Image Compression Using Truncated SVD in C

Namaswi Vajjala-EE25BTECH11060

November 9, 2025

1 Introduction

Digital grayscale images can be represented as matrices where each entry corresponds to a pixel intensity ranging from 0 (black) to 255 (white). Image compression is important to reduce storage and transmission costs while maintaining visual quality. One effective mathematical technique for compression is Singular Value Decomposition (SVD).

2 Summary of Strang's video

The Singular Value Decomposition (SVD) expresses any real matrix $A \in \mathbb{R}^{m\times n}$ 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 ($U^TU = I$, $V^TV = I$), and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix containing the non-negative singular values of A. This factorization generalizes the diagonalization of symmetric matrices and applies to all rectangular matrices.

Importance

SVD connects orthogonality, eigenvalues, and rank. It shows how A scales and rotates vectors in space, forming the basis for applications such as principal component analysis (PCA) and image compression.

Orthogonal Bases

SVD provides orthogonal bases for the column and row spaces of A. Each pair (u_i, v_i) satisfies

$$Av_i = \sigma_i u_i$$

showing that A maps each right singular vector v_i to a scaled left singular vector u_i. Thus,

$$U^TAV = \Sigma$$

diagonalizes A.

Eigenvalue Connection

The singular vectors arise from the symmetric matrices

$$A^T A v_i = \sigma_i^2 v_i, \quad AA^T u_i = \sigma_i^2 u_i$$

The eigenvectors of $A^T A$ form V, and the eigenvectors of AA^T form U. The singular values are $\sigma_i = \sqrt{\lambda_i}$, where λ_i are eigenvalues of $A^T A$.

Rank-One Case

For a rank-one matrix $A = \sigma_1 u_1 v_1^T$, only one singular value σ_1 is nonzero. The column space is spanned by u_1, the row space by v_1, and the null spaces are their orthogonal complements.

Summary

SVD provides orthonormal bases for the four fundamental subspaces and shows that A acts as a pure scaling in orthogonal directions. It is widely used for data compression, noise reduction, and computing pseudoinverses.

$$A = U\Sigma V^T, \quad A^\dagger = V\Sigma^\dagger U^T$$

Error (Frobenius norm)
The squared Frobenius norm of the approximation error admits a compact expression in terms of the discarded singular values:

$$\|A - A_k\|_F^2 = \sum_{i=k+1}^{r} \sigma_i^2. \tag{1}$$

Thus the Frobenius error itself is

$$\|A - A_k\|_F = \left(\sum_{i=k+1}^{r} \sigma_i^2 \right)^{1/2}. \tag{2}$$

In program, you can compute it either by reconstructing A_k and using the matrix Frobenius norm or directly summing the squares of singular values beyond k

3 Implemented Algorithm in C

1. Read the input image (grayscale) into a matrix $A \in \mathbb{R}^{m \times n}$.
2. Compute the SVD of A: $A = U\Sigma V^T$ (use a library or an SVD routine).
3. Choose a target rank k (or choose k so that the retained energy $\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{r} \sigma_i^2$ exceeds a threshold, e.g. 0.99).
4. Form $A_k = U_k \Sigma_k V_k^T$ and convert values back to the valid pixel range $[0, 255]$ (clamp and round as needed).
5. Save the reconstructed image and report the compression ratio and error metrics.

4 Algorithm Explanation (Mathematics + Pseudocode)

A grayscale image is represented as a matrix A of size $m \times n$, with m rows and n columns. The algorithm performs truncated SVD as follows:

1. **Read Image:** Load the PGM file into a dynamically allocated 2D array.
2. **Compute $A^T A$:** This smaller symmetric matrix simplifies singular value computation.
3. **Power Iteration:** Iteratively compute the dominant eigenvector of $A^T A$ until convergence.
4. **Deflation:** Subtract the contribution of the computed eigenvector to find the next largest eigenvector.
5. **Compute Left Singular Vectors:** $U_k = AV_k/\Sigma_k$.
6. **Reconstruct Image:** $A_k = U_k \Sigma_k V_k^T$, scale pixels to $[0, 255]$, and write to a PGM file.

