Numerical Methods Project 3

Moshe Wieder, Levi Langer, and Adam Dennis

April 22

Abstract:

This project explores Singular Value Decomposition (SVD) as a technique for image compression, particularly for color images. We manually compute SVD using eigen decomposition and apply it separately to each color channel (RGB). The compression efficiency is evaluated based on storage reduction and reconstruction error. Experimental results demonstrate that reducing the number of singular values significantly reduces storage while maintaining visual quality.

Introduction:

Image compression is pivotal in digital storage and transmission. The SVD provides an effective way to compress images by approximating their matrix representation. This project involves implementing SVD manually, applying it to color images, and analyzing the impact of varying singular values on compression efficiency and image quality.

<u>Body of the Problem and the Sections:</u>

<u>Explanation of SVD:</u>

SVD is a matrix factorization technique that decomposes an $m \times n$ matrix, A, into three special matrices: $A = USV^T$. U is an $m \times m$ orthogonal matrix whose columns are the left singular vectors of A. S is an $m \times n$ diagonal matrix containing the singular values (non-negative and in descending order). V^T is an $n \times n$ orthogonal matrix whose rows are the right singular vectors of A. The singular values in S represent the importance of different components in the transformation described by A.

SVD works by first computing the eigenvalues and eigenvectors of A^T*A (to obtain V) and $A*A^T$ (to obtain U), with the singular values being the square roots of the eigenvalues of A^T*A.

<u>How to use SVD to compress images (both grayscale and color):</u>

SVD can be used to compress images by approximating the original image with a lower-rank representation. First, an image is represented as a matrix, where each entry corresponds to the pixel intensity values. For a grayscale image, this is a single matrix, and for a colored image (RGB), it consists of three separate matrices—one for each color channel (red, green, blue). The next step is to apply SVD to the image matrix, decomposing it into three matrices:

$$A = U\ S\ V^T$$

where U is an orthogonal matrix containing the left singular vectors, S is a diagonal matrix containing the singular values, and V^T is an orthogonal matrix containing the right singular vectors. The singular values in S represent the importance of different components of the image.

For compression, we don't need to retain all the singular values. Instead, we select only the top "p" singular values, which capture the most important features of the image. The image is then approximated using the reduced rank representation,

$$A_p = U_p S_p V_p^T$$

where U_p , S_p , and V_p^T retain only the first p singular values and their corresponding vectors. This lower-rank approximation reduces the amount of data needed for image representation, leading to compression. The smaller the p, the more the image is compressed, but this also results in some loss of quality. However, even with significant compression, the reconstructed image often appears very similar to the original, as the most important features are preserved.

How to code this (without directly using the linear algebra library SVD command):

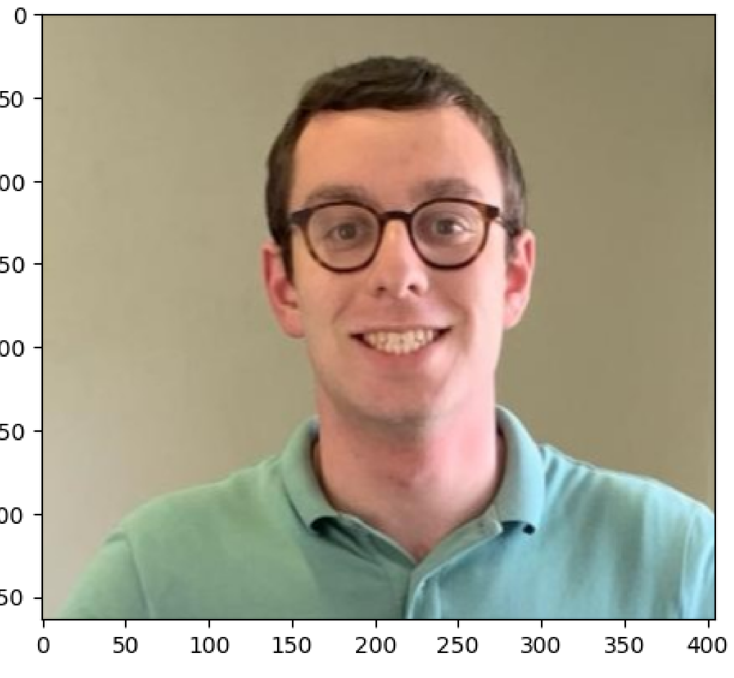
To code color image compression using SVD without relying on the built-in SVD function, we first separate the image into its individual color channels (RGB). Each channel is treated as a matrix, and the steps for SVD are performed on each of these matrices alone. First, for each color channel matrix, we compute the matrix products A^T*A and $A*A^T$, where A represents the color channel matrix. Then, we calculate the eigenvalues and eigenvectors of these matrices. The eigenvectors of A^T*A form the columns of V, and the eigenvectors of $A*A^T$ form the columns of U. The singular values are the square roots of the eigenvalues of A^T*A. Next, we construct the singular value matrix S, placing the singular values placed along the diagonal, and making the remaining positions zeros. To compress, we select the top p singular values and their corresponding eigenvectors, and reconstruct an approximated version of the color channel using $A_p = U_p S_p V_p^T$. Once each color channel is compressed, we combine the three channels back into a single image.

