Image Compression Using K-Means Clustering

Muhammad Luqman Hakim (13523044)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13523044@std.stei.itb.ac.id muhluqhakim@gmail.com

***Abstract*—Image compression is an essential process in reducing the storage or transmission costs of digital images. This study explores the application of K-Means Clustering for lossy image compression. By partitioning the image's pixel data into clusters and replacing each pixel values with their corresponding cluster centroids, the image's size may be reduced without substantial degradation in quality.**

***Index Terms*—image compression, k-means clustering**

I. Introduction

Image compression techniques aim to reduce the amount of data required to represent an image, making it easier to store, transmit and process without significantly compromising visual quality. This study investigates the application of k-means clustering for introducing lossy image compression to further reduce the storage cost of images stored in the PNG file format.

K-means clustering works by partitioning image pixel values, represented as vectors, into clusters, each represented by a centroid, such that the sum of the euclidean distance of every vectors to its respective centroid is minimized. By replacing pixel values with their corresponding cluster centroids, the resulting image can be represented with less color value, which is more easily compressed by the compression algorithm used in encoding PNG files. This effectively reduces the image size while maintaining a reasonable approximate representation of the original content.

The remainder of this paper explores the implementation of K-Means for image compression and evaluates its performance in terms of compression ratio and image quality. By analyzing the compression ratio and image quality produced by this method, this research provides a comprehensive analysis of K-Means Clustering as a modern solution for image compression.

II. Real-Valued Vectors

A real-valued vector is a fundamental mathematical entity used to represent quantities in a space where each dimension is associated with a real number. Mathematically, a real-valued vector is defined as an ordered tuple of real numbers. For example, in a two-dimensional space, a vector $\mathbf{v}$ can be written as $\mathbf{v} = (v_1, v_2)$, where v_1 and v_2 are real numbers representing the vector's position along the x-axis and y-axis, respectively.

In general, a vector in n-dimensional real-valued space (denoted $\mathbb{R}^n$) is expressed as:

$$\mathbf{v} = (v_1, v_2, \dots, v_n),$$

where n is the number of dimensions, and each component v_i is a real number. These components may include any value from the set of real numbers, including integers, fractions, and irrational numbers.

Geometrically, a real-valued vector can be visualized as a directed line segment originating from a reference point, typically the origin of the coordinate system. The vector's components $(v_1, v_2, \dots, v_n)$ determine its position in the space. In $\mathbb{R}^2$, the vector $\mathbf{v} = (v_1, v_2)$ corresponds to a point in a two-dimensional plane. In $\mathbb{R}^3$, $\mathbf{v} = (v_1, v_2, v_3)$ represents a point or direction in three-dimensional space. In higher dimensions ($n > 3$), the vector exists in abstract spaces that are challenging to visualize but are mathematically well-defined.

While vectors are often visualized geometrically, their application extends to representing non-geometric data, such as pixel values in images. In the context of this study, vectors are used to encode the RGB color values of pixels, where each pixel is represented as a three-dimensional vector corresponding to its red, green, and blue intensity levels. This abstraction allows us to treat image data as points in a multi-dimensional color space, enabling mathematical operations like clustering to be applied effectively. By grouping similar pixel vectors using the k-means algorithm, we reduce the number of unique colors, thereby compressing the image while preserving its visual quality. This approach demonstrates the flexibility of vectors as tools for representing and processing diverse forms of data, beyond their traditional geometric interpretations, and highlights their critical role in the compression algorithm presented in this paper.

III. Euclidean Distance

The Euclidean distance is a fundamental concept in geometry and is widely used across various fields such as data analysis, machine learning, and image processing. It measures the straight-line distance between two points in Euclidean space and is equivalent to the length of the line segment connecting these points. This metric provides a natural and intuitive way to quantify the similarity or difference between two points in a given space.

In a general n-dimensional real vector space, the Euclidean distance between two points $\mathbf{a} = (a_1, a_2, \dots, a_n)$ and $\mathbf{b} = (b_1, b_2, \dots, b_n)$ is defined as:

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}, \quad (1)$$

which can also be expressed in vector notation as:

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\| = \sqrt{(\mathbf{a} - \mathbf{b}) \cdot (\mathbf{a} - \mathbf{b})}, \tag{2}$$

where $\|\cdot\|$ represents the vector norm, and $\cdot$ denotes the dot product of two vectors.