Pseudocode:

```
Input: Image matrix A, number of singular values k
Output: Reconstructed image A_k

1. Read A from PGM
2. Compute ATA = transpose(A) * A
3. For i = 1 to k:
      a. Compute largest eigenvector v_i using power iteration
      b. Compute singular value _i = sqrt(_i)
      c. Deflate ATA: ATA = ATA - _i * v_i * v_iˆT
4. Compute U_k = A * V_k / _k
5. Reconstruct A_k = U_k * _k * V_kˆT
6. Scale pixels to [0,255]
7. Write A_k to PGM
```

5 Comparison of Different Approaches and Justification

Different approaches exist for SVD computation:

- **Full SVD using libraries:** Accurate and fast, but requires external libraries like lapack,etc..
- **Jacobi or QR-based SVD:** Computes all singular values, but complex to implement and memory-intensive.
- **Eigen-decomposition of $A^T A$ with power iteration (chosen):** Efficiently computes top k singular values, simple in C, reduces memory and computation, fully compliant with project constraints.

I chose the Power Iteration method because it's much simpler to implement in C and works efficiently for finding only the top few singular values needed for image compression. The Jacobi method, while more accurate for full decomposition, is computationally heavy and complex to code without libraries. Power Iteration gave a good balance between simplicity, speed, and accuracy for this project.

6 Reconstructed Images for Different k

The reconstructed images show the effect of different k values:

