Image Compression Using SVD and DCT Algorithms

Steven Huang, William Freedman, Daniela Shoham

Abstract:

This project covers the usage of matrices in connection with image compression algorithms. We analyze the relationships between compression and image quality, observing the tradeoffs between the two. Our approach utilizes Singular Value Decomposition (SVD), a lossy image compression algorithm, and Discrete Cosine Transformation (DCT), a lossless algorithm, to maintain the overall image while using significantly less memory. We find the relationship between blur degree and the compression ratio of images, allowing for an empirical understanding of the memory constraints that certain levels of image quality necessarily require and vice versa. Our data can be used to optimize image compression by determining the parameters that generate the best information storage vs. image clarity results. Additionally, we aim to constrain the compression by a reasonable runtime. If the perfectly optimal parameters and algorithm takes an unreasonable time to execute, we aim to find a good solution in reasonable time.

I. INTRODUCTION

Research Question

How can the usage of matrices in connection with image compression algorithms be used to predict image quality given a target compression amount, or predict a target compression ratio given a constraint on image quality?

Background/Relevance of Study

With the rise of the digital age and the popularization of the Internet, especially considering social media platforms, images have become an integral part of modern technology. Due to a spike in visual information sharing, researchers have developed a wide variety of image compression methods to enable faster load times, minimize memory requirements, enable rapid image sending, and overall improve user experience (Pu, 2006). Because of the uniquely large memory requirements of images when compared to text, as well as the strain large amounts of image metadata images require places on network channels, image compression has the potential to significantly decrease the resource requirements of large-scale networked systems (Khobragade, 2014). Fortunately, images have a series of relatively unique properties that enable specific compression algorithms that couldn't be applied to other forms of data, most notably text. The first of these is an inherent tolerance for data loss. Because an image is primarily considered by a human user as a whole, the results of data loss such as blurriness and small-scale data corruption aren't as significant[1]. While the loss of a byte of memory representing text would result in a totally different character being rendered, the same memory loss would have a negligible effect on a user's assessment of an image. This makes lossy compression a uniquely viable solution when considering image compression. In addition, the mathematical methods developed for the analysis and manipulation of matrices in Linear Algebra provide a unique set of tools when considering images. Because images are represented in memory as pixel matrices,

[1] The validity of this property may have to be re-evaluated when images are considered as training data or input to Machine Learning Models, where differences in an image that a human can't notice may have a notable impact on the effectiveness of these models.

where each pixel is a 3-tuple of RGB values, Linear Algebra emerges as an obvious place to look for image compression algorithms. Two notable mathematical methods emerging from this field are Singular Value Decomposition (SVD) (Kahu, 2013) and Discrete Cosine Transform (DCT) (Gupta, 2012).

II. METHOD

A. The first image compression algorithm we analyzed used Singular Value decomposition (SVD). SVD is a matrix factorization algorithm, decomposing a matrix M intro three matrices, usually denoted U, Σ and V^T. Submatrices of these three are then recombined into a final image. We will use the matrix $M = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix}$ as an example of the process (Nicholson, 2019). First, we take the transpose of M, $M^T = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$ and multiply the two matrices together to produce a new matrix $Y = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$. Next, we find the eigenvalues and their corresponding eigenvectors, 3 with the eigenvector $\begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$, 1 with the eigenvector $\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$ and 0 with the eigenvector $\begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$, and find the square roots of the non-zero eigenvalues, $\sigma_1 = \sqrt{3}$ and $\sigma_2 = 1$. These values are known as the singular values of the matrix. The Σ matrix is then constructed with these values along the diagonal, $\Sigma = \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$. The matrix V is formed from the unit vectors of the eigenvectors found above, so $V^T = \begin{bmatrix} \frac{\sqrt{6}}{3} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{6}}{6} \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{3}}{3} & -\frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} \end{bmatrix}$. The columns of U are calculated with the following formula: $u_i = \frac{1}{\sigma_i} Mv_i$ where σ_i and u_i are the root of the eigenvalue and the corresponding eigenvector from above. Using this formula,

$u_1 = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{bmatrix}$ and $u_2 = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$, so $U = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$. This process produces the decomposition $M = U\Sigma V^T$.

