SVD Image Compression

Rachel Brough
Scott Hoge
August Masquelier
Ricardo Sonsini

April 9, 2018

1. Introduction

With advances in technology, the cost to store large amounts of data has also seen a drastic decrease. In 1980, storing 26 megabytes of data on a traditional hard disk would cost somewhere around $5,000 USD which results in about $193,000 per gigabyte. By 2009 one could purchase a single terabyte hard drive for a mere $74.99, or about $0.07 per gigabyte[(1)]. This drastic decrease in the cost of storage might result in the conclusion that storing substantial amounts of data, computerized images for example, is cheap. However, this is largely dependent on ones definition of the word substantial. In this paper, we will discuss the ways that computers process images on a lossless basis, and compare this to how they can be compressed into a lossy format to trade image quality for file size.

2. Applications of SVD

As discussed above, data storage might have become much simpler with the decrease in the cost of traditional hard drive storage, yet this is no reason to disregard the way that images are represented on computers themselves. Traditionally, the color spectrum consists of combinations of the three main additive colors, namely red, green and blue that can be used to create somewhere around 16.8 million different colors. Each of these three values ranges from 0 to 255 such that they can each be stored in a single byte, which is convenient when understanding how exactly images are displayed on a computer screen.

Notably, a monitor consists of thousands of small pixels that each display a single one of these 16.8 million color possibilities. In order to set a pixel to a specific color, the RGB values of each of these pixels must be stored and set by the computer to produce a final image. There are many different ways to store these images with the first and least common being completely lossless image storage. For example, the *.tiff* image file format utilizes full image representation which means that each and every RGB value is stored in

the computer's memory[2]. As a result, an image consisting of 5400 x 3600 pixels would require roughly 59 megabytes of storage. When dealing with terabytes of storage this is not a significant amount of memory, but when storing millions of images this quickly becomes an issue.

All things considered, computer scientists and mathematicians alike have developed different methods to store images in computer memory such that they take up significantly less storage. Of course, you cannot directly represent the RGB values of every single pixel while still decreasing the amount of memory it requires, so these compressed formats are considered lossy as they do not store the image in its entirety. The goal with any good compression scheme is to find the right ratio of image quality to file size such that the image retains most of its quality while still decreasing the its size by a significant margin. One such method, coined Singular Value Decomposition uses the fact that images can be viewed in a similar manner to matrices and compressed using linear algebraic techniques.

To elaborate, SVD requires that images be stored in computer memory as a matrix such that they can be operated on in a linear algebraic fashion. To simplify this process, we will consider black and white images such that each pixel is represented by a single value from 0 to 255 rather than each pixel requiring three different values like in the traditional colored RGB format. As a result, each pixel can then be represented with 8-bits or a single byte. Once the image has been stored in the computer as a matrix of black and white pixel data, we can then use SVD to compress the image.

3. Motivation for SVD

Given an image with a specified resolution, we can represent the information within the image with a matrix A. The dimensions of this matrix relate the the resolution of the image itself, say that both can be represented by $n \times d$. We can reduce the dimensionality of this image by re-representing vectors in d dimensions as vectors in r dimensions where $r < d$. Note that we can easily do this dimension reduction by reducing the rank, or number of independent columns, of a matrix to achieve this [4].

Something important to note is that our matrix of information can be factored into an $n \times r$ matrix, and a $r \times d$ matrix such that all of A's columns can be written as linear combinations of the columns in the $n \times r$ matrix and its rows are linear combinations of the rows of the $r \times d$ matrix. Hence, while the original matrix A requires $(n \cdot d)$ numbers to be represented, the factorized matrices can be described by $r(n + d)$ numbers and as a result, a small value of r will be a much simpler representation of the same information[4].

This is where the idea of compression comes in, as we are reducing the rank, r, of a given matrix and therefore its size. In the case of an image, this is useful because we can trade the quality of an image in favor of the size it takes up on our computer.