A. Geometric Interpretation

The Euclidean distance can be interpreted geometrically as the hypotenuse of a right triangle formed by the differences between the corresponding coordinates of **a** and **b**. In a two-dimensional space, this corresponds to the Pythagorean theorem:

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\Delta x^2 + \Delta y^2},$$

where Δx and Δy are the horizontal and vertical distances between the two points.

In higher dimensions, the Euclidean distance generalizes this concept by incorporating all coordinate differences, effectively extending the Pythagorean theorem to n-dimensional space.

B. Applications in Image Compression

In the context of this study, the Euclidean distance plays a crucial role in clustering algorithms such as K-Means. When applied to image compression, the distance is used to measure the similarity between pixel values, represented as vectors in a three-dimensional RGB color space. Specifically, the distance between two pixel vectors $\mathbf{p}_1 = (r_1, g_1, b_1)$ and $\mathbf{p}_2 = (r_2, g_2, b_2)$ is calculated as:

$$d(\mathbf{p}_1, \mathbf{p}_2) = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}.$$

This distance metric enables the algorithm to group similar colors into clusters, with each cluster represented by its centroid. By minimizing the Euclidean distances within clusters, the algorithm ensures that the compressed image retains visual fidelity while reducing the number of unique colors.

IV. K-Means Clustering

Given a set of vectors and a constant k, k-means clustering aims to partition the set into k cluster, each associated with a centroid vector, such that the sum of squared Euclidean distance of each vector with its respective centroid is minimized.

This leads to the partitioning of the data space into Voronoi cells. K-means clustering minimizes the within-cluster variances (squared Euclidean distances), rather than the regular Euclidean distances, which would correspond to the more complex Weber problem. While the mean minimizes squared errors, the geometric median is the one that minimizes Euclidean distances. For example, more optimal Euclidean solutions can be achieved using techniques like k-medians and k-medoids.

Although this problem is NP-hard, efficient heuristic algorithms can rapidly converge to a local optimum, resulting in acceptable approximate solution. For the sake of simplicity, in this paper, Llyod's algorithm is implemented, as it is the most common algorithm used for k-means clustering.

A. Algorithm

Lloyd's algorithm is one of the most common and widely used algorithms for implementing K-means clustering. It proceeds with the following steps: first, it selects k random points from the dataset as initial centroids. There are two commonly used techniques in choosing the initial Forgy and random partition techniques. The Forgy method randomly selects k data from the dataset to be the initial centroids, which tends to result in centroids that are spread out across the data. In contrast, the random partition method first randomly assigns each data point to a cluster and then calculates the centroid of each cluster based on its randomly assigned points. This typically results in centroids that are closer to the center of the data. In the subsequent iterations, the algorithm alternates between two main steps: assignment and update. In the assignment step, each data point is assigned to the cluster corresponding to the closest centroid, with the distance measured by the squared Euclidean distance. In the update step, the centroid of each cluster is recalculated as the mean of all the points assigned to that cluster. This process repeats until the assignment of points to clusters no longer changes, indicating convergence.

The algorithm initially chooses k random points to be the centroids associated with each cluster. Then the algorithm alternates between assigning each data points based on the centroid it is closest to and recalculating each centroids as the mean over its assigned data points. These two steps are repeated until no data point is assigned to a different cluster.

```
Algorithm 1: Lloyd's Algorithm
 Data: $k$ = number of clusters
       $V = \{\mathbf{v_1} \dots \mathbf{v_N}\}$ set of N vectors
 Result: $S_1 \dots S_k$, where $S_i$ is the set of vectors in
         cluster $k$
 $\mathbf{c_1} \dots \mathbf{c_k} \leftarrow$ randomly chosen centroids;
 repeat
     /* assign cluster to each vector */
     for $i \leftarrow 1$ to $k$ do
         $S_i \leftarrow \{\mathbf{v} \in V \mid \mathbf{c_i} \in \operatorname{argmin}_{\mathbf{c} \in \{\mathbf{c_1}..\mathbf{c_k}\}} d(\mathbf{v}, \mathbf{c})^2\}$;
         /* Though some $\mathbf{v}$ may be assigned
            to multiple sets, it must be
            assigned to exactly one of
            them. Typically, the set with
            the lowest index is chosen.
            */
     end
     /* recalculate centroids */
     for $i \leftarrow 1$ to $k$ do
         $\mathbf{c_i} \leftarrow \frac{1}{|Si|} \sum_{\mathbf{v} \in S_i} \mathbf{v}$;
     end
 until converge;
```

