Implementation of Matrix Decomposition in Image Compression Techniques

Lutfi Hakim Yusra and 13523084[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523084@itb.ac.id, [2]luthfihakimyusra@gmail.com

***Abstract*—This paper investigates the use of matrix decomposition techniques, particularly Singular Value Decomposition (SVD) and Principal Component Analysis (PCA), for image compression. By preserving only the most critical features of an image through dimensionality reduction, these methods achieve substantial storage savings with minimal loss of quality. Experimental results reveal that SVD-based compression can reduce file sizes by up to 56.3%. However, the true advantage of SVD lies in its ability to optimize storage in matrix form, achieving reductions of at least 76.5% with high K-values, highlighting its efficiency in matrix processing rather than traditional JPG file compression.**

***Keywords*—Matrix Decomposition, Image Compression, Singular Value Decomposition**

I. INTRODUCTION

In today's age, images have become an integral part of our lives, playing crucial roles in many sectors of our daily lives. As technology advances, the demand for higher quality images has grown exponentially. With the rise of higher resolution cameras and improved display technologies, images have naturally become "heavier"—requiring more storage space and computational resources. While this progression enhances visual clarity and user experience, it introduces significant challenges.

The challenges posed by large image sizes are multifaceted. For personal users, the accumulation of high-resolution images can quickly exhaust storage capacity, forcing them to rely on external storage solutions or cloud services. On a larger scale, servers hosting millions of images for social media platforms, e-commerce websites, or cloud-based applications encounter substantial storage and retrieval costs. Moreover, data scientists and researchers often rely on image datasets to develop algorithms for tasks like object recognition, facial detection, or medical imaging. Handling and analyzing massive datasets of high-resolution images can strain computational resources, slowing down the overall progress of such tasks. This issue can also be observed when dealing with applications over the internet, where bandwidth limitations can impede the transmission of large image files.

A naive solution to this problem might involve simply reducing the size of images through downscaling or lossy compression. However, such approaches often come with a significant trade-off: the loss of important details. For instance, in medical imaging, the high-resolution X-ray or MRI scan require all the crucial details for an accurate diagnosis. Therefore, there is a need for optimized approaches to image compression—methods that strike a balance between reducing file size and maintaining critical image information.

An ideal image compression technique should exhibit several key characteristics. First, it should not be computationally expensive, ensuring that it can be implemented efficiently even on devices with limited processing power. Second, it should preserve the most important visual and structural features of an image, retaining its essential information. Lastly, it should achieve a significant reduction in file size, making the storage and transmission of images more practical and cost-effective. Striving to meet these criteria has led to the exploration of various mathematical techniques, with matrix decomposition emerging as one of the most promising approaches.

Matrix decomposition, a concept taught in linear algebra, provides powerful tools for breaking down complex data structures into simpler components. Among the various decomposition methods, Singular Value Decomposition (SVD) has gained particular attention in the field of image compression. SVD is a technique that represents a matrix as a product of three other matrices, capturing its most essential features while discarding less important information. By utilizing the properties of SVD, it is possible to compress images effectively without compromising their quality too much.

This paper focuses on the application of Singular Value Decomposition (SVD) for image compression, a topic explored during linear algebra class. The usability of SVD lies in its ability to identify the underlying structure of a matrix, making it well-suited for applications like image compression, where retaining key information is crucial. In this discussion, I will delve into the mathematical foundations of SVD, emphasizing its connection to the concept of eigenvalues and eigenvectors—another fundamental topic in linear algebra. Understanding these concepts is important to understand as to why and how

SVD works so effectively for tasks like image compression. By exploring the application of SVD in image compression, this paper aims to highlight the practical relevance of linear algebra in solving real-world problems.

