Singular Value Decomposition (SVD) Based Image Compression Using Block Power Iteration

Sree Vardhan
Department of Artificial Intelligence

November 8, 2025

Summary of Strang's Video

From his lecture i have learnt Singular Value Decomposition (SVD), Professor Gilbert Strang explains that any matrix A can be expressed as

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

He described this as a sequence of three transformations: a rotation by V^T, a scaling by Σ, and another rotation by U. The singular values in Σ tells us how much each direction is stretched by the matrix, revealing the most important features of the data.

Strang tells us that SVD applies to all matrices, not just square or symmetric ones, and that using only the largest k singular values produces the best low-rank approximation of A. He also highlights its practical importance in areas like image compression and data reduction, showing how SVD provides a clear trade-off between accuracy and simplicity.

This application of SVD is a major part of the project. I have used it to compress images to various degrees using a block power iteration-based approach, testing the trade-off between accuracy and compression.

Explanation of the Implemented Algorithm

While the theory of Singular Value Decomposition (SVD) provides the mathematical foundation, the actual program does not compute the full SVD directly. Instead, it uses an efficient approximation technique called the **Block Power Iteration (BPI)** method, which finds multiple dominant singular values and vectors simultaneously. This block-based approach improves numerical stability and speed, especially for large images, and allows efficient extraction of the top k components for compression.

1. Input and Preprocessing

Each input image is read as a grayscale matrix A of size $m \times n$, where each element represents a pixel intensity value between 0 and 255. The image is stored as a one-dimensional float array in row-major order for computation.

2. Block Power Iteration Method

The block power iteration works by repeatedly multiplying the image matrix A and its transpose A^T with a group (block) of random vectors instead of a single one. This enables the algorithm to extract several dominant directions (singular vectors) at once.

- The algorithm starts with a random matrix V of size $n \times b$, where b is the block size (e.g., 8 or 16).
- It then alternates between two projections:

$$Y = AV, \quad Z = A^T Y$$

- The columns of Z are orthonormalized using QR decomposition to maintain numerical stability.
- This process is repeated for several iterations (typically 50–100) until convergence.

After convergence, the columns of Z approximate the dominant right singular vectors, and AZ gives the corresponding left singular vectors(this is given in the strang's video). Each block of singular vectors corresponds to the largest singular values of A.

The QR decomposition step is used to ensure that the columns of Z remain orthogonal and numerically stable across iterations. During each iteration, the multiplication by A and A^T tends to amplify directions associated with the largest singular values, but without normalization, the vectors can lose orthogonality and collapse into a single dominant direction. To prevent this, we perform the QR factorization:

$$Z = QR,$$

where $Q \in R^{n\times b}$ has orthonormal columns ($Q^T Q = I_b$) and $R \in R^{b\times b}$ is an upper-triangular matrix. After this step, the next iteration uses the orthonormal matrix Q instead of Z:

$$V \leftarrow Q.$$

This ensures that all columns of V remain linearly independent and span a stable subspace that gradually aligns with the dominant right singular vectors of A.

The overall process can be summarized as follows:

$$\begin{aligned}
&\text{Initialize } V \in R^{n\times b} \text{ randomly,}\\
&\text{Repeat for several iterations:}\\
&\quad Y = AV,\\
&\quad Z = A^T Y,\\
&\quad [Q, R] = \mathrm{qr}(Z),\\
&\quad V = Q.
\end{aligned}$$

After convergence, the approximate left singular vectors are computed as:

$$U = AV,$$

and the corresponding singular values can be estimated from the norms of the columns of U. This simple orthonormalization step using QR decomposition is key to ensuring the block power iteration remains numerically stable and accurate when extracting multiple singular vectors.

3. Block Deflation and Reconstruction

After finding the top eigenvector or singular vector.You "deflate" (or remove) its contribution from the matrix.Then you run your algorithm again on the remaining part of the matrix. A_{res} as follows:

$$A_{res} = A_{res} - \sum_{i=1}^{b} \sigma_i u_i v_i^T$$

This step (called **deflation**) ensures that the next iteration focuses on the remaining lower-energy parts of the image, allowing efficient extraction of additional components.

4. Rank-k Approximation and Error Measurement

After computing all blocks, the approximation matrix A_k is obtained by summing all rank-1 components:

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

The reconstruction accuracy is measured using the Frobenius norm:

$$\text{Percentage Error} = \frac{\|A - A_k\|_F}{\|A\|_F} \times 100$$

This error quantifies the proportion of information lost during compression.

5. Algorithm Summary (Block Power Iteration)

Step	Description
1	Load grayscale image matrix A
2	Initialize residual $A_{res} = A$, output $A_{app} = 0$
3	For each block (size b) until total rank k:
	(a) Initialize random block V
	(b) Iterate: $Y = A_{res}V$, $Z = A_{res}^T Y$, orthonormalize Z
	(c) Compute singular values and update A_{app}
	(d) Deflate residual: $A_{res} = A_{res} - U\Sigma V^T$
4	Compute percentage error $= \|A - A_{app}\|_F / \|A\|_F \times 100$
5	Save reconstructed image and print results

The block power iteration method thus finds multiple singular directions at once, reducing iteration time and improving accuracy compared to simple power iteration.