It is evident that after each iteration, the sum of distance squared within each cluster decreases monotonically. Therefore this algorithm is guaranteed to always converge to a local optimum, but not necessarily the global one.

V. Compression Algorithm in PNG File Format

A. Overview

The Portable Network Graphics (PNG) file format employs a lossless compression algorithm to efficiently store and transmit image data while preserving its original quality. The compression process in PNG involves a combination of filtering, run-length encoding and the DEFLATE algorithm.

B. Image Filtering

Before the DEFLATE algorithm is applied, the image data undergoes a transformation using a prediction method. Specifically, a filter method is selected for the entire image, while each image line is processed using a dynamically chosen filter type. This transformation prepares the data to be more efficiently compressed. The selected filter type for a scanline is prepended to the line itself, allowing for inline decompression during decoding.

The current PNG specification defines only one filter method (method 0), meaning the primary decision involves selecting the appropriate filter type for each line. This method operates by predicting the value of each pixel based on the values of neighboring pixels and then subtracting the predicted value from the actual pixel value. Lines filtered in this way are typically more compressible than raw image data, especially when the current line closely resembles the one above it. This is because the differences between predicted and actual values tend to cluster near zero, making the data easier to compress. This process is particularly important for relating separate rows within the image since the DEFLATE algorithm treats the image as a linear stream of bytes without recognizing its two-dimensional structure.

Filter method 0 provides five types of filters, each predicting the value of a byte (from the pre-filtered image data) using one or more neighboring bytes: the byte to the left (A), the byte above (B), the byte above and to the left (C), or combinations of these. The filters encode the difference between the actual and predicted values. The filters are applied to byte values, not to entire pixels. Pixel values may span one or more bytes or consist of multiple components per byte but are always processed within byte boundaries. The five filter types are listed at Table I.

Type Byte	Filter Name	Predicted Value
0	None	Zero (raw byte value passes through unaltered)
1	Sub	Byte A (to the left)
2	Up	Byte B (above)
3	Average	Mean of bytes A and B, rounded down
4	Paeth	A, B, or C, whichever is closest to $p = A + B - C$

TABLE I
PNG Filter Types and Their Predicted Values

The filter type is selected dynamically for each row to minimize the size of the encoded data.

C. Data Compression Using DEFLATE

The filtered image data is compressed using the DEFLATE algorithm, a lossless compression method that combines two techniques:

1) LZ77 Compression: Exploits repeated sequences in the data by replacing duplicate strings with references to earlier occurrences, significantly reducing redundancy.
2) Huffman Coding: Encodes data using variable-length codes, assigning shorter codes to more frequently occurring sequences and longer codes to less frequent ones.

This dual-stage approach ensures both high compression efficiency and adaptability to a wide range of image content.

VI. Implementation of K-Means Clustering for PNG Image Compression

The implementation of K-Means clustering for PNG image compression involves several key steps, from preprocessing the image data to applying the clustering algorithm and reconstructing the compressed image. Below is an outline of the process:

1) Preprocessing the Image: The PNG image is read and reshaped into an array of vectors where each vector represents a pixel. This allows the pixel data to be treated as a dataset for clustering.
2) Applying the K-Means Algorithm: the K-Means algorithm is applied to compress the image by partitioning its pixels intok clusters, where k is a user-defined parameter determining the number of unique colors in the compressed image. Initially, cluster centroids are randomly selected. The algorithm iteratively assigns each pixel to the nearest centroid, based on Euclidean distance and updates the centroids to the mean of their assigned pixels. Using Lloyd's algorithm, this process continues until the centroids stabilize or a maximum number of iterations is reached. To avoid excessive computation time, the number of iterations is limited at a predefined limit
3) Quantization: Once clustering is complete, each pixel in the image is replaced with the centroid of the cluster to which it belongs. This step reduces the number of unique colors in the image to k, achieving the desired simplification.
4) Reconstruction of the Image: The resulting image is reconstructed using the quantized pixel values. The image is then saved in the PNG format, which applies its own filtering and DEFLATE compression to further reduce the file size.