B. The second image compression algorithm we analyzed utilized Discrete Cosine Transform (DCT). DCT is a lossless operation (meaning that it will get rid of unnecessary metadata first) that decorrelates neighboring pixels in order to reduce redundancy while perfectly reconstructing the original image from the uncorrelated elements. Since we used DCT with an image, we will be focusing on 2-D DCT. Given a signal with *N* terms $X[n],\ x \in \mathbb{R}^N$, we can find DCT with *N* basis vectors as such:

$\psi k,l[n,m] = \alpha[k]\alpha[l]cos(\frac{\pi(2\pi + 1)k}{2N})cos(\frac{\pi(2m + 1)l}{2N})$

where n,m = 0,1,..., *N*-1, and $\psi k,l$ is the *[k,l]*-th basis vector with *K* and being the parameter values.

Since the cosine transform is done for each pixel, given an image *X[n, m]*, the coefficients *Y[k,l]* applied to each basis vectors is given by:

$Y[k,l] = \alpha[k]\alpha[l] \sum_{n=0}^{N-1} \sum_{n=0}^{N-1} X[n,m] cos\frac{\pi(2n + 1)K}{2N} cos\ \frac{\pi(2m + 1)l}{2N}$

These coefficients *Y[k,l]* represent the contribution of each basis vector to the original image. The DCT transformation is then applied to each pixel of the image to obtain transformed coefficients that preserve essential information while facilitating compression. Consider a 4x4 grayscale image represented by the following matrix M:

$$\begin{bmatrix} 50 & 60 & 70 & 80 \\ 60 & 70 & 80 & 90 \\ 70 & 80 & 90 & 100 \\ 80 & 90 & 100 & 110 \end{bmatrix}$$

The resulting coefficients after applying DCT for each *[k,l]* result in the following matrix:

$$\begin{bmatrix} 3690 & -41.5 & 0 & 0 \\ -46.5 & 1.5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The 0 values in this matrix indicate that the pixels they represent contribute less to the final image and are redundant for compression purposes.

Once the DCT calculations have been applied to produce the new matrix, we can apply the inverse DCT to reconstruct an image using the following function, given DCT coefficients, say, $F(u,v)$:

$f(x,y) = \frac{2}{N}\Sigma_{u=0}^{N-1}\Sigma_{v=0}^{N-1}C(u)C(v)F(u,v)cos(\frac{(2x+1)u\pi)}{2N})cos(\frac{(2y+1)v\pi}{2N})$ where $C(u) = \frac{1}{\sqrt{2}}$ when $u = 0$ and $C(u) = 1$ otherwise. Breaking down each variable in this function:

$\frac{2}{N}$: This is a scaling factor to ensure that the IDCT operation is scaled properly to reconstruct original values.

Summation: The summation of the u and v values from 0 to N - 1 represents the contribution of frequency components in the reconstruction of data.

$C(u), C(v)$: These are scaling factors. These scaling factors adjust the amplitudes of the cosine terms, ensuring that the DC component (frequency component with u=0 and v=0) is scaled properly.

$F(u,v)$: The DCT coefficients represent the magnitude of the frequency components in the frequency domain. Each term in the summations statement is multiplied by their corresponding DCT coefficient, which means that they contributed a frequency component to the image reconstruction.

The two cosine functions correspond to the spatial frequencies in the x- and y-directions, respectively. They capture the oscillatory behavior of the

frequencies, with u and v contributing to the reconstruction with different spatial oscillations.

C. The last mathematical method used was the Laplacian algorithm, used to quantify the sharpness of an image (Pech-Pacheco, 2000). Intuitively, the Laplacian represents the second derivative of a the brightness of an image with respect to distance, viewing an image, and therefore a matrix, as a surface in 3D space with the x and y coordinates being the position of the pixel and the z coordinate being the value of the entry in the matrix. When we apply the Laplacian, we are able to quantify transitions in brightness in the image. This representation encodes variations in intensity that correspond to the perceived sharpness in the image. We then compute the variance of the Laplacian values across the entire image, serving as a quantitative measure of the overall level of detail or blurriness. A higher variance indicates regions with more pronounced intensity changes and, consequently, a sharper image, while a lower variance corresponds to areas with smoother transitions, indicative of blurriness. Despite its name indicating otherwise, blur degree actually correlates with intuitive assessments of sharpness, with a higher blur degree corresponding with a less blurry image. Since external sources refer to this value as blur degree anyway, we choose to follow this pattern (Pech-Pacheco, 2000) (Rosebrock, 2015). Sample images and their blur degrees are shown in Figures 1 and 2.