4 . Explanation of the SVD Process

As we understand, a matrix A which we can say holds black and white image information, is diagonalizable if it can be rewritten as:

$$A = PDP^{-1}$$

where P is an invertible matrix and D is a diagonal matrix, meaning that only the elements on the diagonal of D can be non-zero. From the equation above, we can also conclude that the following is also true:

$$AP = PD$$

Then, if we consider separate columns of P and denote each column by p_i and each diagonal entry in D by λ_i we are left with the following:

$$\mathrm{D} = \begin{bmatrix} \lambda_{11} & 0 & \dots & 0 \\ 0 & \lambda_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_{dn} \end{bmatrix}$$

$$\mathrm{P} = (p_1 \ p_2 \ \dots \ p_n)$$

Such that we get:

$$\mathrm{PD} = (p_1\lambda_1 \ p_2\lambda_2 \ \dots \ p_n\lambda_n)$$

And with this matrix multiplication, PD, we can conclude that each diagonal value in D picks its associated column in P and scales it. We can then conclude that the columns of P must be eigenvectors of A and the diagonal entries of D must be the eigenvalues of A. In order for any of this to hold, it must be true that P^{-1} exists, which means that the matrix P must be square[3]. As this is not the case with most image files we can generalize this process of diagonalization to apply to matrices of any shape through SVD.

SVD is based on the fact that any matrix A which might be non square, has some A^TA that will result in a square matrix that we can then extract eigenvectors from. Note that as a result, these eigenvectors form an orthonormal basis. If we take these eigenvectors and denote them as the columns in a matrix V, we can can take the transpose of V which results in V^T.

$$V^T = eigenvectors(A^TA)^T = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Each of these eigenvectors will have an associated eigenvalue, which we can denote with λ. Then, we can take a modification of these eigenvalues and insert them on the diagonal of a matrix D that has the same dimensions as A and pad the rest of it with zeroes. This modification can be denoted as:

$$\sigma = \sqrt{\lambda}$$

We can then insert these into the matrix D which results in the following for a matrix A that is *3 x 3*:

$$D = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{bmatrix}$$

Finally, we can form the last matrix U by performing the following:

$$U = \begin{bmatrix} \frac{1}{\sigma_1} Av_1 & \frac{1}{\sigma_2} Av_2 & \dots & \frac{1}{\sigma_n} Av_n & NUL(A^T) \end{bmatrix}$$

Where NUL(A^T) represents Strang's Special Solutions to the nullspace of A^T.

Then, the result of $A = UDV^T$ will result in the canceling out of distinct eigenvectors due to orthogonality and dot product, and non-distinct eigenvectors doing nothing in terms of changing the entries of A. As a result, this new matrix A will be a less noisy version of the original based off its new rank k in comparison to the rank of the original matrix.

5. Information Retention

The goal of SVD image compression is to summarize a given matrix of image information into one that is similar but contains fewer values. In terms of compression, this is an appealing characteristic because it allows us to decide exactly how much of a given image we want to summarize. For example, if we wanted to retain 90% of the information stored in an image matrix we would have an SVD matrix similar in size that

contains very little loss of the original image. In contract, we could choose to only retain 5% of the image matrix information in favor of image storage size over image quality and suffer very large loss rates compared to the original image.

As relates to SVD image compression, we can choose the relative loss rates based on the number of singular values, or rank *k*, used to calculate the resulting SVD matrix. The largest singular values come first, so summing them up and dividing by the total number of singular values can be used to ultimately decide the number of singular values to be kept to attain the desired retention percentage. The approximated matrix can then be used to form a compressed version of the original image. See the below section for examples of the application of SVD with different rank values on a given image.

6. Samples

As an example, we can perform SVD image compression on a sample image using code written in Matlab. The code used to complete this process and display the images and graphs below is included with the submission of this paper.

Figure 1. The image that will be used to demonstrate SVD.

See below for examples of SVD using different rank values.

The rank of each image denotes the number of singular values in the SVD matrix approximation. For an analysis of the implications of different rank values and the resulting size of the image, see the below section.