II. Theoretical Basis

A. Image Matrix Representation

Digital images are typically represented as 3D matrices, where the dimensions correspond to the height, width, and color channels of the image. Each pixel in an image is described by its RGB (Red, Green, Blue) values, which define its color intensity[1]. For a color image of size m x n, the representation can be expressed as:

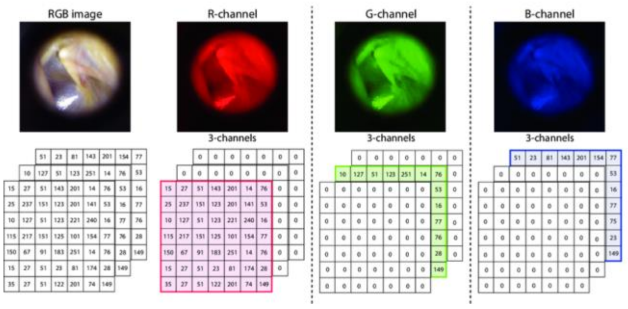


Figure 2.1 Color Channel Matrix Representation
https://www.researchgate.net/figure/Matrix-representation-of-a-digital-image-upper-row-from-left-to-right-image-in-the_fig2_359806413

where R, G, and B are 2D matrices representing the red, green, and blue channels, respectively. Each matrix contains values ranging from 0 to 255, corresponding to the intensity of the respective color component. In grayscale images, this representation simplifies to a single 2D matrix, which is a result of the formula:

$$I(x,y) = 0.2989 \cdot R(x,y) + 0.5870 \cdot G(x,y) + 0.1140 \cdot B(x,y)$$

Figure 2.2 Grayscale Formula
Source: Tugas Besar 2 Algeo 2024

This matrix-based representation provides a foundation for applying mathematical techniques like SVD, enabling advanced operations such as compression, filtering, and enhancement. By treating the color channels as independent matrices, these methods can be applied to each channel separately to optimize storage and computational efficiency.

B. Linear Algebra

Eigenvalues and eigenvectors are important concepts in linear algebra that will be important, representing scalars that reveal key properties of a matrix and its corresponding linear transformation. For a square matrix, an eigenvalue satisfies the equation:

$$Av = \lambda v$$

where A is a square matrix, v is a non-zero eigenvector, and λ is an eigenvalue[2]. To compute eigenvalues, we use the characteristic equation

$$det(A - \lambda I) = 0$$

where I is the identity matrix with the same size as *A*. This equation will yield eigenvalues and can also be used to compute eigenvalues to yield eigenvectors, which will be important in matrix decomposition. Eigenvalues play a significant role in prioritizing important features of the image, and determining what gets thrown away or kept throughout the compression process.

Matrix decomposition is the process of breaking down matrices into simpler, more interpretable components. Among these techniques, Singular Value Decomposition (SVD) will be the main focus of this paper. SVD decomposes a matrix into three components[3]:

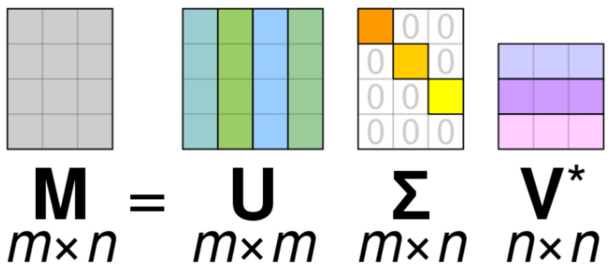


Figure 2.3 Singular Value Decomposition
https://en.wikipedia.org/wiki/Singular_value_decomposition#

where U represents the left singular vectors of A, ∑ represents the singular values, which are equal to the square root of the corresponding eigenvalues, and V* represents the right singular vectors, which is the transpose of an orthogonal matrix, made up of orthonormal eigenvectors of *A(transposed) x A*. By putting the most significant singular values on top, we can discard the less important details, enabling efficient data reduction. This will be the main characteristic in image compression.

The definition of image compression is the process of reducing the amount of data required to store or transfer an image while maintaining as much of the image's visual and structural quality as possible. In image compression, we can apply SVD reduction by truncating singular values up to a certain threshold [4], selecting a subset of the most significant singular values from the diagonal matrix. We can create the compressed image matrix A with the following formula:

$$A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \ldots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T$$

Figure 2.4 Truncated SVD
https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-22-Singular-value-decomposition-Bagian2-2023.pdf

This approximation can be observed in the following diagram.