Figure 1. Blurry image of a (very cool) man with a blur degree of 43.48

Figure 2. Sharp image of a sharp man with a blur degree of 443.5

III. IMPLEMENTATION

A. All of the code used in the completion of this project was written in Python and can be found at the Github repository below (Freedman, 2023). Python was chosen for its relative ease of development, as well as the existence of the numpy and cv2 libraries, both of which provide efficient implementations of the mathematical processes used in this project. The Pillow and Matplotlib libraries were also used for their image processing and graphing capabilities, respectively.

B. For SVD, numpy only provides an implementation of the pure linear algebra method used for the decomposition of a matrix, meaning additional boilerplate code is needed to convert between an image file as stored in memory and the matrix representation of that image. First, the program needs to split the image into its separate color channels, converting from a matrix of 3-tuples each representing an rgb value to three separate matrices. The numpy SVD algorithm is then run on each of these images, resulting in 9 total matrices, the U, Σ and D matrices discussed above for each of the three color matrices. The input parameter k is then used to create three new matrices using only first k singular values in the decomposition (Harrington, 2012). We then construct the final image matrix by essentially stacking the three matrices, where the final matrix M can be described as[2]: $M_{i,j} = (Red_{i,j}, Green_{i,j}, Blue_{i,j})$. This new matrix is then converted into an image file.

C. There are two versions of the DCT compression, one of which handles images in grayscale, and one of which handles color images (Pham, 2017).

   1. First, the grayscale DCT code implementation will be described. The code takes in an image, and applies the 2-dimensional DCT algorithm, which is built into the Python package scipy.fftpack to the image. It is possible to use this method, as it is being done on the 8x8 pixel blocks that are generated from the image. Next, we look at the quantized DCT coefficients of the image, which we get from the first

[2] The specific ordering of the red, green and blue values could theoretically change based on how the operating system handles screen rendering, but this order is consistent with how most modern operating systems render images.

step, and take in an input that determines how much data we keep from the original image (Watson, 1994). For the DCT-grayscale algorithm, the algorithm would take in the max number of DCT coefficients that would be kept after the DCT transformation. For the purposes of our project, it was set to 73900, as this value is close to the max number of coefficients that could reconstruct our images almost identically. It creates images from 0 to 73900 in increments of 100, meaning that as this number increases, more coefficients would be reconstructed, and thus, more of the data of the original image would be brought back. Thus, the higher that input value, the less compressed the output image will be. After doing this, we apply the inverse 2-dimensional DCT (IDCT) calculations on the image, map it to a gray color map, and save it. The step between the DCT and IDCT is necessary, because without it, it would just be DCT and IDCT, bringing back the same image with changing the size, meaning without actual compression.

2. The color image DCT code implementation was slightly different. Because color images have channels for red, green, and blue (RGB), we had to split the image elements into three different channels first. Then, we applied DCT to each channel, utilizing the same method of using the scipy library as in the grayscale. Then, the function takes in a parameter granularity, which is the number of times we run the algorithm on the image, with compression factors increasing in increments of 1/granularity. Each increment of 1/granularity (compression factor)[3] indicates how much data is lost from the compression, or how much we compress the image. For example, if the granularity is 10, then it will run the algorithm using compression factors from 0 to 1 incrementing by 0.1 each time. The algorithm determines how many 0s should be in each pixel, or how much loss of data there is in each pixel by multiplying this value by the rows and columns and setting 0 as values. The higher the compression factor, the more data is lost. We then apply the 2-D inverse DCT algorithm, which allows us to reconstruct the

[3] Note that compression factor and compression ratio are different values. Compression factor refers to one divided by the input to the DCT function (which results in a proportion showing up much data to keep), whereas compression ratio is a comparison of the sizes of the original and compressed images.

image, and save it. Example outputs of all three compression algorithms are shown in Figures 3 through 6.

D.

Figure 3. An uncompressed image of a tree in a forest.

