. If vectors are orthogonal and they are unit vectors with a magnitude of one, they are said to be orthonormal. An orthogonal vector is an invertible matrix with orthonormal column and row vectors. The other characteristic of orthogonal matrices is that the inverse is equal to the transpose: $A^T = A^{-1}$.

Eigenvectors and Eigenvalues: Finally, there are special properties of square matrices called eigenvectors and eigenvalues. Take matrix A and eigenvector $\mathbf{v}$ with the corresponding eigenvalue λ.

$$A\mathbf{v} = \lambda\mathbf{v}$$

Eigenvectors are interesting in that when they are applied to a matrix, the result is a scaled version of itself, the scalar being the corresponding eigenvalue to that eigenvector. Eigenvectors cannot be the zero vector, but eigenvalues can be a zero scalar. The number of non-zero eigenvalues is equivalent to the rank of the matrix. Though the calculations involved with determining eigenvalues and eigenvectors are not complicated, they are unnecessarily convoluted and tedious for the large matrices that we will encounter with image compression. For that reason, we will be using software called MATLAB to do the work instead.

B. SINGULAR VALUE DECOMPOSITION

The theory of Singular Value Decomposition states that any matrix can be factored into a product of three matrices; U, Σ, and V^T. U and V are both orthogonal matrices, that is the transpose is equivalent to the inverse; $U^T = U^{-1}$ and $V^T = V^{-1}$. Also, the magnitude of each column or row vector is 1. Since U and V are invertible, they are square and full rank. Σ is a matrix with elements along called singular values and are represented in notation as σ_i. In the below example, $\mathbf{u}_i$ are the column vectors of U and $\mathbf{v}_j^T$ are the row vectors of V^T. For a m by n matrix A, U is m by m, Σ is m by n, and V^T is n by n. The rank of matrix A is r.

$$\begin{aligned} A &= U\Sigma V^T \\ &= \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \dots \mathbf{u}_m \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & \dots & 0 \\ 0 & \sigma_1 & \dots & 0 & \dots & 0 \\ \vdots & & \ddots & & & \\ 0 & 0 & \dots & \sigma_r & \dots & 0 \\ 0 & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix} \end{aligned}$$

The important thing to notice about Σ is that it is a diagonal matrix that is r by r with zero rows and columns surrounding it to make it m by n. So now the result from equation (6) yields the following result.

$$\begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \dots \mathbf{u}_m \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & \dots & 0 \\ 0 & \sigma_1 & \dots & 0 & \dots & 0 \\ \vdots & & \ddots & & & \\ 0 & 0 & \dots & \sigma_r & \dots & 0 \\ 0 & 0 & \dots & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix}$$

$$= \overbrace{\begin{bmatrix} \sigma_1\mathbf{u}_1 & \sigma_2\mathbf{u}_2 & \dots & \sigma_r\mathbf{u}_r & 0\mathbf{u}_{r+1} & \dots & 0 \end{bmatrix}}^{\text{This is an } m \text{ by } n \text{ matrix.}} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix}$$

$$= \sigma_1\mathbf{u}_1\mathbf{v}_1^T + \sigma_2\mathbf{u}_2\mathbf{v}_2^T + \dots \sigma_r\mathbf{u}_r\mathbf{v}_r^T + \sigma_{r+1}\mathbf{u}_{r+1}\mathbf{v}_{r+1}^T + 0$$

Notice that each $\mathbf{u}\mathbf{v}^T$ is a m by n rank 1 matrix scaled by σ, much like in equation (5). Instead of mn elements, we have $m+n$ elements. This is a much more efficient way to send or store a matrix. For the 1080 by 1920 image matrix example, instead of a 2,073,600 element matrix, we can represent the same information with one 1080 element vector and one 1920 element for a total of 3000 elements, which is so much more efficient. However, there are still r rank 1 matrices in this summation. The total number of elements in a the SVD of the original image is $r(m+n)$. Since pixel image matrices almost always have the maximum possible rank, we can assume the rank is 1080. Then there are a total of $1080(1080+1920) = 3,240,000$ elements. This is more than what was needed for the original matrix, so the reader might be wondering how this is better. The highlight of Singular Value Decomposition is that the singular values represent the contribution of that matrix to the overall matrix, the bigger the singular value, the closer that rank 1 matrix is to the original. If the singular values are sorted in descending order, eventually they are close enough to zero so that they

can be ignored without having a significant impact on the end result.

