Software Assignment Report
Compressing Image using SVD

Venkata Sai - EE25BTECH11023

November 8, 2025

Introduction to SVD

Singular Value Decomposition (SVD) is a fundamental matrix factorization technique that decomposes any rectangular matrix into the product of three matrices.

Summary of Strang's video

SVD states that any matrix A can be represented as

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top \tag{1}$$

where

- **U** and **V** are orthogonal matrices (their columns are orthonormal vectors).
- Σ is a diagonal matrix containing non-negative real numbers called singular values.

The lecture shows the relationship between SVD and eigenvalues

- The vectors in **V** are eigen vectors of $\mathbf{A}^\top\mathbf{A}$
- The vectors in **U** are eigen vectors of $\mathbf{A}\mathbf{A}^\top$
- Singular values are square roots of the eigenvalues of these matrices
- The rank of a matrix equals the number of non-zero singular values.

Usage of SVD in Compression

SVD compresses images by exploiting the fact that most image information is concentrated in a few dominant singular values. When an image matrix is decomposed as

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top \tag{2}$$

The singular values in Σ are ordered from largest to smallest in descending order. The first singular value contains the greatest amount of image information, while subsequent singular values contain progressively decreasing amounts of information. By retaining only the first k singular values and discarding the rest, the lower singular values—which contain negligible or less important information—can be removed without significant image distortion.

Different types of Algorithms for Compressing the Image

1. Power Method

The power method is an iterative technique that directly computes the dominant eigenvalue of a matrix.

Advantages: - Simple and computationally inexpensive.

Disadvantages: - Finds Only the Dominant 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: - Can find any Eigenvalue.

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.

4. QR Iteration Method

QR Iteration is a method that uses repeated similarity transformations to systematically force a matrix into a triangular shape, revealing its eigenvalues on the diagonal.

Advantages: - Can find any eigenvalue. - Applicable to any matrix.

Disadvantages: - Can be Slow.

Power Iteration and its Advantages

The Power Iteration method is one of the most fundamental and intuitive eigenvalue algorithms.It starts with an initial guess for the eigenvector and repeatedly applies the matrix to approximate the eigenvalue. With each iteration, the resulting vector aligns more closely with the direction of the eigenvector associated with the largest eigenvalue.

Advantages of Power Iteration:

- Simple to Implement: The core of the algorithm is just a loop containing a matrix-vector multiplication. This makes it one of the easiest eigenvalue algorithms to understand and code
- Low Memory Usage: The method requires storage for the matrix and only a couple of vectors. It does not need to create large, dense intermediate matrices, making it suitable for very large systems.

Why I Used the Power Iteration Method

The Power Iteration method is extremely efficient if you only need the largest eigenvalue. For applications like image compression, where the most significant component is all that matters, this method avoids the massive computational waste of calculating all eigenvalues, as methods like QR or Jacobi would do.

Math Logic behind SVD compression

Problem Statement: A grayscale image represented as a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ where each entry $\mathbf{A_{ij}}$ is a pixel value.

To Do: Compress $\mathbf{A}$ by keeping only k dominant singular values.

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top \tag{3}$$

where

$$\mathbf{V} = \begin{pmatrix} v_1 & v_2 & v_3 & \dots \end{pmatrix} \text{ and } v_1, v_2, v_3 \dots \text{ are eigen vectors of } \mathbf{A}^\top\mathbf{A} \tag{4}$$

$$\mathbf{U} = \begin{pmatrix} u_1 & u_2 & u_3 & \dots \end{pmatrix} \tag{5}$$

Σ is a diagonal matrix containing singular values$(\sigma_1, \sigma_2, \sigma_3 \dots)$

From SVD Decomposition

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top \tag{6}$$

$$\mathbf{A}^\top\mathbf{A} = \left(\mathbf{U}\Sigma\mathbf{V}^\top\right)^\top\mathbf{U}\Sigma\mathbf{V}^\top = \mathbf{V}\Sigma\mathbf{U}^\top\mathbf{U}\Sigma\mathbf{V}^\top = \mathbf{V}\Sigma^2\mathbf{V}^\top \tag{7}$$

This shows that σ^2 is an eigenvalue(λ) of $\mathbf{A}^\top\mathbf{A}$

Let x be any random vector

$$\mathbf{x} = c_1\mathbf{v_1} + c_2\mathbf{v_2} + c_3\mathbf{v_3}\dots \tag{8}$$