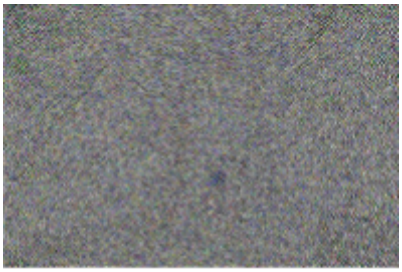

Figure 4. Compressed tree (DCT-RGB algorithm, compression factor = .2)

Figure 5. Compressed tree (SVD algorithm, k=20)

Figure 6. Compressed tree (DCT-greyscale, compression factor = 1/10200)

IV. RESEARCH METHODS

A. Our research goals are to analyze the relationship between the compression factor and the loss in quality of an image. Compression factor can be measured directly by taking the ratio of the size of the compressed image to the size of the original image, and we will use blurriness as discussed above as a value representative of image quality. Limitations of this assumption are discussed below. We generated compressed images using all three algorithms discussed above, then calculated and plotted the compression ratios and blur degrees for various compression parameters. For the purpose of our graphs, compression ratio and blur degree are both calculated as ratios of their respective values in the new image to the value in the old image. This means all data generated has been considered in the context of the original image, preventing a single particularly blurry or sharp image or a massive or incredibly small image skew the data. In addition, in order to prevent any error due to Python's os.save method and any implicit compression it may do when

creating and saving an image file, all source images are decomposed into matrices and reassembled in the same way as the compressed images.

V. RESULTS & ANALYSIS

A. As expected, both compression algorithms display a tradeoff between resulting image size and image quality. Figures 7 through 11 show the graphs generated using various datasets for both compression algorithms. This empirical data confirms that, while having different mathematical meanings, the k value in SVD and the pixel number value in DCT can both be understood as roughly corresponding to "amount of information retained," and as such are inversely correlated with compression ratio and are positively correlated with blur degree. These graphs can be used to approximate the right values to use for compression while maintaining specific constraints. For instance, if a developer needed to compress a set of images to 80% of their original size and wanted to use SVD for that process, our data would indicate that the developer can expect a blur degree of around 1000, allowing the developers to make informed decisions about the memory requirements of their applications.

B. While both graphs demonstrate a positive correlation between compression ratio and blur degree for all values, this trend only truly becomes noticeable at a compression ratio of around .7 for both algorithms. Before .7, increases in compression ratio have a negligible effect on the blur degree of the image, meaning both algorithms can compress images up to 70% while maintaining a reasonable amount of image quality.

C. Our data also demonstrate that SVD, on average, performs better than DCT for the metrics we tracked. In the .7 to 1.0 compression ratio range discussed above, SVD produced images with blur degrees approximately ranging from .1 to .5, while the images compressed using grayscale DCT ranged from .01 to .07 over the same interval. Since blur degree is our proxy for image quality, SVD produced images of significantly higher quality than DCT.

D. While we initially planned on analyzing the runtime of the algorithms we used, the actual observed runtimes ended up being negligible and roughly equivalent. For all of the tests we ran, the time needed to compress an individual image was small enough, as well as not being noticeably different from any other observed runtimes in the test, that we decided this was not a significant factor when deciding on an image compression algorithm and its parameters.

E. One specific interesting result is the behavior of SVD compression at high values of k. At a certain point, when k is greater than approximately 2^8, the resulting image actually has a greater size than the original image, meaning SVD functionally acts as a decompressor. In our analysis we didn't observe any predictors of this specific k value, meaning that finding this value in production would likely require empirical testing.

F. One limitation of our methods is the reliance on blurriness via the Laplacian operator as a proxy for image quality. While this is often an important value to consider in image compression, as a lossy compression algorithm is very likely to produce a blurrier image, it doesn't account for all of the possible side effects of compression. One notable failing of this methodology is its lack of accurate performance, and even counterproductivity, when considering data corrupted by compression. This problem specifically occurred when testing DCT compression on full RGB images. For all high data loss proportions, RGB DCT produced images like the one shown in Figure 4. These images display high contrast, tend to be grainy, and usually display some repeating pattern. Notably, none of these factors are captured by our Laplacian blur function, and may even produce an image that is technically less blurry due to the high contrast and saturation of many of these images. This problem is demonstrated in Figure 7, where the blur degree is over 1, meaning the resulting image was clearer, for all compression ratios. Accounting for this problem would require a more in depth analysis of the image itself, involving tracking the changes in color of specific locations in the image, but that analysis is beyond the scope of this paper.