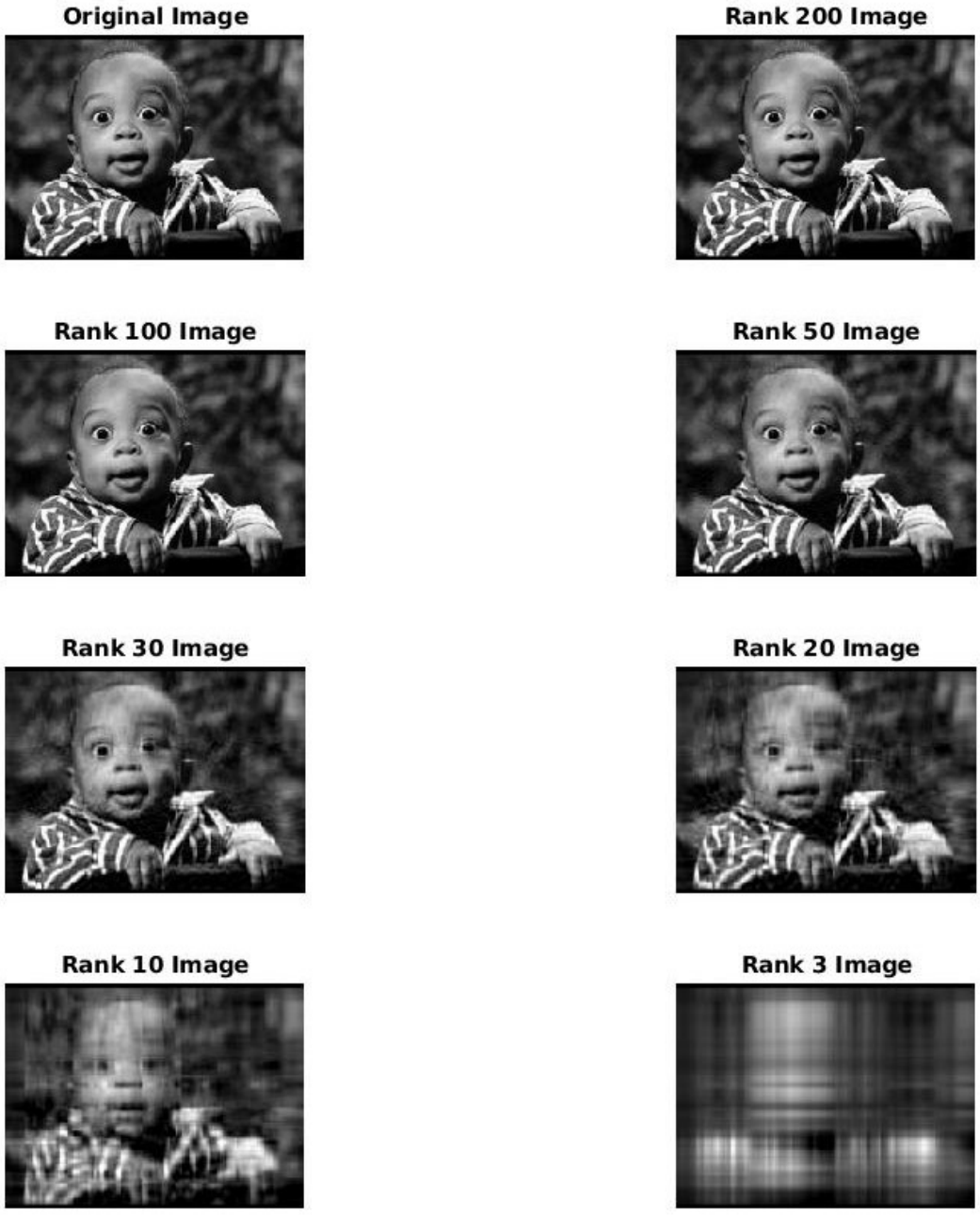

Figure 2. The SVD image with different rank values[3].

