

# **Software Assignment Report**

## **Image Compression using SVD**

TAMMINEEDI RUSHIL SHANMUKHA SRINIVAS - EE25BTECH11057

November 8, 2025

# Introduction to SVD

In the field of Linear Algebra, **Singular Value Decomposition (SVD)** is one of the most powerful and widely used Matrix Factorization techniques. It plays a fundamental role in various areas such as data science, machine learning, image processing and signal analysis. SVD helps in simplifying complex matrices into simpler and meaningful components, making it easier to understand the structure of data and perform operations like compression, noise reduction and dimensionality reduction.

## What is SVD ?

**SVD** is a method of decomposing any real or complex matrix into the product of three special matrices. For a given Matrix **A** of size  $m \times n$ , **SVD** represents it as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top} \quad (1)$$

where **U** is an  $m \times m$  orthogonal matrix.

**$\Sigma$**  is an  $m \times n$  diagonal matrix whose diagonal elements are non-negative and are called **Singular Values**.

$\mathbf{V}^{\top}$  is the transpose of an  $n \times n$  orthogonal matrix **V**.

## Usage of SVD in image compression

**Singular Value Decomposition** is widely used in image compression to reduce the storage size of images without significantly affecting their visual quality. In SVD-based compression, an image matrix is decomposed into three matrices **U**,  **$\Sigma$**  and  $\mathbf{V}^{\top}$ . By keeping only the largest singular values in  **$\Sigma$**  and removing the smaller ones, we obtain a low-rank approximation of the original image. This greatly reduces the amount of data needed to store the image, while still preserving most of its important features and clarity. As a result, SVD provides an effective method for compressing images, maintaining a balance between reduced file size and acceptable image quality.

# Methods for Computing Eigenvalues and Eigenvectors

## 1. Power Method

The power method is an iterative technique for finding the dominant eigenvalue and its corresponding eigenvector of a matrix. It starts with an initial guess for the eigenvector and repeatedly applies the matrix to approximate the eigenvalue.

**Advantages:** - Simple and computationally inexpensive.

**Disadvantages:** - Only retrieves the largest eigenvalue. - Sensitive to matrix conditioning.

## 2. Inverse Power Method

The inverse power method extends the power method to compute the smallest eigenvalue by inverting the matrix before applying iterations. It effectively locates eigenvalues near a given shift.

**Advantages:** - Finds eigenvalues closest to a chosen value.

**Disadvantages:** - Requires matrix inversion, which is computationally expensive for large matrices.

## 3. Jacobi Method

The Jacobi method iteratively diagonalizes a symmetric matrix by rotating it to eliminate off-diagonal elements. It is suitable for dense matrices and provides all eigenvalues simultaneously.

**Advantages:** - Simple and well-suited for symmetric matrices.

**Disadvantages:** - Slow convergence for large matrices. - Inefficient for sparse systems.

## 4. QR Iteration Method

QR iteration is a powerful method for finding all eigenvalues of a matrix. It involves successive QR factorizations and matrix multiplications to reduce the matrix to a nearly diagonal form.

**Advantages:** - Robust and computes all eigenvalues. - Applicable to any matrix.

**Disadvantages:** - Computationally intensive for very large matrices.

## 5. Arnoldi Iteration

The Arnoldi method extends the power method for large sparse matrices by approximating eigenvalues using Krylov subspaces. It is commonly used in numerical simulations.

**Advantages:** - Efficient for sparse systems. - Finds multiple eigenvalues.

**Disadvantages:** - Complex implementation. - Requires good initial approximations.

## Power Iteration Method and its Advantages

The **Power Iteration method** is a simple and iterative numerical technique used to approximate the largest singular value and corresponding singular vectors of a matrix. Since Singular Value Decomposition (SVD) requires finding eigenvalues and eigenvectors of the matrices  $\mathbf{A}^\top \mathbf{A}$  or  $\mathbf{A}\mathbf{A}^\top$ , the Power Iteration method plays an important role in doing it.

### Advantages of Power Iteration Method:

- **Simple and Easy to Implement:** The method involves basic matrix-vector operations, making it straightforward for manual or coded implementation.
- **Efficient for Large Matrices:** Power Iteration is highly suitable when only the top singular values are needed, especially in large datasets or high-dimensional applications.
- **Memory Efficient:** Since it works with vectors rather than full matrix decompositions, it uses much less memory compared to traditional SVD algorithms.

As Image compression deals mostly with top singular values Power Iteration method is better when compared with other methods like QR Iteration, Jacobi Method and Inverse Power Method etc. When compared to Jacobi method, it is also faster for non-symmetric matrices. It is more precise and more flexible when compared with other methods. So it is a better method for **SVD** when compared with other methods.