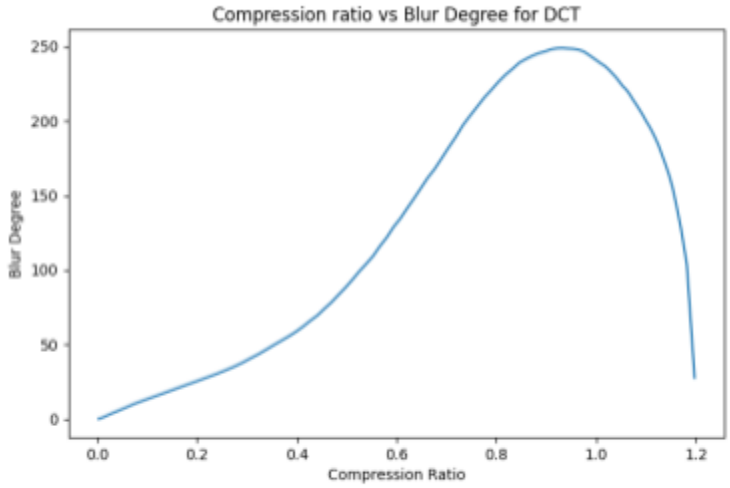

Figure 7. Graph of blur degree (blur degree of compressed image/blur degree of original image) vs. compression ratio (size of compressed image / size of original image) using the DCT-RGB algorithm on a color image

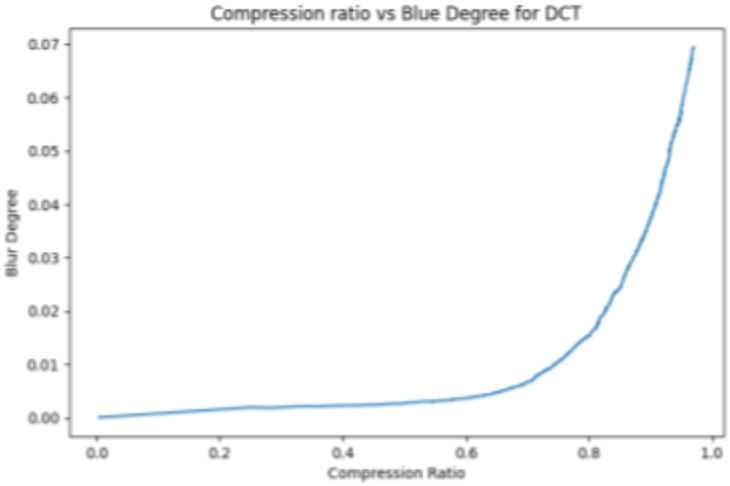

Figure 8. Graph of blur degree (blur degree of compressed image/blur degree of original image) vs. compression ratio (size of compressed image / size of original image) using the DCT-greyscale algorithm on an image

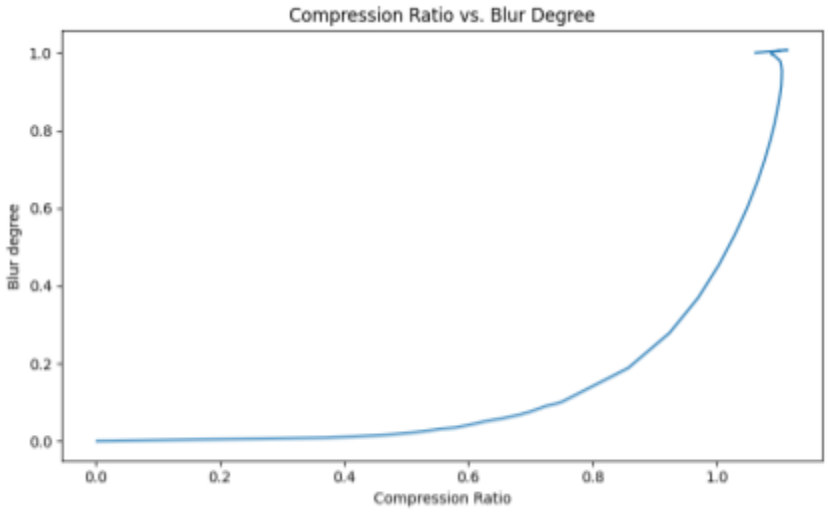

Figure 9. Graph of blur degree (blur degree of compressed image/blur degree of original image) vs. compression ratio (size of compressed image / size of original image) using the SVD algorithm on an image

