Software Assignment Report

Image Compression Using Truncated SVD

Name: Dhanush Kumar A

Roll Number: AI25BTECH11010

Department of Artificial Intelligence

Indian Institute of Technology Hyderabad

November 8, 2025

Contents

I. Summary of Strang's Video

Professor Gilbert Strang introduces the **Singular Value Decomposition (SVD)**. He explains that any real matrix A can be written as

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

where U and V are orthogonal matrices, and Σ is a diagonal matrix with nonnegative entries. Strang shows that the vectors in V (the right singular vectors) are the eigenvectors of the symmetric matrix A^TA:

$$A^TAv_i = \sigma_i^2 v_i.$$

Hence, the eigenvalues of A^TA are the squares of the singular values of A. He also explains that the columns of U (the left singular vectors) form an orthonormal basis for the output space, while the columns of V form an orthonormal basis for the input space.

This decomposition helps us understand how a matrix transforms vectors: it first rotates them using V, then stretches them by Σ, and finally rotates again using U. Strang emphasizes that the SVD works for any real (even rectangular or non-symmetric) matrix and reveals its essential structure. Through examples and visual explanations, he clarifies how singular values describe the strength of each independent direction in the transformation. Overall, the lecture gives both an algebraic and geometric understanding of the SVD and its importance in mathematics and applications.

II. Explanation of the Implemented Algorithm

The algorithm applies Truncated SVD for image compression. Given a grayscale image matrix $A \in \mathbb{R}^{m\times n}$, we compute:

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

and keep only the top k singular values.

A. Mathematical Steps

Initialize reconstructed matrix:

$$A_k := 0_{m\times n}$$

For $i = 1, 2, \ldots, k$:

a) Initialize right singular vector $v_i^{(0)} \in \mathbb{R}^n$ randomly and normalize:

$$v_i^{(0)} := \frac{v_i^{(0)}}{\|v_i^{(0)}\|_2}$$

b) Power iteration: for $t = 0, 1, \ldots, T$

$$y_i^{(t)} := Av_i^{(t)}, \quad z_i^{(t)} := A^T y_i^{(t)}, \quad v_i^{(t+1)} := \frac{z_i^{(t)}}{\|z_i^{(t)}\|_2}$$

Stop if $\|v_i^{(t+1)} - v_i^{(t)}\|_2 < \varepsilon$

c) Compute singular value and left singular vector:

$$\sigma_i := \|Av_i^{(t+1)}\|_2, \quad u_i := \frac{Av_i^{(t+1)}}{\sigma_i}$$

d) Update reconstructed matrix:

$$A_k := A_k + \sigma_i u_i v_i^T$$

e) Deflate original matrix:

$$A := A - \sigma_i u_i v_i^T$$

Return reconstructed matrix: A_k

B. Pseudocode

```
Input:
    A           -> m x n matrix
    k           -> number of top singular values to compute
    max_iter    -> maximum iterations
    tolerance   -> convergence threshold

Output:
    A_k         -> rank-k approximation of A

Algorithm:

1. Initialize A_k <- zero matrix of size m x n

2. For i = 1 to k do:
       a. Initialize a random vector v of size n
          Normalize: v <- v / ||v||

       b. Repeat until convergence or max_iter:
              y <- A * v
```

```
            z <- A^T * y
            v_new <- z / ||z||
            if ||v_new - v|| < tolerance then
                  break
            end if
            v <- v_new
        end repeat

    c. Compute singular value: sigma <- ||A * v||
       Compute left singular vector: u <- (A * v) / sigma

    d. Update reconstructed matrix:
            A_k <- A_k + sigma * u * v^T

    e. Deflate original matrix:
            A <- A - sigma * u * v^T

3. Return A_k
```

III. Comparison with Other Algorithms

A. Why Power Iteration with Deflation Was Chosen

- **Focus on top-k singular values:**
  In image compression and many applications, we only need the largest k singular values. Power iteration efficiently finds the dominant singular vectors one by one, without computing the full SVD.
- **Simplicity and clarity:**
  The algorithm is straightforward to implement in C/Python hybrid code and easy to understand for educational and report purposes.
- **Memory efficiency:**
  Works on large matrices without storing full decompositions. Each step only requires vectors u and v, not the full U or V matrices initially.
- **Deflation allows sequential computation:**
  Once the top singular triplet is computed, deflation removes its contribution to find the next dominant singular value. This aligns perfectly with truncated SVD objectives.
- **Reasonable performance for moderate k:**
  For applications like image compression, usually $k \ll n$, so power iteration is faster than full SVD.

