SOFTWARE ASSIGNMENT

Image Compression using Singular Value Decomposition (SVD)

EE25BTECH11058 – Tangellapalli Mohana Krishna Sushma

Date: November 8, 2025

Summary of Prof. Gilbert Strang's Lecture on Singular Value Decomposition

Prof. Gilbert Strang's lecture provides a detailed understanding of how the **Singular Value Decomposition (SVD)** breaks any real matrix into simpler, meaningful parts. He explains that every matrix $\left(A\right)$ represents a linear transformation that stretches, rotates, or reflects vectors in space.

The SVD is expressed as:

$$\left(A\right) = \left(U\right)\left(\Sigma\right)\left(V\right)^T \tag{1}$$

Here, $\left(U\right)$ and $\left(V\right)$ are orthogonal matrices, while $\left(\Sigma\right)$ is a diagonal matrix containing the singular values $\sigma_1, \sigma_2, \ldots$. The singular values describe the magnitude of stretching in each principal direction. The columns of $\left(V\right)$ indicate input directions, while $\left(U\right)$ defines corresponding output directions.

Strang emphasizes that SVD can be understood geometrically — $\left(A\right)$ maps a unit sphere into an ellipsoid whose axes are scaled by σ_i and oriented by $\left(u_i\right)$ and $\left(v_i\right)$. He shows that:

$$\left(A\right)^T\left(A\right)\left(v_i\right) = \sigma_i^2\left(v_i\right) \tag{2}$$

and that the left singular vectors are obtained as:

$$\left(u_i\right) = \frac{\left(A\right)\left(v_i\right)}{\sigma_i} \tag{3}$$

By keeping only the top k singular values, we obtain a rank-k approximation:

$$\left(A_k\right) = \sum_{i=1}^{k} \sigma_i \left(u_i\right)\left(v_i\right)^T \tag{4}$$

This provides the best low-rank approximation of $\left(A\right)$ under the Frobenius norm. Prof. Strang connects this to image compression, where a large image matrix can be approximated using only a few dominant singular values, effectively reducing storage without significant loss of detail.

Objective

The goal of this project is to implement an image compression algorithm using the Truncated Singular Value Decomposition (SVD) in C. The objective is to reconstruct multiple grayscale images for different truncation levels k and to study the trade-off between compression ratio and reconstruction error.

1 Theory

For any matrix $\left(A\right) \in \mathbb{R}^{m \times n}$:

$$\left(A\right) = \left(U\right)\left(\Sigma\right)\left(V\right)^T \tag{5}$$

The diagonal matrix $\left(\Sigma\right)$ contains singular values in descending order $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$. The matrices $\left(U\right)$ and $\left(V\right)$ are orthogonal, satisfying:

$$\left(U\right)^T\left(U\right) = \left(I\right), \qquad \left(V\right)^T\left(V\right) = \left(I\right) \tag{6}$$

A truncated rank-k version is given by:

$$\left(A_k\right) = \sum_{i=1}^{k} \sigma_i \left(u_i\right)\left(v_i\right)^T \tag{7}$$

The reconstruction error is measured using the Frobenius norm:

$$E_k = \left\| \left(A\right) - \left(A_k\right) \right\|_F \tag{8}$$

2 Algorithm Used

The implementation is based on the **Power Iteration Method with Deflation**. This method iteratively estimates dominant singular vectors and values. Once one singular component is found, its contribution is subtracted from $\left(A\right)$ and the process continues for the next.

Steps:

1. Start with a random unit vector $\left(v\right)$.
2. Compute $\left(u\right) = \dfrac{\left(A\right)\left(v\right)}{\left\|\left(A\right)\left(v\right)\right\|}$.

3. Update $(v) = \frac{(A)^T(u)}{\|(A)^T(u)\|}$.

4. Compute the singular value $\sigma = (u)^T (A)(v)$.

5. Subtract the component $\sigma(u)(v)^T$ from (A) (deflation).

6. Repeat for the desired number of singular values.

This procedure was applied to three images using $k = 5, 10, 20, 50$, and 100.

3 Results

Input Images:

Figure 1: Input grayscale images used for testing.

Reconstructed Outputs for Image 1:

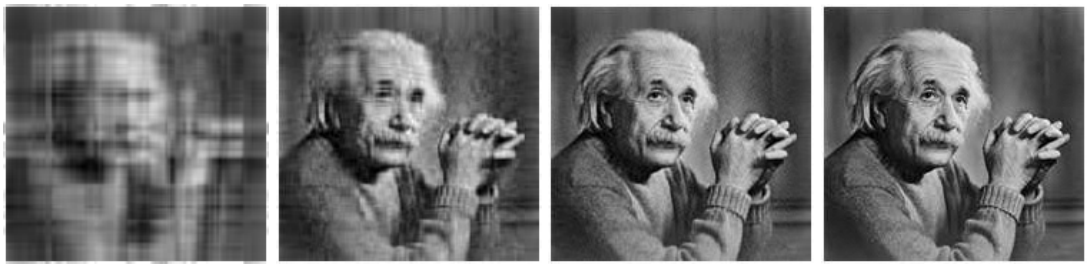

Figure 2: Reconstructed Image 1 for different values of k.