When an image is compressed using Singular Value Decomposition, the final product is an approximation of the information from the original image. Eliminating singular values that have the least amount effect on the original matrix leads to the use of fewer rank 1 matrix products to generate the approximate image. To determine the maximum amount of singular values for SVD to be more efficient, we use the below equation.

$$mn = r(m+n) \implies r = \frac{mn}{m+n} \tag{7}$$

To find the singular values, we use one of the following equations.

$$AA^T\mathbf{u}_i = \sigma_i^2\mathbf{u}_i$$
$$A^TA\mathbf{v}_i = \sigma_i^2\mathbf{v}_i$$

There is a lot of complicated math that requires a broader understanding of Linear Algebra to understand how these equations are derived. If the reader is so inclined, it can be found in §7.1 of Strang. Otherwise, all the reader needs to notice is that these equations are simply eigenvector and eigenvalue equations. The eigenvectors of AA^T are the column vectors of U with the corresponding eigenvalues being σ_i^2. Similarly, the eigenvectors of A^TA are the column vectors of V and the eigenvalues are also σ_i^2. To get the singular values, one only needs to take the square root of each eigenvalue.

IV. SVD IN ACTION

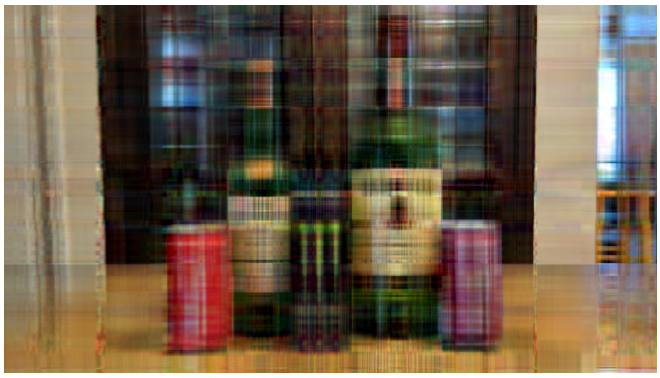

Fig. 1: This image shows the necessary materials for writing a paper on this subject.

Here is a 1080 by 1920 pixel image with a rank of 1080. Recall that each element of the matrix uses 3 bytes of data so this image is roughly 5.93 GB. We can use the below equation to determine where the SVD approximation becomes more efficient than the pixel image matrix. From equation (7) we make the determination that 691 singular values is the most we can have to be more efficient than the original matrix. This is 2,073,000 elements, which is still about 5.93GB and not much of an improvement.

Fig. 2: This is the approximate image with 691 singular values

The approximate image using a little more than half of the singular values doesn't look different at all. Now see what happens keeping only 350 singular values.

Fig. 3: This is the approximate image with 350 singular values. 1,050,000 elements or 3.00 GB

Again, there isn't any discernible difference between this approximate image and the original. Notice that this is about half of the images original size. Let's reduce the singular values again to 200.

Fig. 4: This is the approximate image with 200 singular values. 600,000 elements or 1.72GB

Here we start to notice a slight blurring of the edges and the reflections become a little duller. However, we have

eliminated 80% of the information from the SVD and it still a very close approximation to the original image. This is about a fifth of the singular values and requires. Now look at 100 singular values.

Fig. 5: This is the approximate image with 100 singular values. 300,000 elements or 879MB

Here the image is starting to become slightly grainy but the objects in the image are still easy to see. This image was formed from a matrix that only required of information.

Fig. 6: This is the approximate image with 50 singular values. 150,000 elements or 439 MB

At 50 singular values, there is definitely lost quality but it is still pretty close to the original image.

Fig. 7: This is the approximate image with 25 singular values. 75,000 elements or 214 MB

25 singular values is where the the quality of the image becomes too low for reasonable use. The information in the matrix is too far from the original for a clear picture. Even though the picture is low quality, objects in the image are still clear enough to be distinguished.

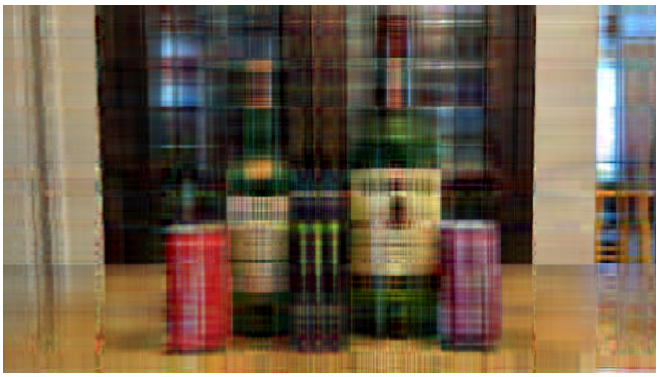

Fig. 8: This is the approximate image with 10 singular values. 30,000 elements or 87.9 MB

Now with the lowest amount of singular values, it is easier to see what exactly is happening when the decomposed matrix has been reassembled. Notice that though the image has blurred, there are clear horizontal and vertical lines.

