BLG202E Numerical Methods in Comp. Eng. Spring 2025- Term Project

[*]Due: May 20, 2025

1st Onat Barış Ercan
Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Istanbul, Turkey
ercano21@itu.edu.tr

I. Introduction

Image compression is essential for reducing storage and transmission costs without significantly degrading visual quality. One effective approach is low-rank matrix approximation, with Singular Value Decomposition (SVD) being a widely used method.

Truncated SVD (TSVD) approximates a matrix using only its dominant singular components, preserving key features while reducing data size. In this project, TSVD is implemented from scratch without relying on high-level numerical libraries. Instead, core numerical techniques like power iteration and orthogonalization are used.

RGB images are compressed by processing each channel separately. The performance is evaluated using approximation error and storage cost to explore the trade-off between compression and reconstruction quality.

II. Methodology

The Truncated Singular Value Decomposition (TSVD) technique provides a compact approximation of a given matrix by preserving only its most dominant singular components. For a real matrix $A \in \mathbb{R}^{m\times n}$, the full SVD decomposes it as:

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

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices containing the left and right singular vectors, and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix holding the singular values in descending order.

The Truncated SVD of rank r uses only the top r singular values and their corresponding vectors:

$$A_r = U_r \Sigma_r V_r^T$$

To implement this decomposition from scratch, we avoid using high-level SVD or eigendecomposition functions. Instead, we approximate the top r eigenvalues and eigenvectors of the symmetric matrix $A^T A$ using *power iteration*. This iterative method converges to the dominant eigenvector by repeatedly applying the matrix and normalizing the result.

To obtain multiple eigenvectors, *Gram-Schmidt orthogonalization* is applied at each step to ensure orthogonality with previously found vectors. Singular values are then derived from the square roots of the corresponding eigenvalues, and left singular vectors are computed as:

$$u_i = \frac{1}{\sigma_i} A v_i$$

Since the images are in RGB format, each color channel is treated as a separate matrix and processed individually using the above method. The final compressed image is reconstructed by stacking the approximated R, G, and B channels.

This manual implementation provides both theoretical insights and practical control over the behavior of TSVD-based compression.

III. Implementation

The entire TSVD-based compression system was implemented in Python using only low-level numerical capabilities provided by NumPy. High-level decomposition routines such as ‘numpy.linalg.svd‘, ‘numpy.linalg.eig‘, or any function from ‘scipy.linalg‘ were strictly avoided.

The core of the implementation consists of the following modular functions:

- `power_iteration(A, num_iter)`: Iteratively approximates the dominant eigenvalue and eigenvector of a symmetric matrix A. Convergence is controlled via a small tolerance threshold.
- `compute_top_k_eigen(A, k)`: Uses repeated power iteration with Gram-Schmidt orthogonalization to extract the top k eigenpairs of $A^T A$.
- `tsvd(A, r)`: Constructs a rank-r approximation of matrix A using the previously computed singular values and vectors.

$$u_i = \frac{1}{\sigma_i} A v_i$$

and forms the reconstructed matrix as:

$$A_r = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

The RGB image is first loaded using the Pillow library and converted into three separate matrices representing the red, green, and blue channels. Each channel is processed independently using the ‘tsvd()‘ function, and the compressed channels are merged using ‘numpy.stack()‘ to produce the final reconstructed image.

All approximations are clipped to the valid [0, 255] pixel range and cast to ‘uint8‘ for display and visualization using ‘matplotlib‘. The entire compression process is executed inside Jupyter Notebook, which allows interactive analysis and plotting of the reconstruction results, approximation errors, and storage cost.

IV. Experiments and Results

The effectiveness of Truncated SVD in image compression was evaluated on two RGB images. Each image was processed by separating the R, G, and B channels and applying TSVD with various ranks $r \in \{2, 4, 8, 16, 32, 64\}$. All reconstructed images were visually compared and quantitatively assessed using error metrics and storage calculations.

A. Original Images

Figure 1 shows the original RGB images used in this project prior to any compression.

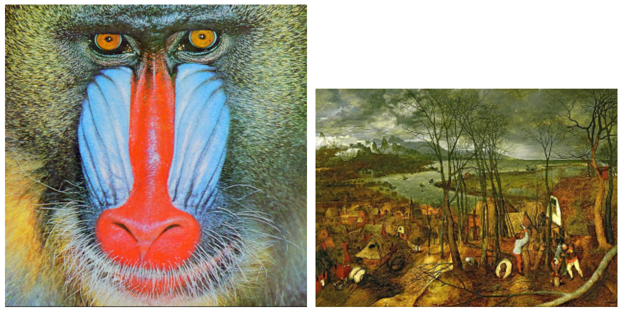

Fig. 1. Original images: Mandrill (left) and Brueghel (right).