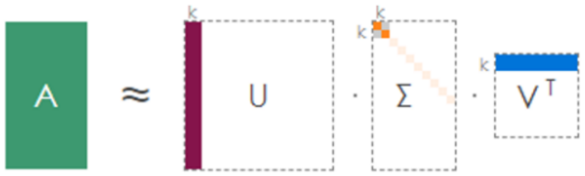


Figure 2.5 Truncated SVD Diagram
https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-22-Singular-value-decomposition-Bagian2-2023.pdf

C. Similarity Calculation

After that, we need a method of comparing the features before and after the compression as a degree of quality. This can be achieved with Mean Squared Error (MSE). MSE is gained from measuring the average squared difference between pixel intensities of the original and compressed images [4]. A lower value would indicate a higher similarity, with 0 indicating perfect similarity. It can be calculated from the following formula:

$$MSE = \frac{1}{m \times n} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} [f(i,j) - g(i,j)]^2$$

Figure 2.6 Mean Squared Error Formula
https://pemrogramanmatlab.com/tag/mse-adalah/

With m and n being the row and column, respectively.

III. Image Compression Application

In order to create the program, I used Python as the programming language of choice, because it has many modules that can be used. Python is a versatile, high-level programming language that is widely used in data analysis, machine learning, and image processing due to its extensive library.

In developing the algorithms, I use NumPy to help with matrix calculations. NumPy is a core library for numerical and matrix operations in Python. It provides support for multidimensional arrays, efficient numerical computations, and various linear algebra functions. From basic operations such as multiplying and adding arrays directly, to truncation and reconstruction of matrices.

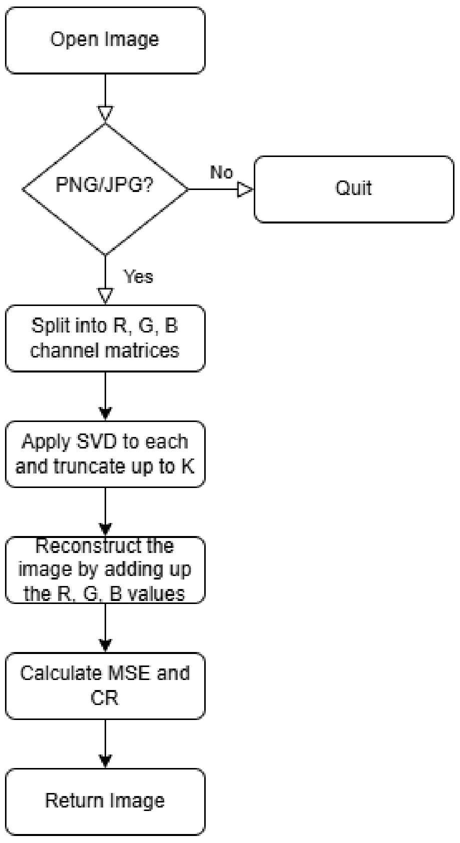
Pillow (PIL) is used as the main library to handle the image-to-matrix conversion and vice versa. It provides image opening, filtering, and saving images. With PIL, it will read an image file into a matrix that can be used freely. One thing to note regarding saving matrices into a physical file is that storing it in a PNG or JPG file would increase the file size when compared to simply storing the matrix, because of different formats have their own space saving algorithm, that is not directly compatible with the matrix representation of images.

To determine the quality of the compression algorithm and determining the optimal K-value, we will calculate the MSE as stated before, alongside the Compression Ratio (CR). Simply put, the Compression Ratio is the original image size as a matrix divided by the compressed data size. The formula is as follows:

$$\text{Compression Ratio} = \frac{\text{Size of original image}}{\text{Size of compressed image}}$$

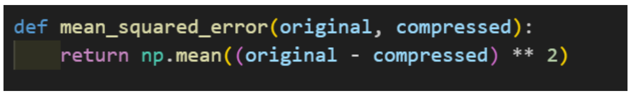
Figure 3.1 Compression Ratio
https://www.researchgate.net/publication/333421316_An_Improved_SVD_based_Image_Compression