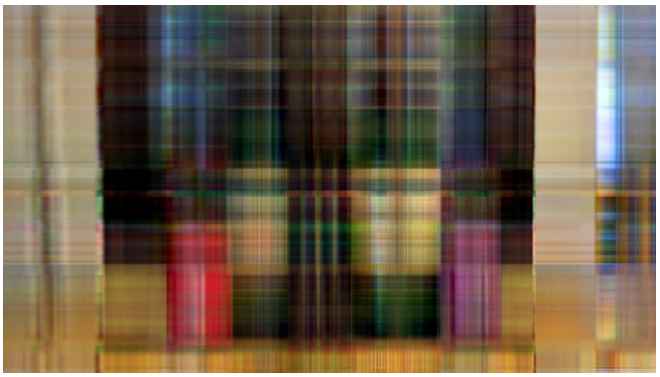

Fig. 9: This is the approximate image with 5 singular values. 150,000 elements or 43.9 MB

This image with only 5 singular values shows the linear combinations. Notice how the items in the picture have turned into blocks of color. Recall that each image is made with 3 matrices on top of each other each representing a color: red, green, and blue. Each of these matrices have values that range from 0 to 255. However, each row and column only has 5 elements that are linearly independent, meaning that the other values in the matrix are combinations of these 5 numbers. To visualize this better, here are a few numerical examples of this effect.

This is an 8x8 matrix of random integers

$$\begin{bmatrix} 3 & 3 & 0 & 2 & 8 & 3 & 2 & 5 \\ 7 & 5 & 5 & 1 & 5 & 7 & 1 & 6 \\ 2 & 7 & 1 & 3 & 5 & 1 & 7 & 2 \\ 3 & 2 & 2 & 6 & 1 & 6 & 6 & 5 \\ 5 & 5 & 6 & 2 & 5 & 6 & 4 & 1 \\ 6 & 4 & 3 & 7 & 4 & 4 & 4 & 3 \\ 5 & 0 & 4 & 8 & 4 & 7 & 6 & 2 \\ 2 & 3 & 8 & 4 & 4 & 1 & 3 & 2 \end{bmatrix}$$

This is an approximation of the matrix with 2 singular values

$$\begin{bmatrix} 3 & 3 & 3 & 3 & 4 & 4 & 3 & 3 \\ 5 & 4 & 4 & 5 & 5 & 5 & 5 & 4 \\ 4 & 3 & 3 & 4 & 4 & 4 & 4 & 3 \\ 4 & 3 & 4 & 4 & 4 & 4 & 4 & 3 \\ 5 & 4 & 4 & 4 & 5 & 5 & 4 & 3 \\ 5 & 4 & 4 & 5 & 5 & 5 & 4 & 4 \\ 5 & 4 & 4 & 5 & 5 & 5 & 5 & 4 \\ 4 & 3 & 3 & 3 & 4 & 4 & 3 & 3 \end{bmatrix}$$

Notice how the elements of the approximate matrix don't vary in value as much as the original matrix. There are also blocks of the same number, much like there are blocks of the same color in the rank 5 approximation of the image.

V. CONCLUSIONS

Singular Value Decomposition is a powerful tool when it comes to image compression. In the example above, the image was still very clear with only 200 of the 1080 singular values preserved. In the end, this image only needed 879 MB which is about an 85% reduction from the 5.93 GB for the original image. However, the highlight of SVD is it's ability to enhance other compression algorithms. The images in this paper were converted to the JPEG format. The below image shows the effect that the approximation had on the sizes of the files. Again, we see about an 85% reduction in file size.

Fig. 12: The effect of SVD on JPEG compression.

Singular Value Decomposition working in tandem with other data compression algorithms give us the ability to many things, among which is streaming movies and TV shows. On that note, the last episode of House M.D. left off with a cliffhanger so I'm going to get back to that.

REFERENCES

[1] Andrew Gibiansky. 2013. Cool Linear Algebra: Singular Value Decomposition. (May 2013). Retrieved June 22, 2018 from http://andrew.gibiansky.com/blog/mathematics/cool-linear-algebra-singular-value-decomposition/

[2] Anon. Documentation. Retrieved June 23, 2018 from https://mathworks.com/help/matlab/index.html

[3] Carla D. Martin and Mason A. Porter. 2012. The Extraordinary SVD. *American Mathematical Monthly* 119 (December 2012), 838851. DOI:http://dx.doi.org/https://doi.org/10.4169/amer.math.monthly.119.10.838

[4] Gilbert Strang. 2016. *Introduction to Linear Algebra* 5th ed., Wellesley: Wellesley-Cambridge Press.

[5] G.W. Stewart. 1993. On the Early History of the Singular Value Decomposition. *SIAM Review* 35, 4 (December 1993), 551566.