After multiplying the vector $\mathbf{x}$ by k times, we get a new vector $\mathbf{x_{new}}$ which has been aligned in the direction of the eigen vector of $\mathbf{A}^\top\mathbf{A}$

$$\mathbf{x_{new}} = \left(\mathbf{A}^\top\mathbf{A}\right)^k\mathbf{x} \tag{9}$$

$$\mathbf{x_{new}} = \left(\mathbf{A}^\top\mathbf{A}\right)^k\left(c_1\mathbf{v_1} + c_2\mathbf{v_2} + c_3\mathbf{v_3}\dots\right) \tag{10}$$

$$\mathbf{x_{new}} = c_1\left(\mathbf{A}^\top\mathbf{A}\right)^k\mathbf{v_1} + c_2\left(\mathbf{A}^\top\mathbf{A}\right)^k\mathbf{v_2} + c_3\left(\mathbf{A}^\top\mathbf{A}\right)^k\mathbf{v_3}\dots \tag{11}$$

$$\mathbf{x_{new}} = c_1\left(\lambda_1^k\right)\mathbf{v_1} + c_2\left(\lambda_2^k\right)\mathbf{v_2} + c_3\left(\lambda_3^k\right)\mathbf{v_3}\dots \tag{12}$$

$$\mathbf{x_{new}} = c_1\left(\sigma_1^{2k}\right)\mathbf{v_1} + c_2\left(\sigma_2^{2k}\right)\mathbf{v_2} + c_3\left(\sigma_3^{2k}\right)\mathbf{v_3}\dots \tag{13}$$

Since in SVD, $|\sigma_1| > |\sigma_2| > |\sigma_3|$,

$$\mathbf{x_{new}} \sim c_1\left(\sigma_1^{2k}\right)\mathbf{v_1} \tag{14}$$

for which now $\mathbf{x_{new}}$ is in the direction of eigen vector $\mathbf{v_1}$.Now normalising the $\mathbf{x_{new}}$ we get the new eigen vector

$$\mathbf{v_{new}} = \frac{\mathbf{x_{new}}}{\|\mathbf{x_{new}}\|} \tag{15}$$

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top \tag{16}$$

$$\mathbf{AV} = \mathbf{U}\Sigma\mathbf{V}^\top\mathbf{V} \implies \mathbf{AV} = \mathbf{U}\Sigma \tag{17}$$

$$\mathbf{Av_i} = \sigma_i\mathbf{u_i} \tag{18}$$

$$\mathbf{A} = \sigma_1\mathbf{u_1}\mathbf{v_1} + \sigma_2\mathbf{u_2}\mathbf{v_2}\dots \tag{19}$$

Now we subtract the largest σ from $\mathbf{A}$ by subtracting $\mathbf{Av_{new}v_{new}}$(From (18) $\sigma\mathbf{u}$ can be written as $\mathbf{Av}$)

Pseudo Code

```
FUNCTION reconstruct(input_array, output_array, M, N, k)
    DECLARE u AS ARRAY OF M FLOATS
    DECLARE v AS ARRAY OF N FLOATS which is an EIGEN VECTOR to 'A
        TRANSPOSE A'
    DECLARE temp AS ARRAY OF M FLOATS to store A multiplied v in
        FUNCTION power_iteration
    DECLARE A AS A 2D ARRAY (MATRIX) OF M ROWS AND N COLUMNS
        OF FLOATS to STORE the input matrix we had obtained FROM the IMAGE

    FOR i FROM 0 TO M−1
        FOR j FROM 0 TO N−1
            SET A[i][j] TO input[i ∗ N + j]
        END FOR
    END FOR

    FOR l FROM 0 TO k−1
        FOR i FROM 0 TO N−1
            SET v[i] TO a random floating number between 0.0 and 1.0
        END FOR
        CALL power_iteration(A, v, temp, M, N) to MULTIPLY v WITH 'A
            TRANSPOSE A' 20 times by MULTIPLYING 'A TRANSPOSE' with
            temp
        CALL compute_u(A, v, u, M, N) to COMPUTE A MULTIPLIED v
        CALL out_matrix(A, u, v, ARRAY named output, M, N) to ADD 'u
            MULTIPLIED by v' to the output matrix and SUBTRACT the same
            FROM A
    END FOR

    RELEASE THE MEMORY FOR u, v, temp, and A to CLEAN UP and to be
        REUSED again
END FUNCTION
```

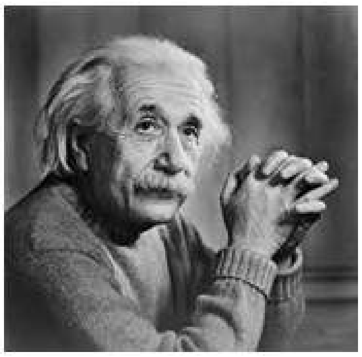
Error Analysis

Error Analysis is done using the Frobenius method.

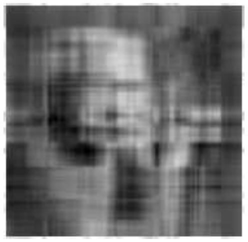
$$\|A - A_k\|_F \tag{20}$$

where $\|.\|_F$ is the Frobenius norm.