The flow of the program will mainly be composed of the following diagram. We open the image into matrices that would be ready for processing. Then, we apply SVD into the matrix for each color, splitting them into the three components. Then, we remake the image by truncating the SVD components and then adding them back into a matrix. The resulting matrix would then be calculated through the MSE alongside the base matrix for comparison, then saved as an image. More details regarding the program can be seen in the following link:
https://github.com/pixelatedbus/svd_compression



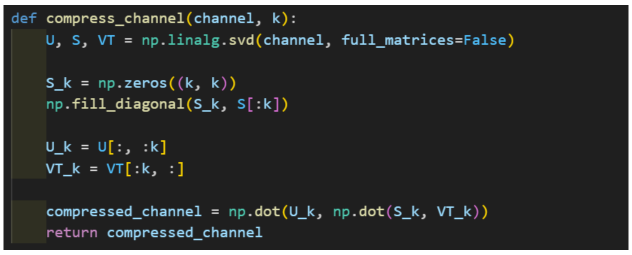
Open Image
PNG/JPG?
No
Quit
Yes
Split into R, G, B channel matrices
Apply SVD to each and truncate up to K
Reconstruct the image by adding up the R, G, B values
Calculate MSE and CR
Return Image

Figure 3.2 Program Flow
Private Documentation



```python
def mean_squared_error(original, compressed):
    return np.mean((original - compressed) ** 2)
```

Figure 3.3 MSE Calculation
Private Documentation



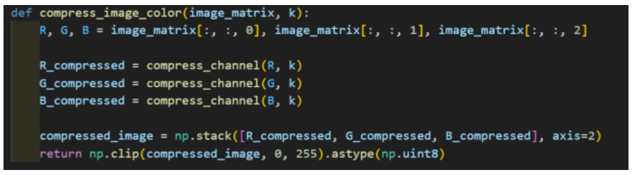
```python
def compress_channel(channel, k):
    U, S, VT = np.linalg.svd(channel, full_matrices=False)

    S_k = np.zeros((k, k))
    np.fill_diagonal(S_k, S[:k])

    U_k = U[:, :k]
    VT_k = VT[:k, :]

    compressed_channel = np.dot(U_k, np.dot(S_k, VT_k))
    return compressed_channel
```

Figure 3.4 SVD Compression on One Channel
Private Documentation

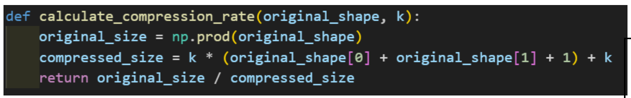


```python
def compress_image_color(image_matrix, k):
    R, G, B = image_matrix[:, :, 0], image_matrix[:, :, 1], image_matrix[:, :, 2]

    R_compressed = compress_channel(R, k)
    G_compressed = compress_channel(G, k)
    B_compressed = compress_channel(B, k)

    compressed_image = np.stack([R_compressed, G_compressed, B_compressed], axis=2)
    return np.clip(compressed_image, 0, 255).astype(np.uint8)
```

Figure 3.5 SVD Compression for RGB Image
Private Documentation



```python
def calculate_compression_rate(original_shape, k):
    original_size = np.prod(original_shape)
    compressed_size = k * (original_shape[0] + original_shape[1] + 1) + k
    return original_size / compressed_size
```

Figure 3.6 Compression Rate
Private Documentation

IV. EXPERIMENTAL

To evaluate the performance of the developed code, several images were selected for testing. The first image, Firefly.jpg, has dimensions of 2196 x 2196 and a file size of 343 KB. The second image, Hina.jpg, measures 343 x 497 and has a size of 52 KB. The third image, Citlali.jpg, is a grayscale image with dimensions of 686 x 387 and a file size of 55.6 KB. These images were compressed using multiple K-values and analyzed to compare the results across the different test cases.