Clustering the pixels of the image reduce the number of colors into k, color. This creates more redundancy which can be exploited by the preprocessing and compression algorithm used in PNG. This approach demonstrates how k-means clustering can enhance the efficiency of PNG compression while maintaining high visual fidelity, making it suitable for applications requiring both quality and compact storage.

VII. Program Design and Implementation

A. Language and Libraries Choice

The program is implemented in C, utilizing the stb libraries for image processing and file handling. These lightweight, single-header libraries simplify loading and saving images in various formats, including PNG, ensuring seamless integration with the compression algorithm. Additionally, the implementation leverages the OpenMP (OMP) library to enable parallel processing, significantly improving computational performance by distributing tasks across multiple threads. This combination of tools ensures a portable, fast and reliable implementation of the K-Means clustering-based image compression method.

B. Data Structures

In the `stb_image` library, the pixel data of an image is stored in a simple, contiguous array of `unsigned char`-s. The array consist of rows of pixel with the first pixel in the array is the top-left-most pixel in the image, with each pixel represented by multiple bytes depending on the color channels. For PNG file format, each pixels are represented by 3 bytes:1 byte for the red, 1 byte for the green, and 1 byte for the blue value, consecutively.

This representation, however, is difficult to do the required computations with. Therefore another data structure is used for computing k-means method.

At the preprocessing stage of the image, the pixels are each represented as a C struct:

```
typedef struct _pixel {
 double R;
 double G;
 double B;
} pixel;
```

All the pixels of the image are then stored as an array of `pixel`-s.

The association between a pixel and the cluster it belongs to is stored in an array of `int` such that the i-th element of the array is the cluster to which the i-th pixel belongs. This is done to minimize the memory waste due to memory alignment, as opposed to storing the cluster index inside the `pixel` struct.

C. Data Flow and Context Diagram

The data flow for the program begins with the input image, which is passed to the `compress_image` program, optionally along with the number of clusters k. This image is first processed by the `stbi_load` function from the `stb_image` library, which reads the file and converts it into a raw byte array, where each pixel is represented by its color channels (e.g., RGB). The byte array is then transformed into a vector representation, treating each pixel as a three dimensional vector based on its color channels. This vectorized data is passed to the k-means clustering algorithm, which groups the pixels into k clusters based on their similarity, represented by the proximity of their colors in the vector space. The algorithm replaces each pixel's values with the centroid of its cluster. The clustered data is then converted back into a raw byte array. Finally, the `stbi_write_png` function encodes the modified byte array into a compressed PNG file, generating the output image with reduced size while maintaining acceptable visual quality.

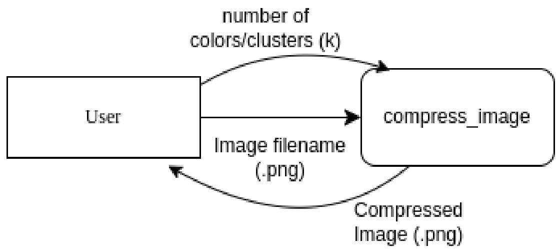

Fig. 1. Context Diagram.

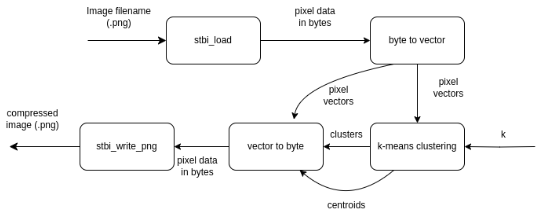

Fig. 2. Data Flow Diagram.

D. Implementation

The full implementation can be accessed at the link provided in Appendix A.