7. Analysis

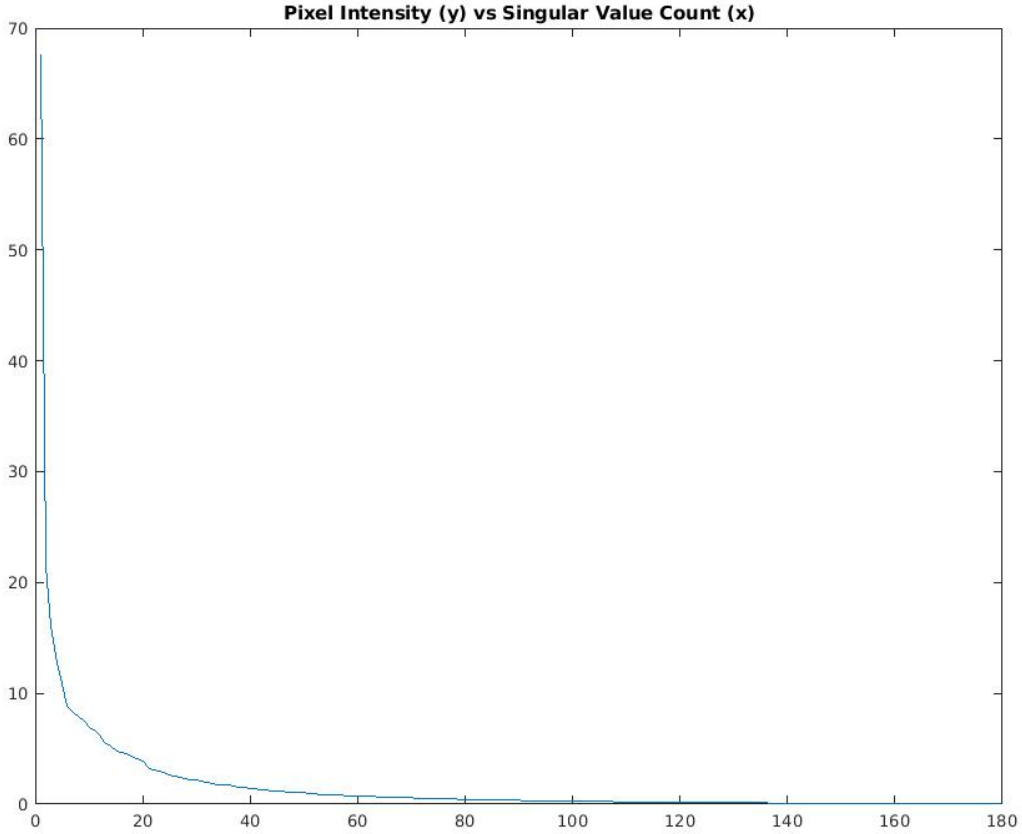

Figure 3. A graph showing the count of each singular value versus the relative pixel intensity representation of each value[3].

After converting the image to a matrix where elements represent pixel intensity at each location in the image, we then want to determine which singular values contain relevant information with regards to the image. From the graph above, we can determine that the first 20 or so singular values contain a vast majority of the information about the image matrix, which means it would create a decent representation of the original image.