## Why I Used the Power Iteration Method

The Power Iteration method was chosen due to its accuracy and versatility in computing eigenvalues and eigenvectors. Unlike other methods, Power Iteration Method works efficiently for both symmetric and non-symmetric matrices, making it a generalized approach. Furthermore, Power Iteration Method is simple to learn and use and also minimizes errors during computation. This method's ability to find eigenvalues and eigenvectors quickly and with a simple approach makes it the most suitable choice for this assignment.

## Summary of Strang's Video

According to **Singular Value Decomposition (SVD)** any rectangular matrix can be expressed as a product of 3 different matrices. If there exists a matrix  $\mathbf{A}$  with size  $m \times n$  then

$$\mathbf{A} = U\Sigma V^T \quad (2)$$

where

$U$  and  $V$  are orthogonal Matrices (columns are orthonormal).

Also  $U$  is an unitary matrix.

Columns of  $U$  are eigen vectors of  $\mathbf{A}\mathbf{A}^T$  and columns of  $V$  are eigen vectors of  $\mathbf{A}^T\mathbf{A}$ .

$\Sigma$  is a diagonal matrix with its diagonal entries as non-negative singular values.

Singular Values are the square roots of corresponding Eigen values.

## Math Logic behind SVD for Image Compression

**Image as a Matrix:** We need to convert a grayscale image to a matrix  $\mathbf{A}$  of size  $m \times n$

$$A[i, j] = 0.299R + 0.587G + 0.114B \quad (3)$$

This is the standard formula for converting RGB into grayscale.

**Singular Value Decomposition (SVD) :**

$$\mathbf{A} = U\Sigma V^T \quad (4)$$

**Low Rank Approximation :** Keep top  $K$  singular values and vectors.

$$A_K = \sum_{i=1}^K \sigma_i u_i v_i^T \quad (5)$$

**Power Iteration to find Singular Values:** Let  $\mathbf{A}^T\mathbf{A}$  have eigen values

$$\lambda_1 > \lambda_2 > \dots > \lambda_n \geq 0 \quad (6)$$

and the corresponding orthonormal eigenvectors are

$$e_1, e_2, \dots, e_n \quad (7)$$

Let us take a random vector  $v$ .

$$v = c_1 e_1 + c_2 e_2 + \dots + c_n e_n \quad (8)$$

Now

$$\mathbf{A}^\top \mathbf{A}v = c_1 \lambda_1 e_1 + c_2 \lambda_2 e_2 + \cdots + c_n \lambda_n e_n \quad (9)$$

$$v_1 = \frac{\mathbf{A}^\top \mathbf{A}v}{\|\mathbf{A}^\top \mathbf{A}v\|} \quad (10)$$

Now continue the process by multiplying  $\mathbf{A}^\top \mathbf{A}$  with  $v_1$  and dividing with the norm and repeat this process several times.

We will get

$$v_{K+1} = \frac{\mathbf{A}^\top \mathbf{A}v_K}{\|\mathbf{A}^\top \mathbf{A}v_K\|} \quad (11)$$

Now  $v_{K+1}$  is the required eigen vector for greatest singular value.

Now let

$$y = \mathbf{A}v \quad (12)$$

$$s = \|y\|_2 = \sqrt{\sum_i y_i^2} \quad (13)$$

$$z = \mathbf{A}^\top y \quad (14)$$

$$v = \frac{z}{\|z\|} \quad (15)$$

Repeat this iteration multiple times until convergence.

After converging v(right singular vector) we get  $\sigma_1$  by

$$\sigma_1 = \|\mathbf{A}v\| \quad (16)$$

$$u_1 = \frac{\mathbf{A}v}{\sigma_1} \quad (17)$$

Now apply Deflation and find the new  $\mathbf{A}$

$$\mathbf{A}_{\text{new}} = \mathbf{A}_{\text{old}} - \sigma_k u_k v_k^\top \quad (18)$$

Now Sum all top K components to reconstruct the image:

$$C = \sum_{i=1}^K \sigma_i u_i v_i^\top \quad (19)$$

This is the low-rank approximation of the original matrix  $\mathbf{A}$ .

Each element  $C[i, j]$  is in  $[0, 1]$  multiply by 255 and clamp to 0–255.

Now finally we can get the required output.

## Pseudo Code

These are the important functions of my c code.