VIII. Testing and Results

A. Program Usage

The program is executed by running the executable file from terminal and giving it 2 (or 3) parameters. The fisrt parameter is the image that is to be compressed. The second parameter is the name for the resulting compressed image. The last, optional, parameter is the number of clusters (and therefor colors) that is used for the k-means algorithm. If this parameter is not included, the program defaults to 256 clusters/colors.

B. Description of the Samples

Fig. 3. and Fig. 4. are some samples of images that went through the compression.These images have been processed to reduce the number of distinct colors, representing a compressed version of the original images. Sample 1 is a digital illustration and sample 2 is a photograph. The digital illustration, with its simplified shapes, solid colors, and clear boundaries, provides a controlled environment to test how well the algorithm handles images with limited color diversity and sharp contrasts. The photograph, on the other hand, presents a

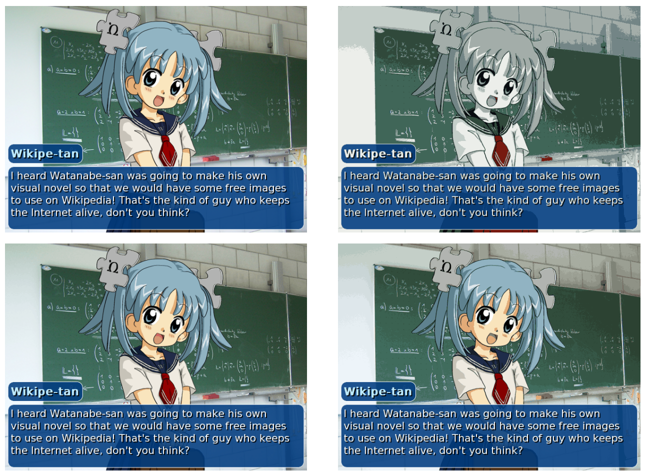
Wikipe-tan
I heard Watanabe-san was going to make his own visual novel so that we would have some free images to use on Wikipedia! That's the kind of guy who keeps the Internet alive, don't you think?
Wikipe-tan
I heard Watanabe-san was going to make his own visual novel so that we would have some free images to use on Wikipedia! That's the kind of guy who keeps the Internet alive, don't you think?
Wikipe-tan
I heard Watanabe-san was going to make his own visual novel so that we would have some free images to use on Wikipedia! That's the kind of guy who keeps the Internet alive, don't you think?
Wikipe-tan
I heard Watanabe-san was going to make his own visual novel so that we would have some free images to use on Wikipedia! That's the kind of guy who keeps the Internet alive, don't you think?

Fig. 3. Sample image 1. From top-right, clockwise, sample image compressed (k=10), sample image compressed (k=50), sample image compressed (k=250), sample image original

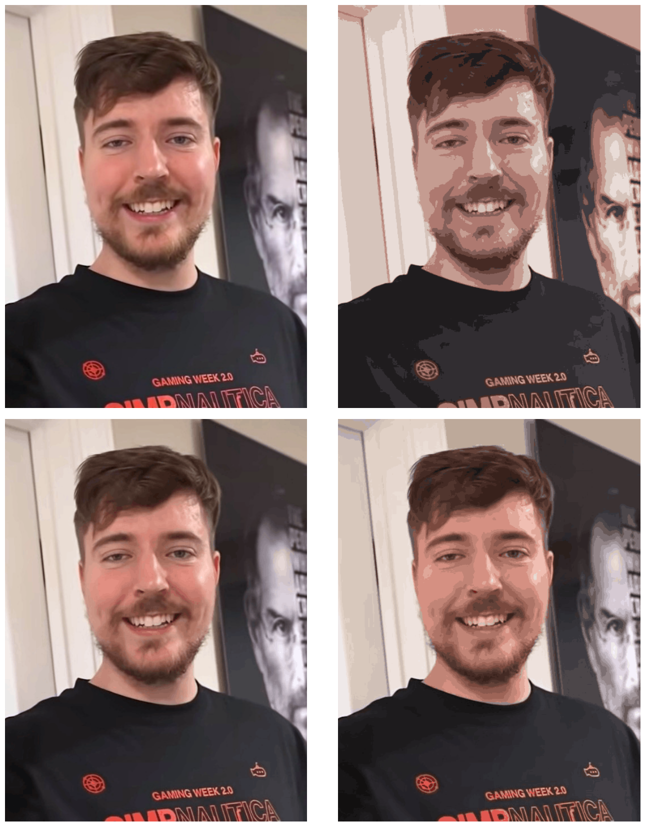

Fig. 4. Sample image 2. From top-right, clockwise, sample image compressed (k=10), sample image compressed (k=50), sample image compressed (k=250), sample image original

more complex scenario, featuring subtle color gradients, textures, and intricate details. Using both types of images allows for a comprehensive evaluation of the algorithm's performance across different image styles, assessing its ability to compress images effectively while maintaining visual quality.