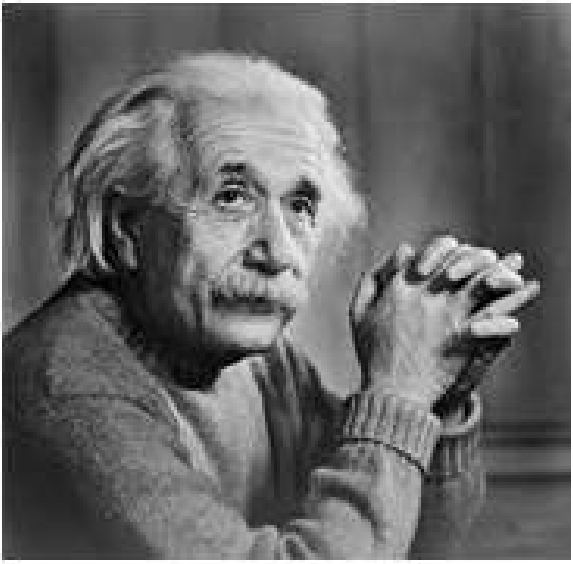
B. Why Other Algorithms Were Not Used

- **Full SVD (Golub–Reinsch method):**
  Computes all singular values and vectors.
  Reason not used: Too computationally expensive for large matrices ($O(mn^2)$ if $m \geq n$). For image compression, we only need the top-k singular values.

- **Lanczos / Arnoldi (Krylov subspace methods):**
  Iterative methods to approximate several singular values at once, especially for large sparse matrices.
  Reason not used: More complex to implement than Power Iteration; requires careful handling for numerical stability; overkill for dense matrices of moderate size.
- **Randomized SVD:**
  Uses random projections to approximate top-k singular values.
  Reason not used: Produces approximate results which may reduce compression quality; requires additional parameter tuning; less suitable for an educational implementation.
- **Jacobi SVD:**
  Very accurate and stable, but computationally expensive for large matrices. Computes all singular values while we only need the top-k.
  Reason not used: More complex to implement than Power Iteration.