B. Reconstructed Images

The visual results indicate that even with very low ranks such as $r = 8$, the overall structure and dominant colors of the images are preserved. As r increases, fine details and edges become clearer, and the image becomes visually indistinguishable from the original around $r = 64$.

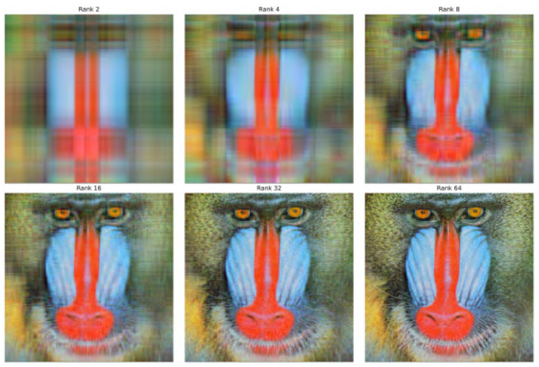

Fig. 2. Compressed reconstructions for Mandrill image at different rank values.

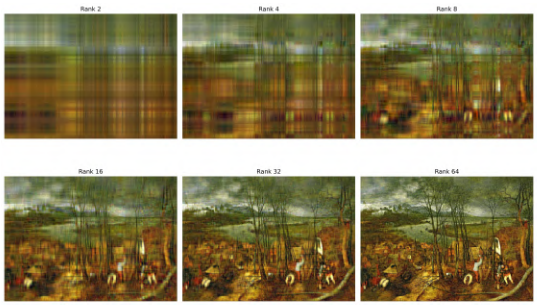

Fig. 3. Compressed reconstructions for Brueghel image at different rank values.

C. Absolute Error

The total absolute error was computed as the sum of the absolute differences between the original and reconstructed pixel values across all RGB channels:

$$\text{Absolute Error} = \sum_{i,j,c} |A_{i,j,c} - A_r i, j, c|$$

This error metric decreases significantly as the rank increases. The curve drops steeply for small ranks, especially up to $r = 16$, and then flattens out. This suggests that a significant portion of the image information is captured within a small number of singular components.

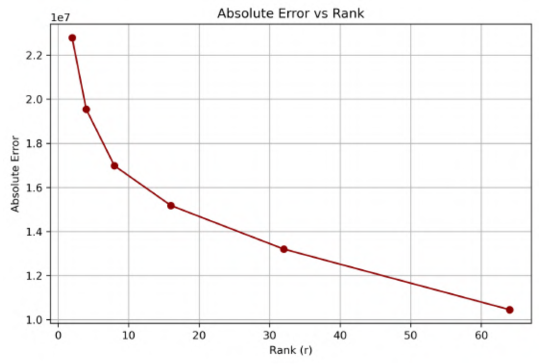

Fig. 4. Absolute error for Mandrill image at different rank values.

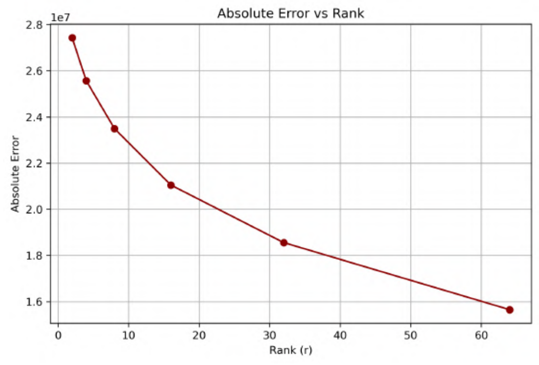

Fig. 5. Absolute error for Brueghel image at different rank values.

D. Storage Cost

To quantify the efficiency of compression, the storage cost was calculated for each rank using:

$$\text{Cost}(r) = 3 \cdot r \cdot (m + n + 1)$$

This accounts for storing r left and right singular vectors and singular values for each RGB channel. The cost increases linearly with rank, emphasizing the trade-off between compression and image fidelity.