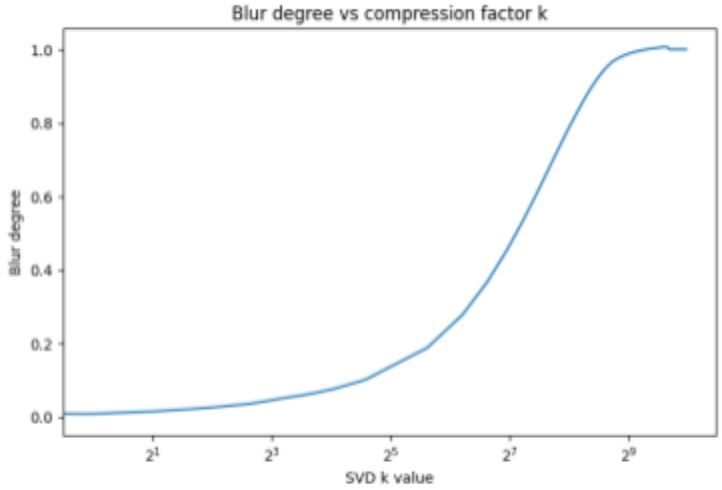

Figure 10. Graph of blur degree (blur degree of compressed image/blur degree of original image) vs. k value using the SVD algorithm on an image

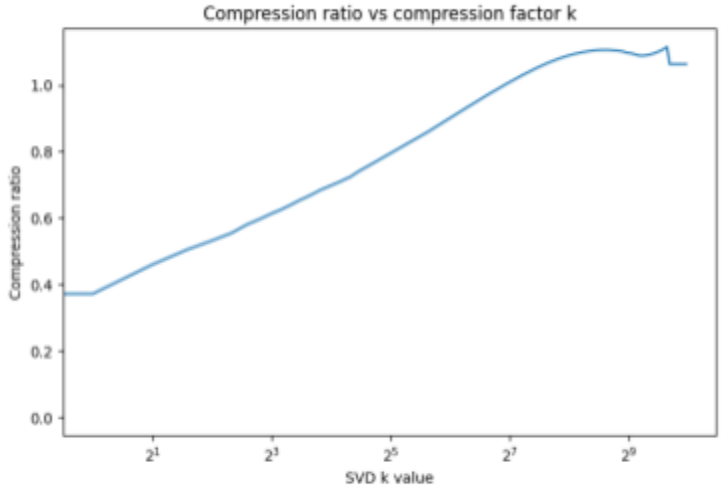

Figure 11. Graph of compression ratio (sizeof compressed image/size of original image) vs. k value using the SVD algorithm on an image

Works Cited

Freedman, W., Huang S. & Shoham D. (2023). linear_project. GitHub. https://github.com/WilliamFreedman/linear_project

Gupta, M., & Garg, A. K. (2012). Analysis of image compression algorithm using DCT. *International Journal of Engineering Research and Applications (IJERA)*, *2*(1), 515-521.

Harrington, P. (2012, April 9). *Chapter 14. simplifying data with the singular value decomposition · machine learning in action*. · Machine Learning in Action. https://livebook.manning.com/book/machine-learning-in-action/chapter-14/6

Kahu, S., & Rahate, R. (2013). Image compression using singular value decomposition. *International Journal of Advancements in Research & Technology*, *2*(8), 244-248.

Khobragade, P. B., & Thakare, S. S. (2014). Image compression techniques-a review. *Int. J. Comput. Sci. Inf. Technol*, *5*(1), 272-275.

Nicholson, K. W. (2019). Linear Algebra with Applications. *Lyryx*.

Pech-Pacheco, J. L., Cristóbal, G., Chamorro-Martinez, J., & Fernández-Valdivia, J. (2000, September). Diatom autofocusing in brightfield microscopy: a comparative study. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000* (Vol. 3, pp. 314-317). IEEE.

Pham, K. (2017). Discrete Cosine Transform in Image Compression. *Github Notes*. https://vkhoi.github.io/notes/discrete-cosine-transform-in-image-compression

Pu, I. M. (2006). Image Compression. *Fundamentals of Data Compression*. https://www.sciencedirect.com/topics/computer-science/image-compression

Rosebrock, A. (2015). Blur Detection with OpenCV. *pyimagesearch. https://pyimagesearch.com/2015/09/07/blur-detection-with-opencv/*

Watson, A. B. (1994). Image compression using the discrete cosine transform. *NASA Ames Research Center*.
http://sites.apam.columbia.edu/courses/ap1601y/Watson_MathJour_94.pdf