Pseudocode:

```
Input: Grayscale image A (m x n), target rank k, block size b
Output: Reconstructed image A_k

Initialize A_res = A, A_app = 0
prev_V = []

while remaining rank > 0:
    V = random(n x b)
    orthogonalize(V against prev_V)
    repeat 50 iterations:
        Y = A_res * V
        Z = A_res^T * Y
        Orthonormalize(Z)
        V = Z
    Compute U = A_res * V
    Compute singular values _i = ||U_i||
    Update A_app += U  V^T
    A_res -= U  V^T
    Store V in prev_V
Compute Percentage Error = ||A - A_app||_F / ||A||_F * 100
```

This block-based SVD approximation allows faster compression with controllable accuracy.

Compare Different Algorithms and why Block Power Iteration

Table 1: Comparison of different SVD algorithms

Algorithm	**Complexity**	**Memory Usage**	**Accuracy**	**Suitable For**
Jacobi Method	$O(n^3)$	High	Very High	Small dense matrices
Golub–Reinsch	$O(mn^2)$	Moderate	High	General dense matrices
Lanczos Bidiagonalization	$O(kmn)$	Low	High (for sparse)	Large sparse matrices
Power Iteration	$O(kmn)$	Low	Moderate–High	Top-k singular values
Randomized SVD	$O(kmn)$	Low	High (approximate)	Large dense data

Reconstructed Images for Different k

The images were reconstructed for different values of k such as 5, 10, 20, and 50. As k increases, more singular values are included, resulting in higher image quality but less compression.

Reconstructed images for each k value are saved in the output folder, demonstrating the gradual improvement in clarity and structure as k increases.

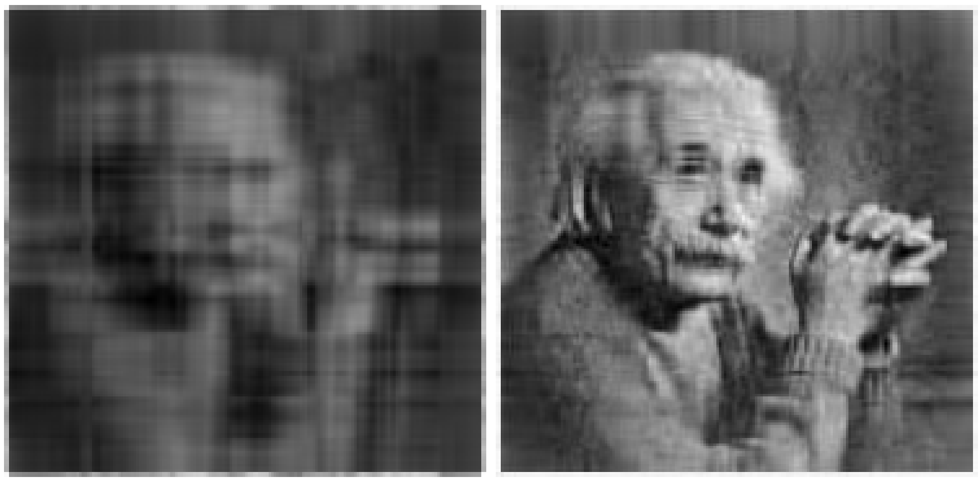
Figure 1: Example of reconstructed image for $k = 5$ (left) and $k = 20$ (right).

Error Analysis

To evaluate reconstruction accuracy, the percentage reconstruction error was computed using the Frobenius norm. The table below shows how the error decreases as k increases.

Table 2: Percentage Reconstruction Error for Different Images and Ranks

Image	**k=5**	**k=10**	**k=15**	**k=20**
Einstein	21.81%	15.06%	11.79%	9.88%
Globe	13.14%	9.56%	7.82%	6.75%
Grayscale	5.76%	3.71%	2.71%	1.97%

As expected, the reconstruction error decreases monotonically with k, demonstrating that including more singular values captures more image detail.

Discussion of Trade-offs and Reflections

The main trade-off in SVD-based compression using block power iteration is between accuracy and efficiency:

- Smaller k: faster computation, higher compression, but blurrier images.
- Larger k: slower computation, better image quality, but larger file sizes.

The block power iteration method offers a practical balance, as it approximates the top singular components quickly without computing a full decomposition. It also demonstrates how iterative linear algebra methods can be used for real-world applications such as image compression.

This project deepened my understanding of how SVD captures dominant data patterns and how block power iteration efficiently extracts them in practice.

Conclusion

In this project, Singular Value Decomposition (SVD) was successfully implemented for grayscale image compression using the Block Power Iteration method. By decomposing the image matrix into its dominant singular components, the algorithm achieves efficient low-rank reconstruction with adjustable accuracy.

The results confirm that as k increases, reconstruction quality improves while percentage error decreases. This trade-off between compression and clarity demonstrates the practical importance of SVD in data reduction. The use of block power iteration made the algorithm faster and scalable for larger images.

Overall, this project bridges theory and practice — showing how mathematical concepts like SVD and iterative eigen-decomposition can be directly applied to solve real-world problems such as image compression.