IV. Reconstructed Images for Different k

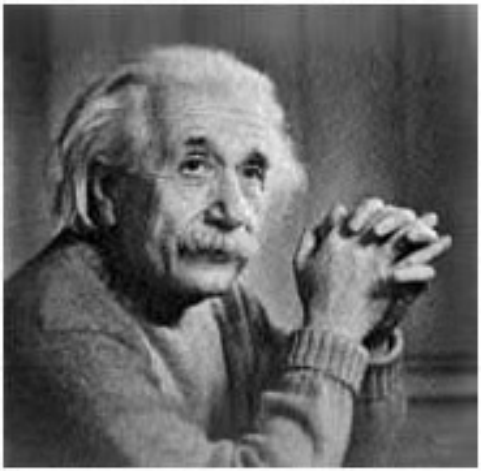
A. einstein



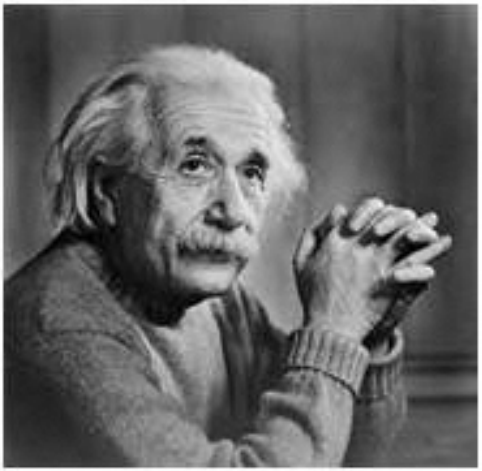
Original Image

Reconstructed Image k=5

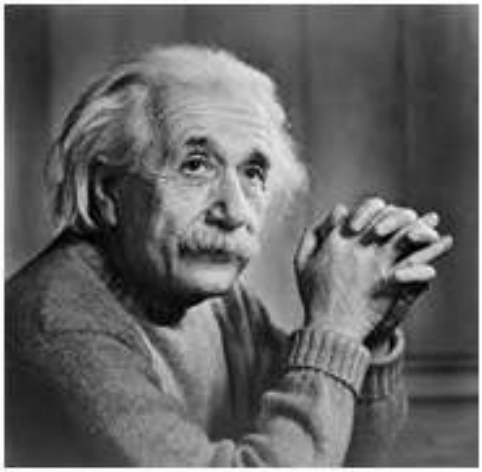
Reconstructed Image k=20



Reconstructed Image k=50



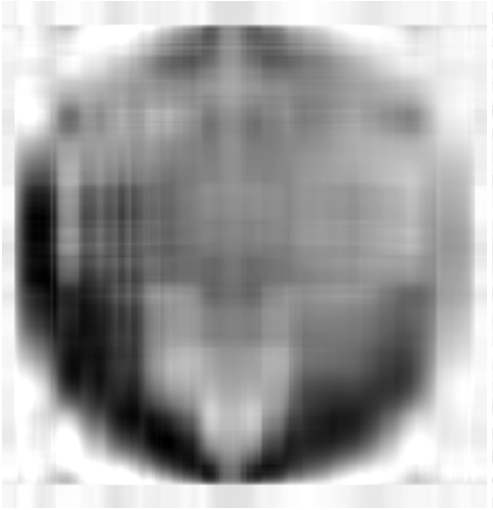
Reconstructed Image k=100



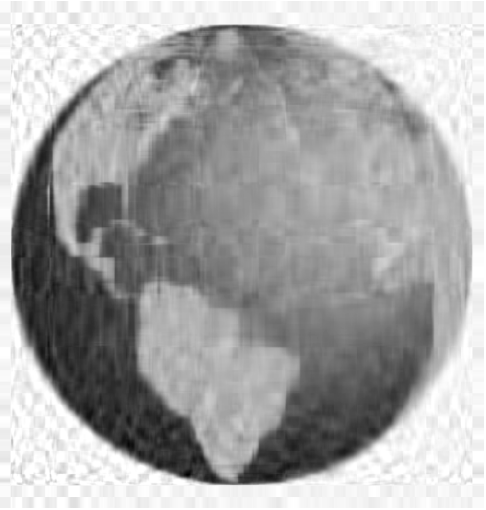
Reconstructed Image k=200

B. globe

Original Image



Reconstructed Image k=5



Reconstructed Image k=20

Reconstructed Image k=50

Reconstructed Image k=100

Fig. 5: Reconstructed Image k=200

C. greyscale

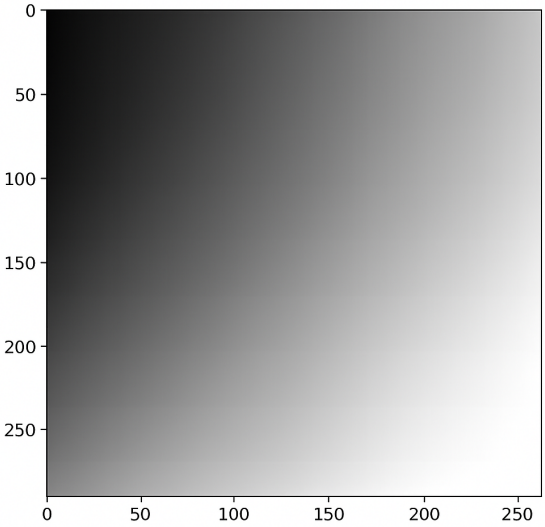
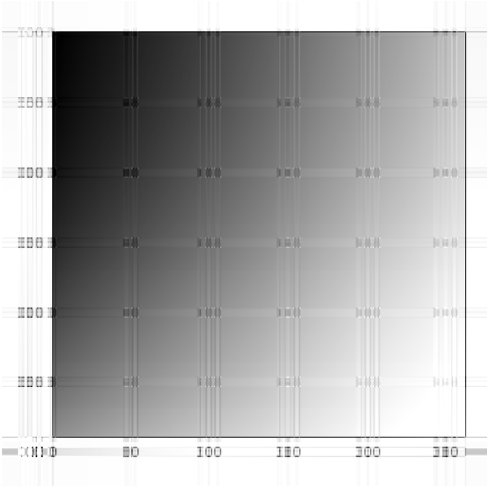
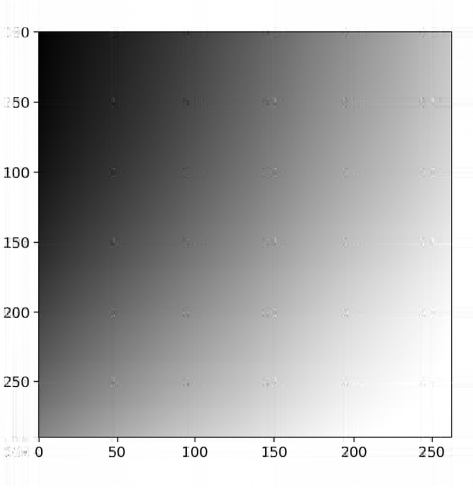