Computer Experiments/Simulations and Results

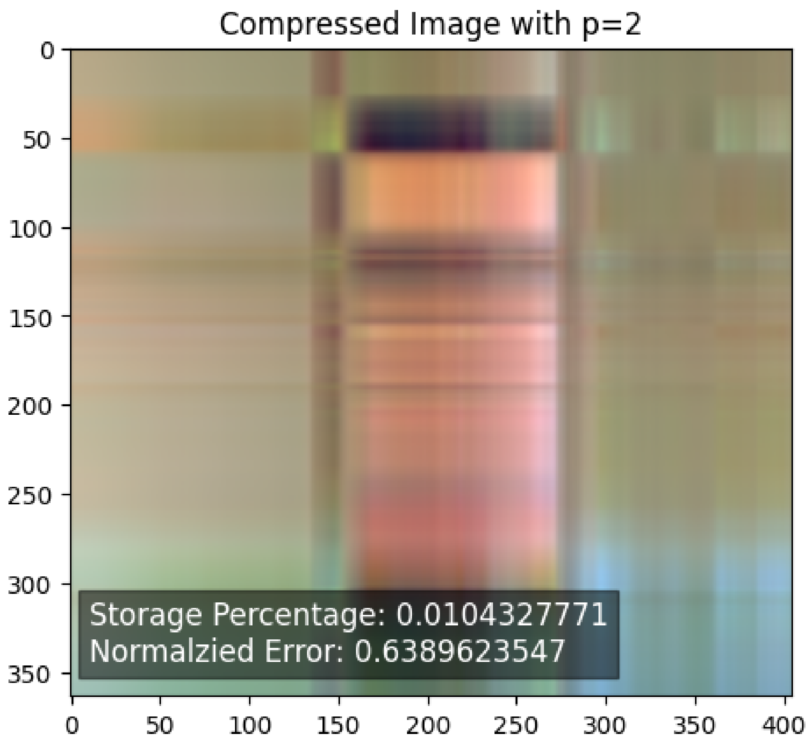
We ran tests for varying p values to find the compressed images, their storage percentages, and their normalized errors. The p values ranged from 2 to 77 at intervals of 15. The storage percentage was calculated by dividing the compressed image's storage size by the original image's storage size. The error was calculated by finding the difference of the compressed image matrix and the original matrix, and finding the norm of the resulting matrix. This was then normalized by dividing the error by the maximum possible error for the given matrix's size.

Below is displayed the original image along with selected compressed images that we calculated.