FUNCTION SVD(A, m, n, u, v, p, tol):

Create vector y of size m initialized to 0

Create vector z of size n initialized to 0

Initialize vector v with random values

ps = 0

LOOP k from 1 to p:

$y = A * v$

$s = \text{norm}(y)$

IF  $s == 0$ : EXIT LOOP

$z = A^T * y$

$nz = \text{norm}(z)$

IF  $nz == 0$ : EXIT LOOP

Normalize  $v = z / nz$

$IF s - ps \leq tol * (1 + ps) : BREAK$

ps = s

END LOOP

$y = A * v$

$u = y / s$

Free y, z

RETURN s

END FUNCTION

FUNCTION subtractrank1(A, m, n, u, v, s):

FOR i = 1 to m:  $t = u[i] * s$  FOR j = 1 to n:  $A[i][j] = A[i][j] - t * v[j]$

END FUNCTION

## Reconstructed Images for different K :

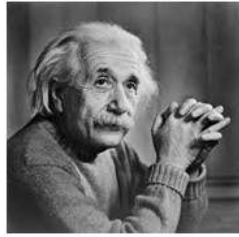
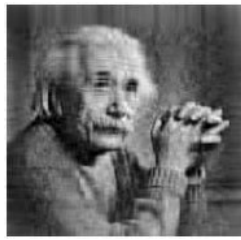


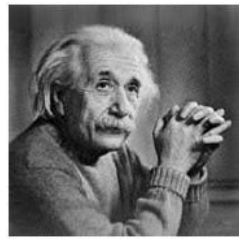
Figure 1: Original



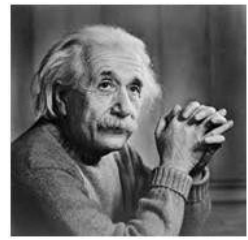
(a)  $k = 5$



(b)  $k = 20$



(c)  $k = 50$

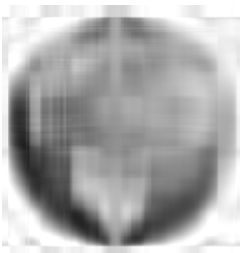


(d)  $k = 100$

Input image name	k=5	k=20	k=50	k=100
einstein.jpg	4713.594016	2126.560684	880.503600	164.781299



Figure 3: Original



(a)  $k = 5$



(b)  $k = 20$



(c)  $k = 50$



(d)  $k = 100$

Input image name	k=5	k=20	k=50	k=100
globe.jpg	20704.275969	10634.417315	6185.644086	3672.902810

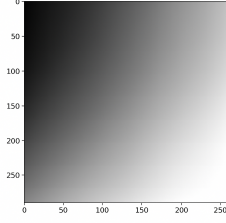
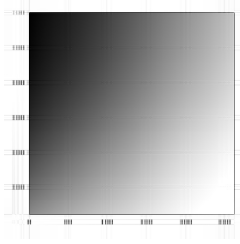
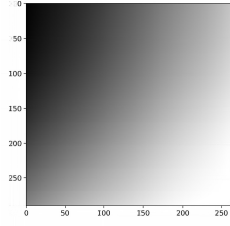


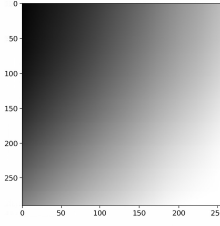
Figure 5: Original



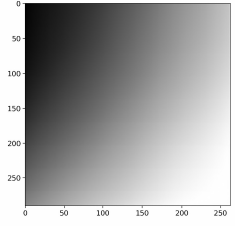
(a)  $k = 5$



(b)  $k = 20$



(c)  $k = 50$



(d)  $k = 100$

Input image name	$k=5$	$k=20$	$k=50$	$k=100$
greyscale.png	11148.127344	3798.725768	1126.298728	440.916629

## ERROR ANALYSIS

The **Frobenius norm** tells us how close the reconstructed matrix  $\mathbf{A}_K$  is to the original matrix  $\mathbf{A}$ .

$$E = \|\mathbf{A} - \mathbf{A}_K\|_F \quad (20)$$

where  $\|\cdot\|_F$  is the Frobenius Norm.

Input image name	$k=5$	$k=20$	$k=50$	$k=100$
einstein.jpg	4713.594016	2126.560684	880.503600	164.781299
globe.jpg	20704.275969	10634.417315	6185.644086	3672.902810
greyscale.png	11148.127344	3798.725768	1126.298728	440.916629

## Observation and Conclusion

### Observation:

#### 1) Effect of Rank $k$ on image quality

As the value of  $k$  increases, the reconstructed image becomes more similar to the original image.

For small values of  $k$  ( $k=5$ ) the output image appears very blurred.

For medium values of  $k$  ( $k=20, k=50$ ) the quality of output image increases when compared to the previous one.

For large values of  $k$  ( $k=100$ ) the quality of output image is almost same as that of the original one.

## 2) Error Measurement

The Frobenius norm error decreases as  $k$  increases.

The error for  $k=5$  is highest and gradually decreases for  $k=20, 50$  and  $100$ .

**Conclusion:** Image Compression using **SVD** is an effective technique for reducing the storage while maintaining image quality. By selecting a lower rank  $k$  we will not get a clear output image but as  $k$  increases the output image will be more clear and have more quality.

**SVD** provides a trade-off between **Image quality** and **Compression Level**.

This technique is good because in image compression approximations are acceptable. So image compression using **Truncated SVD** is a very good approach.