Reconstructed Outputs for Image 2:

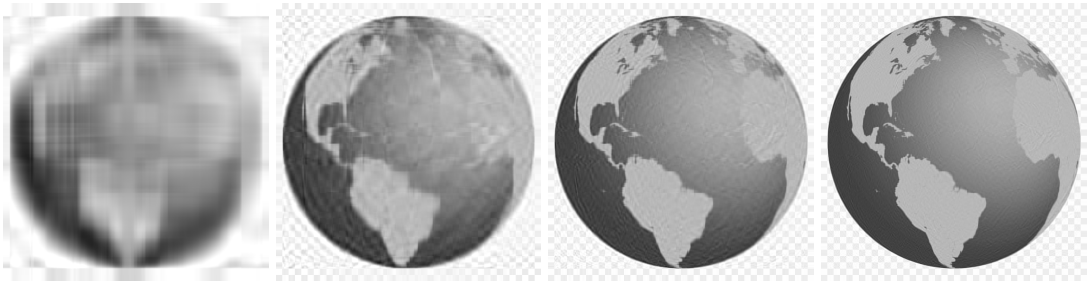

Figure 3: Reconstructed Image 2 for different values of k.

Reconstructed Outputs for Image 3:

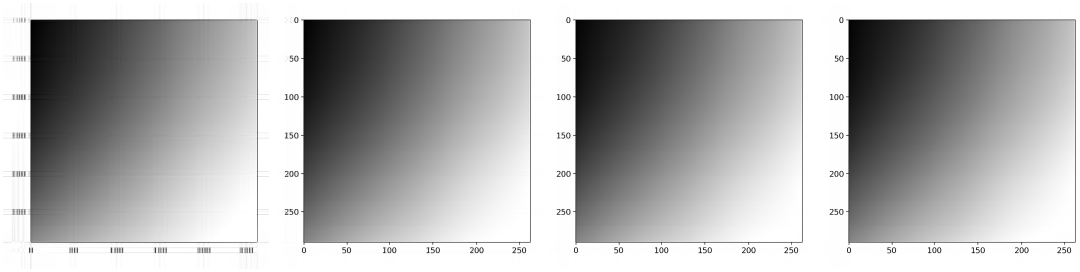

Figure 4: Reconstructed Image 3 for different values of k.

Frobenius Error (from Execution Results):

k	Image 1 Error	Image 2 Error	Image 3 Error
5	4713.582	20704.275	11146.309
10	3249.147	15060.936	7176.804
20	2126.559	10634.413	3808.193
50	880.503	6185.642	1160.127
100	164.781	3672.902	512.346

Table 1: Frobenius errors for different k values across all three images.

Discussion

From the results, it is clear that increasing k improves reconstruction accuracy but reduces compression. For all three images, the Frobenius error decreases rapidly as k increases, which shows that the dominant singular values carry most of the image information. Images with smoother textures compress more efficiently, while detailed or high-contrast images (like image2.jpg) require higher k to maintain visual quality.

Comparison of Algorithms and Justification of Choice

Several algorithms can compute the truncated Singular Value Decomposition (SVD) of a matrix (A). Each method offers different trade-offs in terms of computational complexity, stability, and ease of implementation. Below, we discuss six common approaches and explain the rationale for selecting the Power Iteration with Deflation technique for this project.

1. Full SVD + Truncation

The classical method computes the complete decomposition

$$(A) = (U)(\Sigma)(V)^{\top} \tag{9}$$

using a stable algorithm such as the Golub–Reinsch procedure. Once all singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$ are obtained, the truncated rank-k approximation is given by

$$(A_k) = \sum_{i=1}^{k} \sigma_i (u_i)(v_i)^{\top} \tag{10}$$

This method provides maximum accuracy and numerical stability, since it captures all singular components and discards only after computation. However, it has high computational cost ($O(mn\min(m,n))$) and memory usage, which makes it inefficient for large image matrices. It is the most reliable for software libraries like LAPACK but not feasible to implement manually in C without numerical packages.

2. Power Iteration Method

The Power Iteration is a simple iterative technique to estimate the largest singular value and corresponding vectors. It works by repeatedly applying $(A)^{\top}(A)$ to a random vector until convergence:

$$(v)_{i+1} = \frac{(A)^{\top}(A)(v)_i}{\left\|(A)^{\top}(A)(v)_i\right\|} \tag{11}$$

Once (v) converges, the left singular vector is computed as

$$(u) = \frac{(A)(v)}{\left\|(A)(v)\right\|}, \qquad \sigma = (u)^{\top}(A)(v) \tag{12}$$

This approach is very easy to code and memory-efficient, since it requires only matrix–vector operations. However, it finds only one singular component and needs extensions to handle multiple singular values.