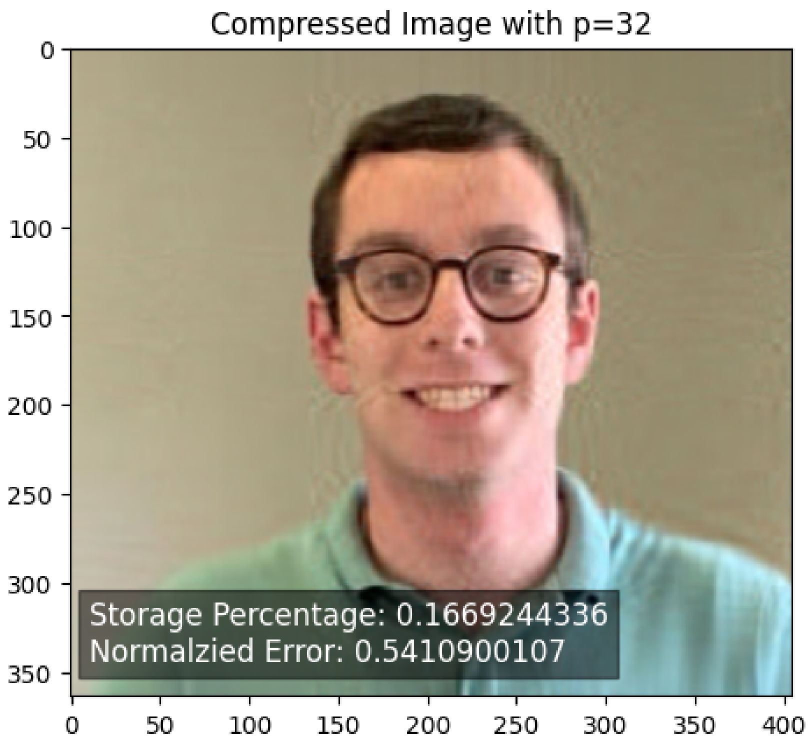
Original:



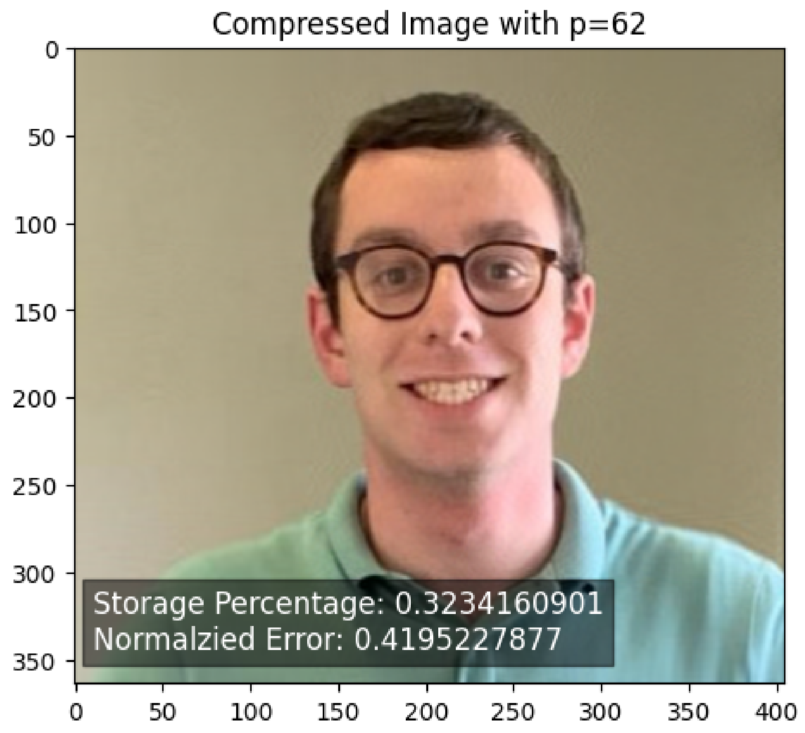
p=2


Compressed Image with p=2
Storage Percentage: 0.0104327771
Normalzied Error: 0.6389623547
0
50
100
150
200
250
300
350
400