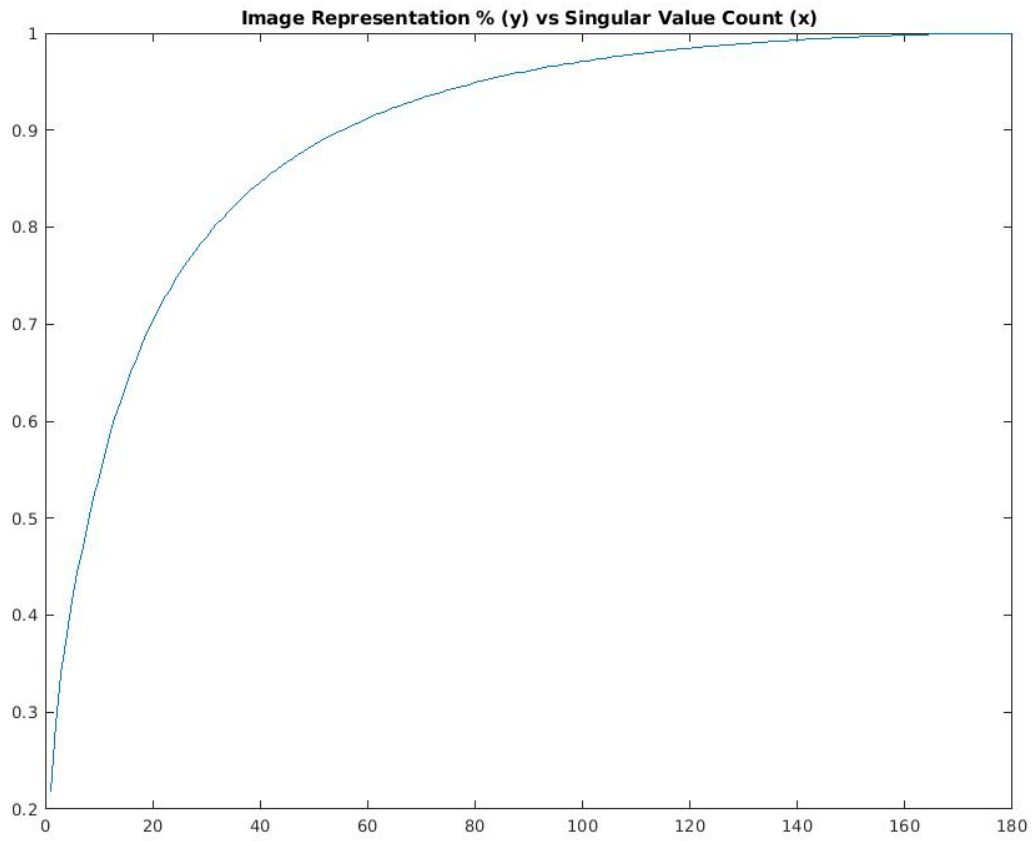

Figure 4. A graph showing the percentage of information contained by a given number of singular values[3].

If we want a more detailed way to choose a rank value, we can plot the cumulative percentage that shows how much information each singular value contains. Notice that at roughly 60 singular values we can obtain 90% of the information in the original image. We can sample different 'rank' values to show the relative image quality for each, like what was shown above.

8. Conclusion

Even with the drastic decreases in the cost of traditional hard drive storage over the last century, it is still important that we utilize the space we have available in the best possible manner. In the case of image storage and many other common file types we can achieve this goal by using some sort of compression method to reduce the overall size of the image or file. Sometimes, like in the case of SVD image compression, this results in a loss of information in the image to effectively trade quality for file size. The benefit here is that SVD image compression allows for the user to specify the rank of the resulting

decomposition which is useful for achieving the desired ratio. Amongst many other things, linear algebra has found a way into most parts of our lives and ultimately provides many useful tools for problems similar to this one.

9. Notes

The source code to produce the SVD images and graphs can be found under resource [(3)] below, as cited where used above. In order to run the code on different images, simply have an image in the same directory and change the filename on line 1 of the compress.m file.

10. Works Cited

[1] Komorowski, Matt. "A History of Storage Cost." *Mkomo.com*, 8 Sept. 2009, www.mkomo.com/cost-per-gigabyte.

[2] "TIFF." Wikipedia, Wikimedia Foundation, 9 Apr. 2018, en.wikipedia.org/wiki/TIFF.

[3] Gibiansky, Andrew. "Cool Linear Algebra: Singular Value Decomposition." Andrew Gibiansky Blog, 29 May 2013, andrew.gibiansky.com/blog/mathematics/cool-linear-algebra-singular-value-decomposition/.

[4] Roughgarden, Tim, and Gregory Valiant. "The Singular Value Decomposition and Low-Rank Value Matrix Approximations." CS168, Stanford, 27 Apr. 2015, theory.stanford.edu/~tim/s15/l/l9.pdf.