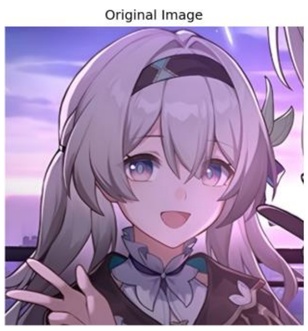


Figure 4.1 Firefly.jpg

https://www.google.com/url?sa=i&url=https%3A%2F%2Fid.pinterest.com%2Fdoctorgx9%2Ffirefly-hsr%2F&psig=AOvVaw2eRr_tbQrrBd0Qyp_c7diJ&ust=1735813602029000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCIjYhd2n1IoDFQAAAAAdAAAAABAE

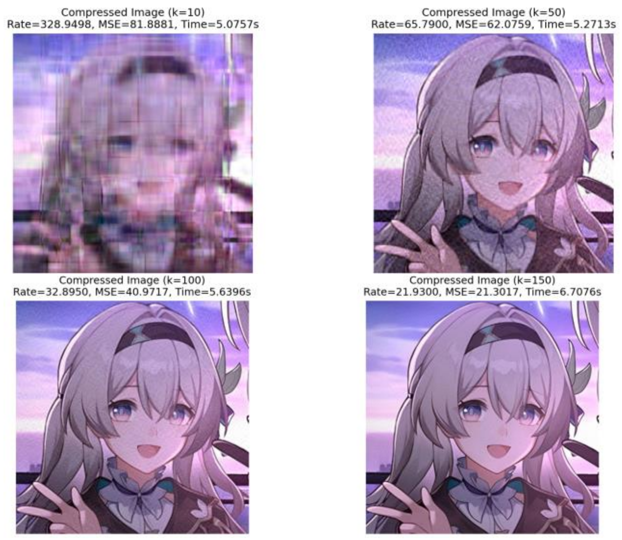

Compressed Image (k=10)
Rate=328.9498, MSE=81.8881, Time=5.0757s
Compressed Image (k=50)
Rate=65.7900, MSE=62.0759, Time=5.2713s
Compressed Image (k=100)
Rate=32.8950, MSE=40.9717, Time=5.6396s
Compressed Image (k=150)
Rate=21.9300, MSE=21.3017, Time=6.7076s

Figure 4.2 Firefly.jpg SVD Compression
Private Documentation

Table 4.1 Firefly.jpg Compression Results

K	CR	MSE	Time	File size (KB)
10	328.95	81.89	5.07	187
50	65.79	62.08	5.27	284
100	32.90	40.98	5.63	307
150	21.93	21.30	6.71	310

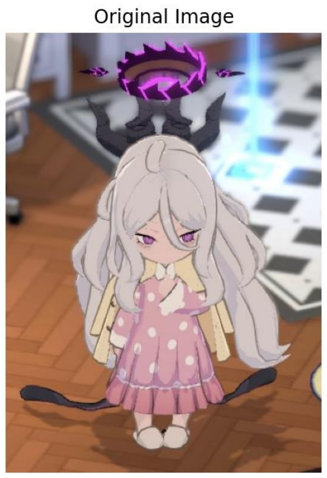


Figure 4.3 Hina.jpg
https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.reddit.com%2Fr%2FBlueArchive%2Fcomments%2F19ejj9f%2Fsome_screen_of_the_new_game_mode_with_hina_new%2F&psig=AOvVaw1jG6zQCl9TDVhJtgOQLzEP&ust=1735813771187000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCNij-a2o1IoDFQAAAAAdAAAAABAE