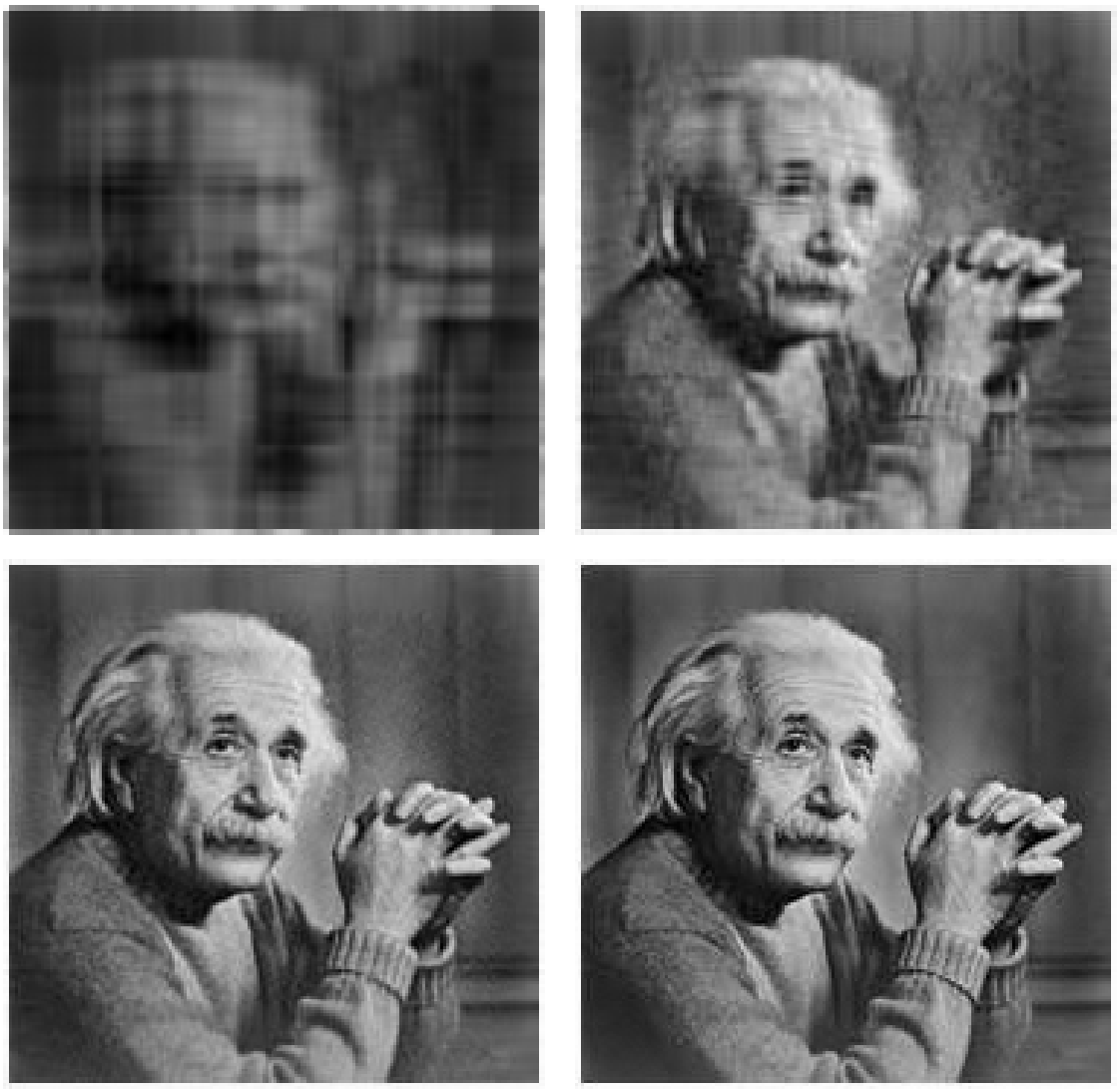

Figure 1: Einstein Image Reconstruction for Different k Values
k=5,20,50,100 respectively