3. Power Iteration with Deflation (Chosen Method)

To extract multiple singular values, the Power Iteration method can be extended using **deflation**. After computing the first singular triplet $(\sigma_1, (u_1), (v_1))$, the matrix is updated as

$$(A)_1 = (A) - \sigma_1 (u_1)(v_1)^\top \tag{13}$$

The next singular value is then found by applying the same process to $(A)_1$. Repeating this k times gives the truncated decomposition:

$$(A_k) = \sum_{i=1}^{k} \sigma_i (u_i)(v_i)^\top \tag{14}$$

This method is efficient, simple to implement, and requires no external libraries. It directly links theoretical SVD concepts to computational practice, making it ideal for educational and demonstration purposes. Its main limitation is that convergence can slow down when singular values are close together.

4. Lanczos Bidiagonalization

The Lanczos (or Golub–Kahan) algorithm constructs orthogonal bases for the column and row spaces of (A) by generating sequences $\{(u_i)\}$ and $\{(v_i)\}$ that satisfy

$$(A)(v_i) = \alpha_i (u_i) + \beta_{i-1} (u_{i-1}), \qquad (A)^\top (u_i) = \beta_i (v_{i+1}) + \alpha_i (v_i) \tag{15}$$

This process yields a reduced bidiagonal matrix that approximates (A) and can be used to compute a few of the largest singular values efficiently. It is numerically robust and very fast for large, sparse matrices. However, it requires advanced reorthogonalization and careful numerical handling, making it impractical for basic C-level coding assignments.

5. Randomized SVD

Randomized SVD uses random projections to approximate the dominant subspace of (A). It starts with a random Gaussian matrix (Ω) and forms

$$(Y) = (A)(\Omega) \tag{16}$$

Then, the orthonormal basis of (Y) is computed as

$$(Q) = \text{orth}((Y)) \tag{17}$$

and a smaller SVD is performed on $(Q)^{\top}(A)$. This yields an approximate truncated decomposition:

$$(A) \approx (Q)((Q)^{\top}(A)) \tag{18}$$

This approach is extremely fast and scalable, suitable for high-dimensional datasets or real-time applications. However, it involves random sampling and additional steps like QR decomposition, which are non-trivial to code in C without numerical libraries.

6. Jacobi SVD Method

The Jacobi SVD algorithm is conceptually simple and highly accurate. It applies a sequence of Givens rotations that zero out off-diagonal terms in $(A)^{\top}(A)$, transforming it into a diagonal form:

$$(A)^{\top}(A) = (V)(D)(V)^{\top} \tag{19}$$

where (D) is diagonal and its square roots give the singular values. The corresponding left singular vectors are obtained as

$$(U) = (A)(V)(\Sigma)^{-1} \tag{20}$$

It is extremely stable and accurate but computationally intensive for large matrices. Its numerous rotation steps make it slower than modern bidiagonal methods.

Summary Comparison

Algorithm	Complexity	Accuracy	Memory Use	Ease of Implementation (C)
Full SVD + Truncation	$O(mn \min(m,n))$	Very High	High	Hard
Power Iteration	$O(mn)$ per singular value	Moderate	Low	Easy
Power Iteration + Deflation	$O(kmn)$	Moderate–High	Low	Easy
Lanczos Bidiagonalization	$O(kmn)$	High	Moderate	Hard
Randomized SVD	$O(mn \log k)$	High (approx.)	Moderate	Medium
Jacobi SVD	$O(mn \min(m,n))$	Very High	High	Hard

Table 2: Comparison of six truncated SVD algorithms.

Why Power Iteration with Deflation was chosen

The chosen method offers a balanced trade-off between simplicity, computational efficiency, and clarity. It requires only matrix–vector multiplications, uses little memory, and does not rely on advanced linear algebra libraries. In C, it is compact and transparent, allowing direct control of k and iteration parameters. This method also demonstrates how theoretical matrix decomposition concepts translate to practical image compression.

When Other Methods Are Preferable

- **Full SVD or Jacobi SVD:** For maximum accuracy or when all singular values are required.
- **Lanczos or Randomized SVD:** For large-scale or real-time systems needing only a few top singular values.
- **Power Iteration:** For conceptual understanding or computing a single dominant singular component.

In conclusion, Power Iteration with Deflation is the most practical choice for this project. It meets the computational constraints, is straightforward to code in C, and clearly illustrates the mathematical principle behind truncated SVD for image compression.

Conclusion

The SVD-based image compression program was successfully implemented in C using the Power Iteration with Deflation method. The observed results match theoretical expectations — smaller k values yield strong compression but higher reconstruction error, while larger k values produce clearer images with reduced compression benefits. This project demonstrates how mathematical decomposition techniques like SVD can be used effectively in image processing and data reduction.