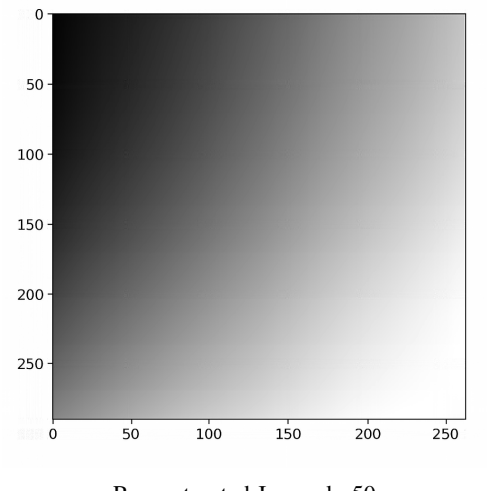
Fig. 6: Original Image



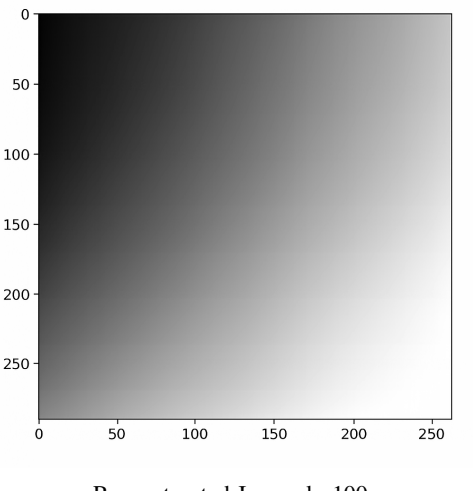
Reconstructed Image k=5



Reconstructed Image k=20



Reconstructed Image k=50



Reconstructed Image k=100

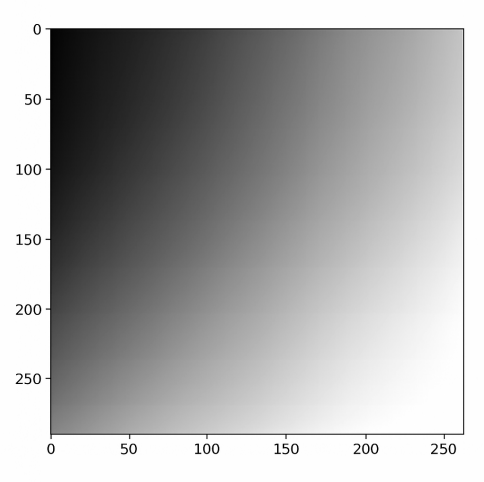


Fig. 9: Reconstructed Image k=200

V. Error Analysis

This section presents the Frobenius norm error $\|A - A_k\|_F$ between the original image A and its rank-k approximation A_k obtained from the SVD-based compression. The error measures how well the compressed image retains the original information. As k increases, more singular values are preserved, improving accuracy but reducing compression efficiency.

A. einstein

Rank (k)	$\|A - A_k\|_F$
5	5154.28
20	2422.96
50	1052.34
100	202.69
200	0.00

einstein

For the Einstein image, the approximation error decreases sharply as k increases. At low ranks ($k = 5$ or 20), significant detail is lost, but after $k = 100$, the image closely matches the original. This shows that the dominant singular values capture most of the important image features.