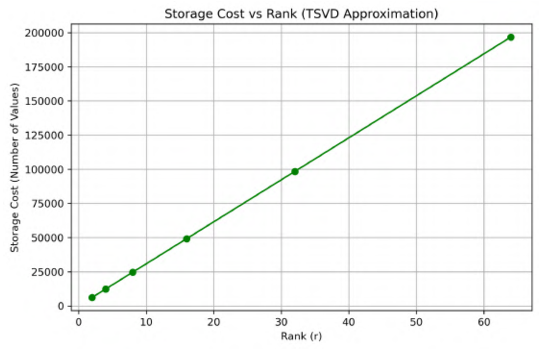

Fig. 6. Number of values stored for each rank r in TSVD. Storage increases linearly with rank. (Mandrill)

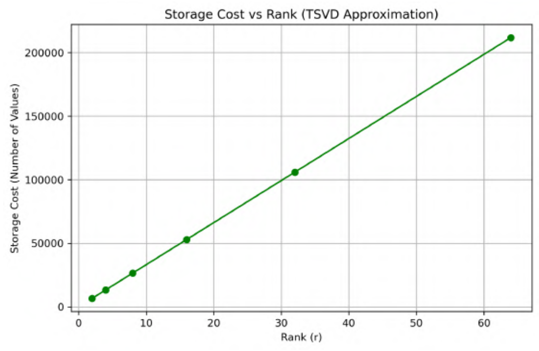

Fig. 7. Number of values stored for each rank r in TSVD. Storage increases linearly with rank. (Brueghel)

V. Discussion

While working on this project, I observed that Truncated SVD can be highly effective for compressing RGB images, even when implemented manually without built-in functions. By applying TSVD to each color channel independently, I was able to evaluate how well different rank values preserved the structure and color of the original image.

The absolute error results showed a steep decline for low ranks such as $r = 8$ or $r = 16$, after which the improvements became more gradual. This behavior suggests that the dominant singular components of the image carry most of the essential information, while higher-rank components mainly refine details.

The storage cost analysis confirmed that memory usage increases linearly with the rank. This highlighted the clear trade-off between compression ratio and image quality. Based on both visual inspection and error values, ranks around 16 to 32 appeared to offer the best balance between efficiency and accuracy.

From an implementation perspective, I encountered some sensitivity in the convergence behavior of power iteration, especially for higher ranks. Ensuring numerical stability and proper normalization was essential to maintain consistency in the results. These challenges provided practical insight into how numerical methods behave under real data and iterative constraints.

Overall, this project helped me gain a deeper understanding of matrix approximation and its practical application to image compression. The results were consistent with the theoretical expectations and confirmed the value of low-rank techniques for representing complex data with reduced storage.

VI. Conclusion

In this project, I implemented the Truncated Singular Value Decomposition (TSVD) method from scratch and applied it to image compression tasks. Without relying on high-level numerical libraries, I constructed the entire decomposition using power iteration for eigenvalue estimation and Gram-Schmidt orthogonalization to ensure vector independence.

By applying the TSVD to the R, G, and B channels of each image separately, I was able to reconstruct compressed versions of the original images for various rank values. Based on the results, I observed that even low-rank approximations (such as r = 8 or r = 16) captured the essential structure and color of the image quite well. The absolute error dropped significantly with increasing rank, which aligned with theoretical expectations from the course.

From a storage perspective, the compression was clearly effective for small r values, but the cost grew linearly as more singular components were retained. This highlighted the practical trade-off between image quality and memory efficiency—a concept frequently discussed in numerical methods when considering approximation and error.

Overall, this project helped me reinforce key concepts such as matrix decomposition, iterative eigenvalue computation, and error analysis. More importantly, it gave me hands-on experience in balancing computational accuracy and resource efficiency using linear algebra tools.

References

[1] H. R. Swathi, S. C. Sharma, and K. S. Jagadeesha, "Image compression using singular value decomposition," *IOP Conference Series: Materials Science and Engineering*, vol. 263, no. 4, p. 042082, 2017.

[2] T. Baumann, "SVD Image Compression Demo," Available: https://timbaumann.info/svd-image-compression-demo/ [Accessed: May 20, 2025].

[3] 3Blue1Brown, "What is the singular value decomposition?" YouTube, Apr. 6, 2020. [Online Video]. Available: https://www.youtube.com/watch?v=H7qMMudo3e8 [Accessed: May 20, 2025].

[4] NumPy Developers, "NumPy: The fundamental package for scientific computing in Python," Available: https://numpy.org/

[5] Pillow Contributors, "Pillow (PIL Fork) Documentation," Available: https://pillow.readthedocs.io/

[6] Matplotlib Developers, "Matplotlib: Visualization with Python," Available: https://matplotlib.org/