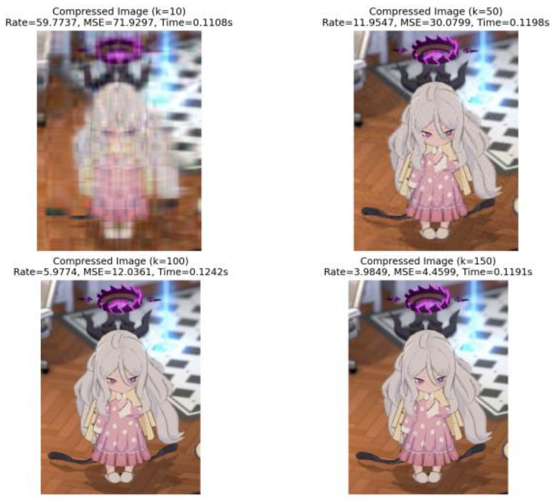
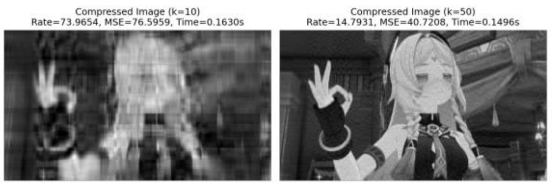

Compressed Image (k=10)
Rate=59.7737, MSE=71.9297, Time=0.1108s
Compressed Image (k=50)
Rate=11.9547, MSE=30.0799, Time=0.1198s
Compressed Image (k=100)
Rate=5.9774, MSE=12.0361, Time=0.1242s
Compressed Image (k=150)
Rate=3.9849, MSE=4.4599, Time=0.1191s

Figure 4.4 Hina.jpg SVD Compression

Table 4.2 Hina.jpg Compression Results

K	CR	MSE	Time (s)	File Size (KB)
10	59.77	71.93	0.11	29.1
50	11.95	30.08	0.12	39.8
100	5.98	12.04	0.12	40.8
150	3.98	4.46	0.12	41.5

Figure 4.5 Citlali.jpg
Grayscaled, from
https://i.ytimg.com/vi/b0YucpgNMM0/hq720.jpg?sqp=-oaymwEhCK4FEIIDSFryq4qpAxMIARUAAAAAGAElAADIQj0AgKJD&rs=AOn4CLD4E27KCeOtlcAvD0f_dEPBO_PJsA


Compressed Image (k=10)
Rate=73.9654, MSE=76.5959, Time=0.1630s
Compressed Image (k=50)
Rate=14.7931, MSE=40.7208, Time=0.1496s

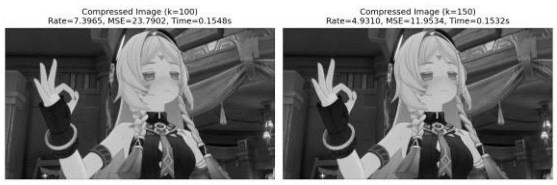

Compressed Image (k=100)
Rate=7.3965, MSE=23.7902, Time=0.1548s
Compressed Image (k=150)
Rate=4.9310, MSE=11.9534, Time=0.1532s

Figure 4.6 Citlali.jpg SVD Compression

Table 4.3 Citlali.jpg Compression Results

K	CR	MSE	Time (s)	File Size (KB)
10	73.97	76.60	0.15	24.3
50	14.80	40.72	0.15	34.5
100	7.40	23.76	0.15	37.5
150	4.93	12.00	0.15	36.0

Throughout the sample tests, we observed that the algorithm effectively reduced file sizes when saving the compressed images as JPGs. However, for larger-sized JPGs with a high rank, the reduction in file size became barely noticeable. For instance, the difference in file size between a 100 K-value and a 150 K-value was minimal. This can be attributed to the fact that the 100 K-value already retains nearly as much important information as the 150 K-value, rendering the PNG or JPG formatting less impactful on file size. Furthermore, it became evident that higher ranks demand more computational resources and time, which is reflected in the slightly larger file sizes for higher K-values.

It is important to highlight that the compression ratio of images demonstrates the unique advantage of SVD compression. This method proves to be far more effective for storing images in the form of matrices, which are particularly useful for further processing and analysis, compared to directly converting them into JPGs. As shown in the data tables above, a maximum of 56.3% of the original storage can be saved when compressing into a JPG format, whereas a minimum of 75.6% of the original image matrix can be saved through SVD compression into matrix representation even with a high K-value. These results emphasize the efficiency of SVD in reducing data storage while maintaining a significant portion of the image's critical information. This method shines particularly when prioritizing storage optimization for processing over mere file conversion.