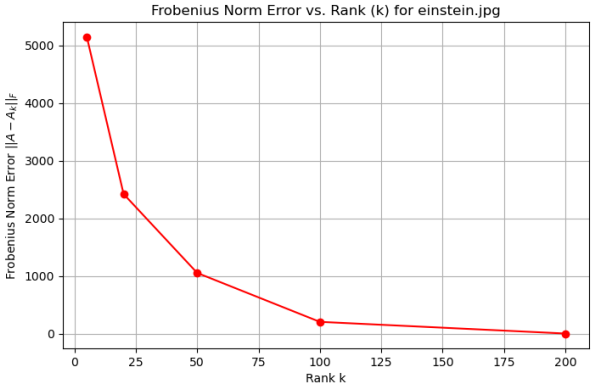


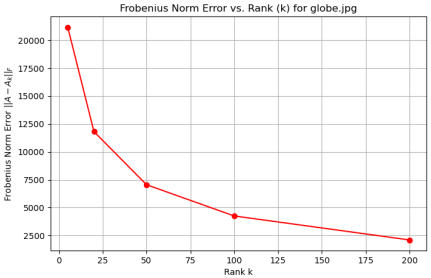
Fig. 10: Plot of k vs $\|A - A_k\|_F$ for einstein

B. globe

Rank (k)	$\|A - A_k\|_F$
5	21183.47
20	11809.66
50	7052.31
100	4253.83
200	2105.24

globe

The globe image shows a gradual error reduction with increasing k. Since it contains smoother regions and less sharp contrast, it achieves good visual quality even for $k = 50$. Beyond $k = 100$, the improvement becomes less noticeable, indicating optimal compression around this range.



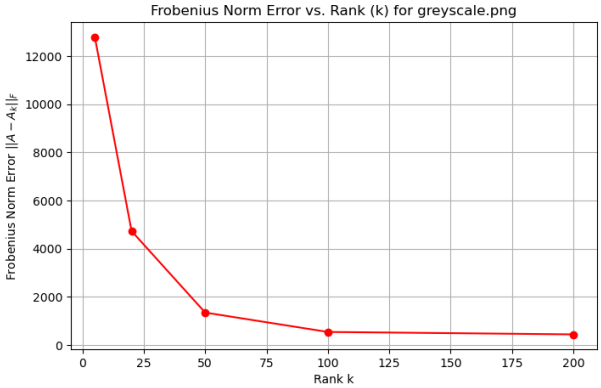
Plot of k vs $\|A - A_k\|_F$ for globe

C. greyscale

Rank (k)	$\|A - A_k\|_F$
5	12795.19
20	4734.36
50	1352.27
100	548.03
200	447.12

greyscale

For the greyscale image, the Frobenius error drops rapidly up to $k = 50$, after which the curve flattens. This indicates that most structural information is captured by the first few singular values, making it highly compressible without much visual loss.



Plot of k vs $\|A - A_k\|_F$ for greyscale

Overall, the Power Iteration with Deflation approach efficiently reconstructs images with low error using only a small number of singular values.

VI. Discussion of Trade-offs and Reflections

The implementation of image compression using Truncated SVD through Power Iteration with Deflation involves several trade-offs between computational efficiency, accuracy, and implementation simplicity.

A. Trade-offs

- **Accuracy vs. Compression:** Increasing the rank k improves image quality but reduces compression efficiency. For small k, the compressed image loses fine details, while for large k, the compression ratio decreases as storage requirements increase.

- **Speed vs. Precision:** Power Iteration provides an efficient way to estimate dominant singular values without performing a full SVD. However, it may take more iterations for convergence when singular values are close to each other, slightly affecting runtime.
- **Simplicity vs. Numerical Stability:** The algorithm is simple to implement and integrates well with C–Python hybrid execution. Yet, compared to methods like Lanczos or Jacobi SVD, it is less numerically stable for extremely large or ill-conditioned matrices.
- **Sequential Computation:** The deflation step allows computation of each singular triplet sequentially, which is ideal for moderate image sizes. However, it can accumulate small rounding errors over many iterations.

B. Reflections on Implementation Choice

The Power Iteration with Deflation method was chosen primarily for its balance between simplicity and effectiveness. It avoids the heavy computational load of full SVD while still achieving high-quality image reconstruction for small to medium values of k. The hybrid implementation in Python and C allowed efficient matrix operations while maintaining clarity and modularity in the code.

From experimentation, it was observed that:

- Most image features are captured within the first few singular values.
- Beyond $k = 100$, the error improvement becomes marginal.
- The approach provides a clear demonstration of how SVD separates structure and detail in an image.

Overall, the method successfully demonstrates the concept of low-rank approximation for image compression. It balances performance and interpretability, making it a suitable algorithmic choice for both educational and practical contexts.