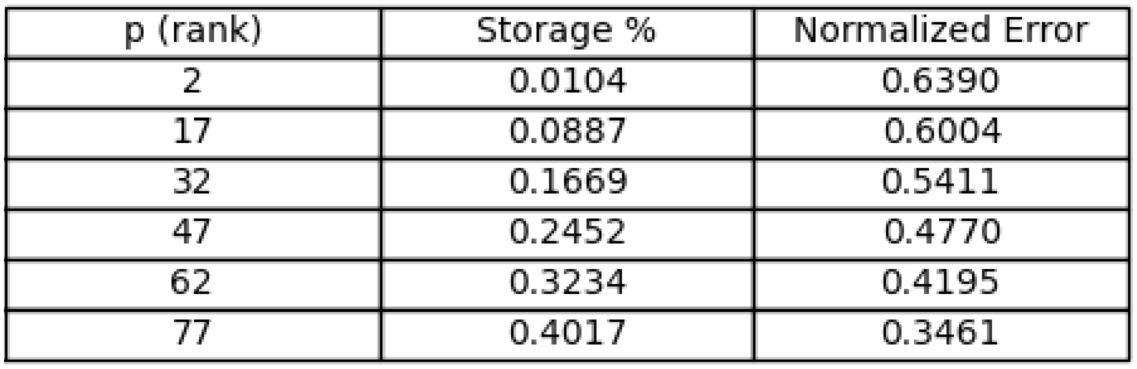
p=32


Compressed Image with p=32
Storage Percentage: 0.1669244336
Normalzied Error: 0.5410900107
0
50
100
150
200
250
300
350
400

p=62


Compressed Image with p=62
Storage Percentage: 0.3234160901
Normalzied Error: 0.4195227877

Below is a table summarizing the storage percentages and normalized errors for all tested p values.



p (rank)	Storage %	Normalized Error
2	0.0104	0.6390
17	0.0887	0.6004
32	0.1669	0.5411
47	0.2452	0.4770
62	0.3234	0.4195
77	0.4017	0.3461

Conclusions

The proportion of storage required increases linearly with p according to the following equation:

$s(p) = 0.005216p - 2.167 * 10\hat{}(-7)$

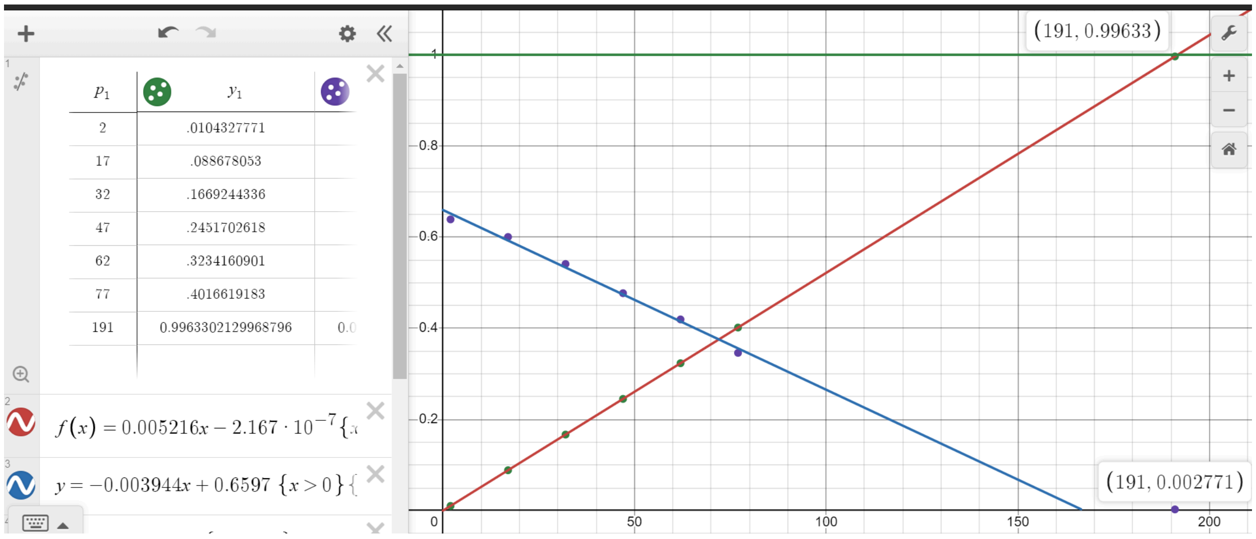
The standardized error decreases roughly linearly with p according to the following expression:

$e(p) \approx - 0.003944p + 0.6597$

Thus, the error and storage proportion are inversely correlated approximately linearly, according to the following expression:

$s(e(p)) \approx - 1.32251521298e + 0.872463069303$, though this is not accurate at the extreme, where e ~ 0