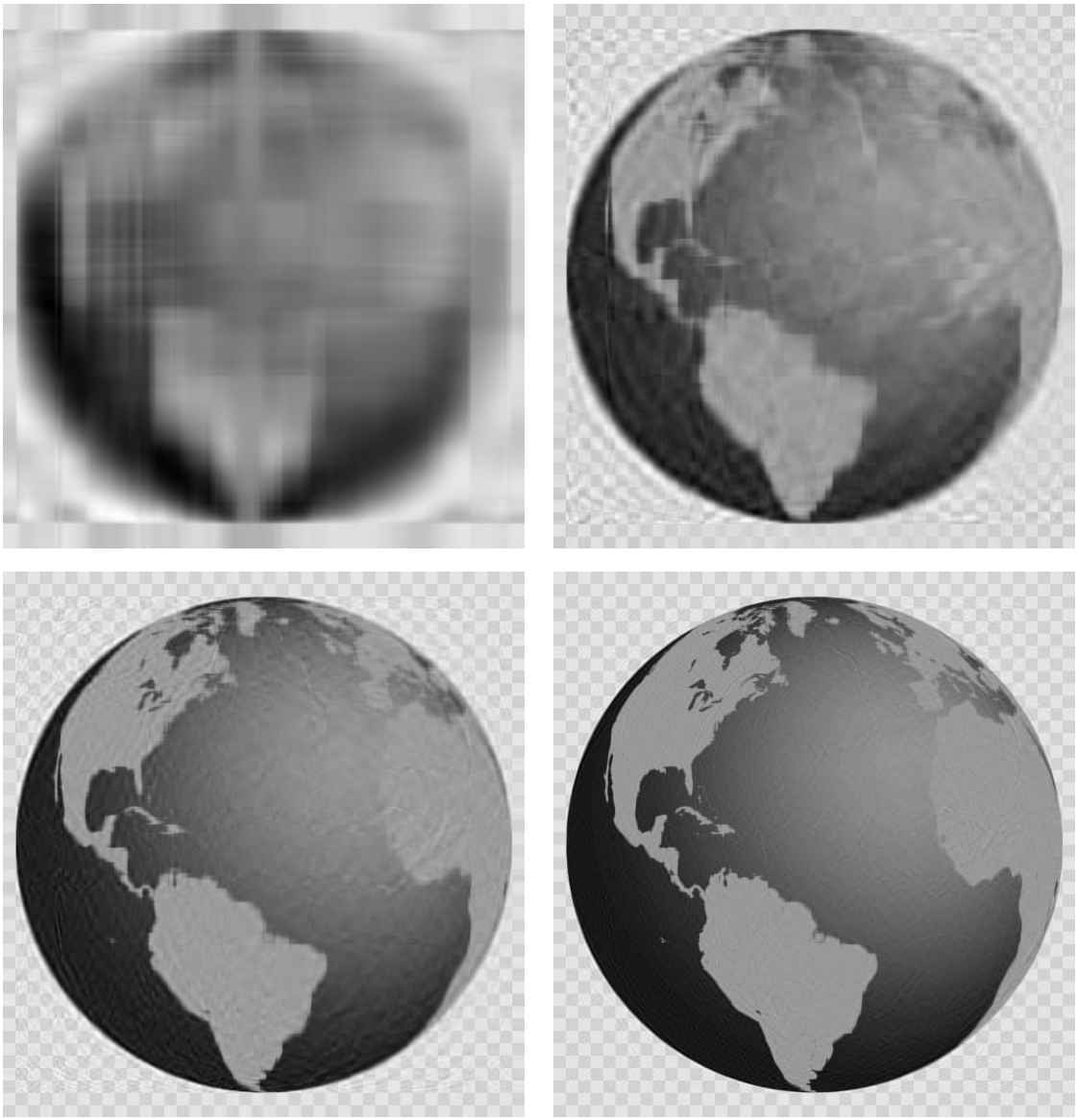

Figure 2: Globe Image Reconstruction for Different k Valuesk=5,20,50,100 respectively