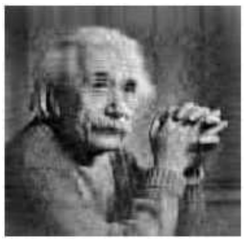
Sample Images



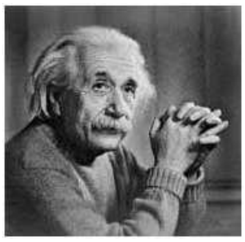
einstein.jpg



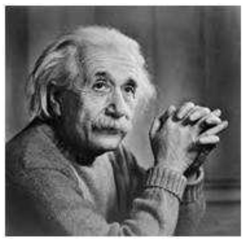
(a) k=5



(b) k=20



(c) k=50

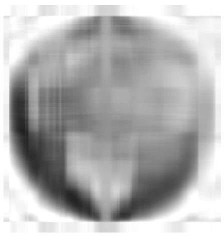


(d) k=100

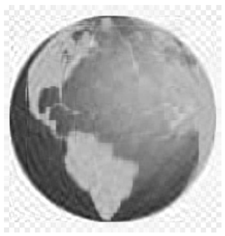
Name of the image	K=5	K=20	K=50	K=100
einstein.jpg	4721.995716	2126.820265	881.840190	164.796816

Table 1: Frobenius Error

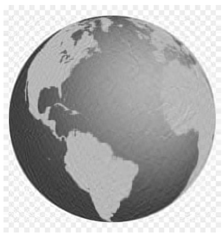
globe.jpg



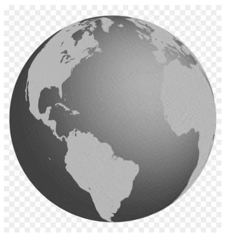
(a) k=5



(b) k=20



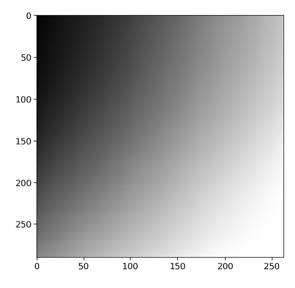
(c) k=50



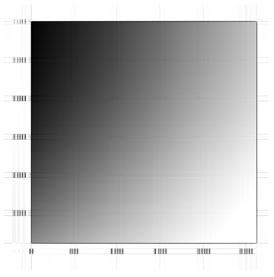
(d) k=100

Name of the image	K=5	K=20	K=50	K=100
globe.jpg	20704.274617	10642.207978	6187.175456	3674.336802

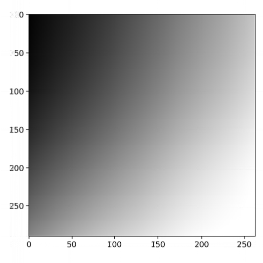
Table 2: Frobenius Error



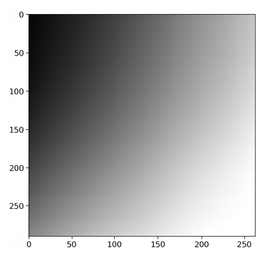
greyscale.png



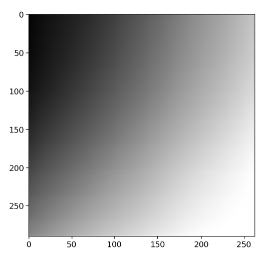
(a) k=5



(b) k=20



(c) k=50

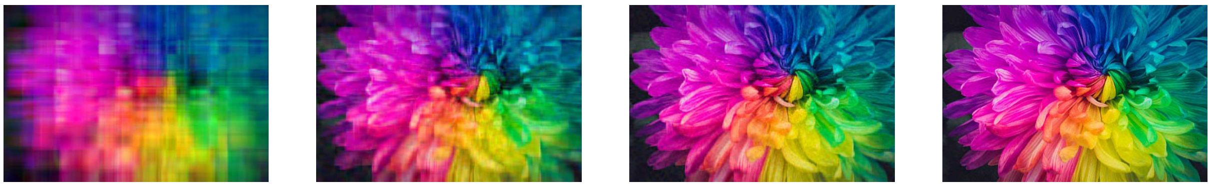


(d) k=100

Name of the image	K=5	K=20	K=50	K=100
greyscale.jpg	11157.808240	3825.360739	1208.521819	606.007087

Table 3: Frobenius Error

color.jpg


(a) k=5 (b) k=20 (c) k=50 (d) k=100

K	Channel 1(R)	Channel 2(G)	Channel 3 (B)
5	13546.095112	13746.822781	11921.025818
20	8075.180034	8605.524057	7546.784318
50	4886.374761	5266.569521	4746.590938
100	2824.716173	2895.548589	2826.996473

Table 4: Frobenius Error

Observation

As the value of k increases, the quality of the image increases and the Frobenius error decreases.

The difference between colour and grayscale images

Unlike a grayscale image, Colour images contain three channels, namely R, G, and B, in which we have to apply SVD for each channel separately and combine them at the end to reconstruct the image.

Conclusion

In conclusion, Singular Value Decomposition (SVD) proves to be a powerful and very effective method for image reconstruction and compression. This makes SVD an efficient tool for applications where storage is a great concern. SVD is a fundamental technique in linear algebra and can be applied to real-world problems, such as image compression.