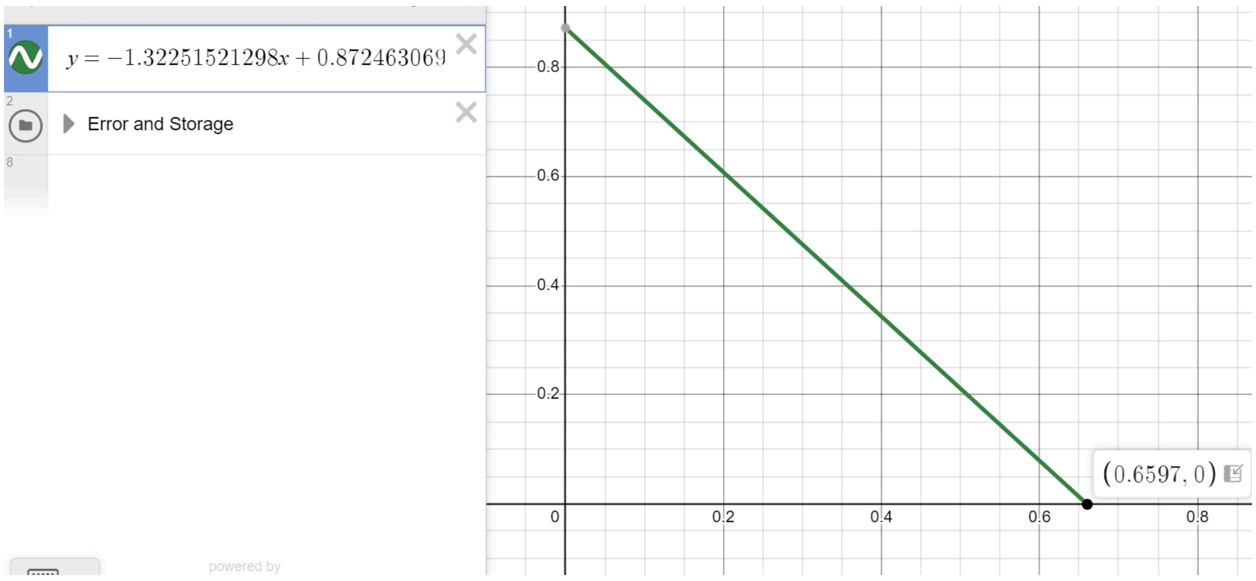
At p = 192, s > 1, so the greatest p that makes sense to use is 191, where s = 0.9963302129968796 and e = 0.002770954370177777. This is the minimal standardized error that will exist whenever we compress this image usefully.



p_1	● y_1	●
2	.0104327771	
17	.088678053	
32	.1669244336	
47	.2451702618	
62	.3234160901	
77	.4016619183	
191	0.9963302129968796	0.0

$f(x) = 0.005216x - 2.167 \cdot 10^{-7}\{x$

$y = -0.003944x + 0.6597 \ \{x > 0\}\{$



The above graph shows s(e). The projected maximum e, .6597, approximates the value of e when p = 1, which is 0.6580972165775043. Indeed, s at p = 1 is close to 0; it is 0.005216388549721883.

Visually, p = 62, shown earlier, creates an image that looks pretty similar to the original image, saving more than two-thirds of the storage space (s = .3234160901). We recommend using p = 62.

This project successfully demonstrated SVD-based image compression with manual SVD computation and color image processing. The results show that reducing the number of singular values significantly reduces storage while maintaining reasonable image quality. Future work could explore adaptive singular value selection for optimal compression.

References cited

"NumPy Reference — NumPy V1.23 Manual." *Numpy.org*, numpy.org/doc/stable/reference/index.html#reference.

Sauer, Timothy. *Numerical Analysis: Pearson New International Edition*. Second ed., Harlow, Pearson Education Limited, 2014, pp. 557–562, Section 12.4 - Applications of the SVD.

"Vector and Matrix Norms - MATLAB Norm." *Www.mathworks.com*, www.mathworks.com/help/matlab/ref/norm.html.

Appendix:

Link to the code:

Project3-code.pynb

Link to Desmos:

https://www.desmos.com/calculator/7ifcq3weor