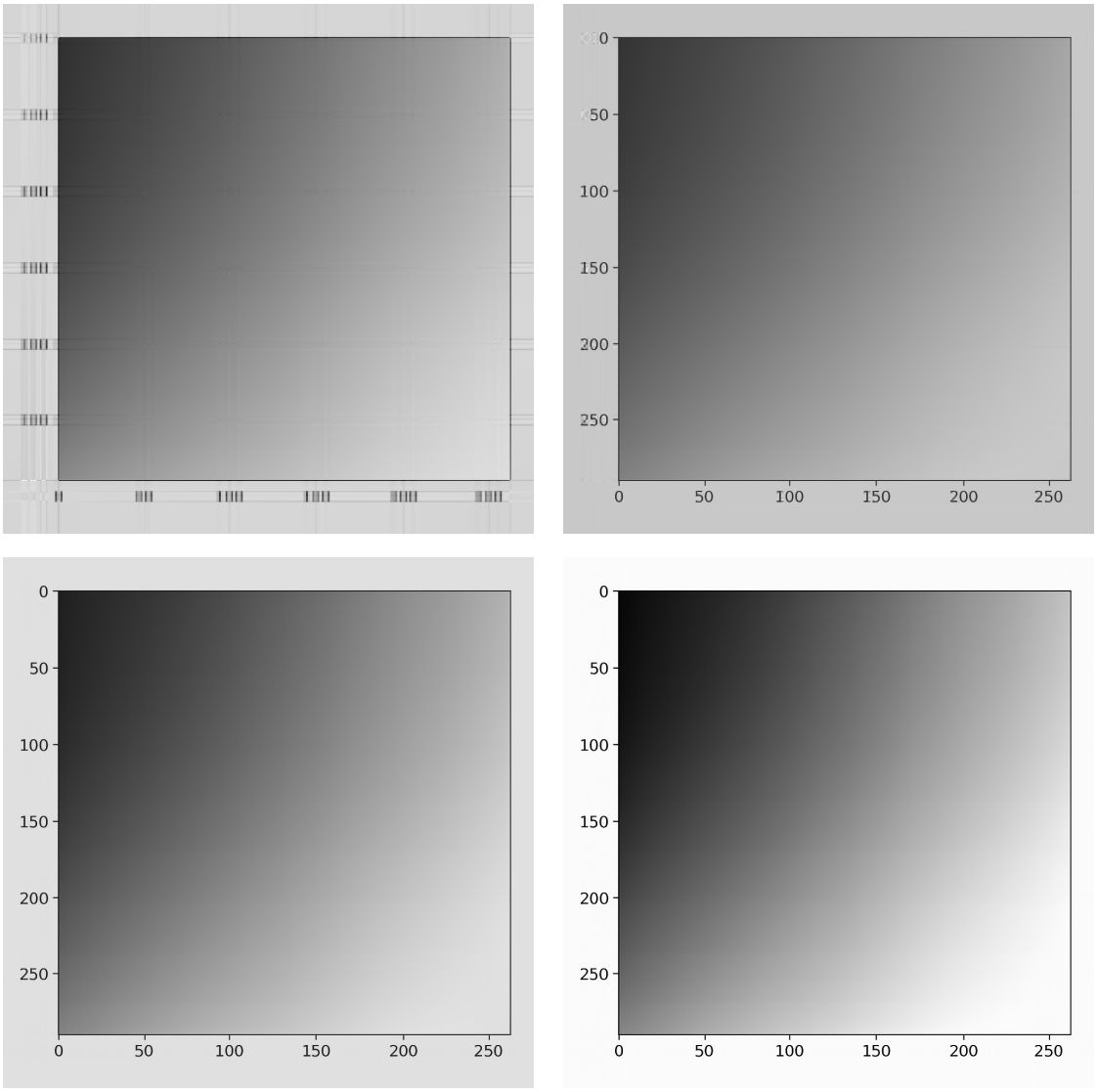

Figure 3: Greyscale Image Reconstruction for Different k Values
k=5,20,50,100 respectively

7 Error Analysis

The reconstruction error is measured using the Frobenius norm:

$$\|A - A_k\|_F = \sqrt{\sum_{i,j} (a_{ij} - a_{ij}^{(k)})^2}$$

As k increases, the error decreases since more singular values capture additional image information.

Loaded image: 1024x1024 → greyscale_ascii
For $k = 5$, Frobenius norm error = 32067.598046
For $k = 20$, Frobenius norm error = 38754.743439
For $k = 50$, Frobenius norm error = 21667.885490
For $k = 100$, Frobenius norm error = 2635.680646

Loaded image: 879x840 → globe_ascii
For $k = 5$, Frobenius norm error = 35260.732064
For $k = 20$, Frobenius norm error = 26212.870563
For $k = 50$, Frobenius norm error = 25355.144323
For $k = 100$, Frobenius norm error = 28012.020277

Loaded image: 182x186 → einstein_ascii
For $k = 5$, Frobenius norm error = 8702.980822
For $k = 20$, Frobenius norm error = 2303.426310
For $k = 50$, Frobenius norm error = 1022.630793
For $k = 100$, Frobenius norm error = 272.076524

8 Discussion of Trade-offs and Reflections

The choice of k controls the balance between compression and image quality:

- **Small k (5):** High compression, low memory usage, but blurry images.
- **Moderate k (20–50):** Balanced approach; most structures preserved with moderate compression.
- **Large k (100):** High fidelity; minimal error; reduced compression benefit.

Reflections:

- Truncated SVD effectively captures essential image features while reducing storage.
- Power iteration with deflation is a practical method for top-k singular values, fully implemented in C.
- Increasing k improves image quality but increases memory and computation cost.
- The project demonstrates both quantitative (Frobenius error, storage) and qualitative (visual quality) trade-offs.

9 Conclusion

This project successfully implements truncated SVD for grayscale image compression in C, demonstrating the relationship between singular values, reconstruction quality, and compression. Reconstructed images and error analysis illustrate the trade-offs clearly, showing how low-rank approximations can efficiently preserve essential image information while reducing storage requirements.