V. Evaluation

From the conducted experiment, the pros and cons of using SVD as the main image compression method are as follows.

A. Pros of SVD Compression

- Efficient Dimensionality Reduction
  SVD compression is highly effective in reducing the dimensionality of image data. By retaining only the most significant singular values, it minimizes storage requirements while preserving the essential features of the image.
- Preservation of Image Quality

Even at lower K-values, SVD manages to retain the core structure and details of an image. This ensures that compressed images maintain a high level of perceptual similarity to the original, making the method ideal for applications requiring visual fidelity.

- Versatility Across Data Types
  SVD can be applied to both grayscale and color images by treating the image as a matrix (or set of matrices). This versatility makes it applicable to a wide range of use cases, including those involving multi-channel data.
- Customizable Compression Levels
  The ability to adjust the KKK-value gives users control over the trade-off between compression ratio and image quality. This flexibility allows the method to be tailored to different storage and computational requirements.
- Efficient Matrix-Based Storage
  SVD compression is particularly beneficial for storing images in matrix form, which is advantageous for data processing tasks where images are treated as mathematical objects, such as machine learning or numerical simulations.

B. Cons of SVD Compression

- Computationally Intensive
  The computation of Singular Value Decomposition, especially for high-resolution images or large matrices, is resource-intensive. It requires significant processing power and memory, which can be a limitation for large datasets or real-time applications.
- Not Optimized for Direct Visualization
  SVD compression focuses on mathematical representation rather than direct visualization. As a result, compressed images in their decomposed matrix form are not immediately viewable, requiring reconstruction before visualization.
- Format-Specific Limitations
  SVD does not integrate with standard image compression formats (e.g., JPG, PNG) as a native technique. This makes it less practical for end-users who prioritize smaller file sizes over efficient mathematical representation.

VI. Conclusion

The SVD compression algorithm demonstrates a powerful and versatile approach to reducing data storage requirements while maintaining essential information, particularly in matrix representations. Through the careful selection of K-values, SVD compression allows for a balance between quality retention and computational efficiency, highlighting its adaptability to different use cases.

However, it is important to note that while SVD achieves high compression ratios in terms of matrix storage, its impact on file size for standard image formats like JPG may be limited. This limitation arises from the file format’s own compression mechanisms, which can overshadow the benefits of SVD for higher-rank approximations. Nonetheless, this does not diminish the value of SVD compression in scenarios where images need to be stored, transmitted, or analyzed in their matrix form, such as in machine learning, scientific research, and other data-intensive applications.

In conclusion, the SVD compression algorithm highlights the practical utility of linear algebra in solving real-world challenges. Its ability to compress data effectively while retaining essential information makes it a valuable tool, particularly in fields where matrix representation and data processing efficiency are critical. By combining mathematical rigor with practical application, SVD compression showcases the profound impact of mathematical techniques on modern computational challenges.

VI. Acknowledgment

The author wishes to express heartfelt gratitude to several parties for their support in the creation of this paper. First, gratitude is extended to God for His guidance throughout the process of learning and writing. The author also acknowledges the invaluable teachings and support of Mr. Rinaldi Munir, the lecturer of ITB's Aljabar Linier dan Geometri IF2123 course, whose guidance greatly enriched the learning experience. Finally, the author thanks their family and friends for their unwavering support throughout the semester.

References

[1] Vesna Vučković. IMAGE AND ITS MATRIX, MATRIX AND ITS IMAGE, Преглед НЦД 12 (2008), 17–31
[2] Mishra, Kapil & Singh, Satish Kumar & Nagabhushan, P.. (2018). An Improved SVD based Image Compression. 10.1109/INFOCOMTECH.2018.8722414.
[3] Swathi, H & Sohini, Shah & Surbhi, & Gi, Gopichand. (2017). Image compression using singular value decomposition. IOP Conference Series: Materials Science and Engineering. 263. 042082. 10.1088/1757-899X/263/4/042082
[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2024-2025/algeo24-25.htm (Accessed in 28 December 2024)

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 27 Desember 2024

[illegible]

Lutfi Hakim Yusra 13523084