C. *Image Quality*

When applying K-means clustering for image compression, the number of clusters k plays a critical role in determining the quality and compression ratio of the resulting image. At $k = 10$, the image undergoes significant compression, as only 10 distinct colors (centroids) are retained. This level of compression drastically reduces the image size, but it also leads to a loss of color detail. In the first sample, The girl's in the image appears colorless. this is due to the fact that there is not enough colors to properly represent the girl's appearance. In the second example, the man's skin color and overall appearance also seems less saturated. Additionally, the man's skin has apparent ”layers” of colors, due to the fact that there is not enough colors to express the gradient in the image. With large areas of similar colors. Fine details and subtle color variations in the image are smoothed out, and the image looks simplified, less saturated, and ”layered”. While the compression ratio is high, the loss in quality may be noticeable, especially for images that contain intricate details or gradients.

At $k = 50$, the number of colors retained increases, leading to better color representation in the compressed image. The image still exhibit some compression artifacts, but the overall appearance is better compared to the $k = 10$ case. More color variations are preserved, and finer details of the image are more visible. The same issue is still noticable, with colors clustering in areas with subtle gradients, but the image will appear significantly closer to the original version, with recognizable features and textures. The compression ratio is still reasonable, but the image quality is noticeably better than at $k = 10$.

At $k = 250$, the compression is much lighter, and 250 distinct colors are used to represent the image. This level of compression retains more fine details and color nuances, producing an image that closely resembles the original. Sample 2, which is photograph, has some visible compression artifact , whereas in sample 1, the compression artifact is unnoticable. Overall, the image is much more faithful to the original color distribution. The image appears sharper, with smoother transitions and fewer areas of uniform color. While the compression ratio is lower compared to $k = 10$ and $k = 50$, the quality of the image is much higher. This is ideal for applications where high fidelity is necessary, and the file size is less of a concern.

Both figures provide insight into the trade-off between compression ratio and image quality. As the number of clusters decreases, the compression ratio improves, but the image may lose some of its original visual fidelity. By experimenting with different values of k, it is possible to strike a balance between compression efficiency and maintaining a perceptible image quality. These visual comparisons are critical in understanding the practical implications of using k-means clustering for image compression.

D. Compression Ratio

k	Compression Ratio	
	Sample 1	Sample 2
$k = 10$	4.99	5.08
$k = 50$	2.27	2.40
$k = 250$	1.04	1.15

TABLE II
PNG FILTER TYPES AND THEIR PREDICTED VALUES

The compression ratios shown in the table correspond to the performance of the K-means image compression algorithm applied to two different samples, with varying values of k. The compression ratio is the ratio of the original image size to the compressed image size, with higher values indicating greater compression (i.e., smaller file sizes). The table presents the compression ratios for both samples at three different values of k: 10, 50, and 250.

At $k = 10$, both Sample 1 and Sample 2 exhibit relatively high compression ratios of 4.99 and 5.08, respectively. This indicates that the images are highly compressed, with only 10 distinct colors retained. As a result, the file sizes are significantly reduced, but the image quality suffer due to the substantial reduction in color detail. The high compression ratios suggest that the algorithm has compressed the images effectively at this value of k.

When $k = 50$, the compression ratios decrease to 2.27 for Sample 1 and 2.40 for Sample 2. This means the images are less compressed compared to the $k = 10$ case, as more color details are retained, resulting in larger file sizes. Although the images are still compressed relative to their original size, the decrease in compression ratio indicates that less aggressive compression is applied, leading to improved image quality.

At $k = 250$, the compression ratios are further reduced to 1.04 for Sample 1 and 1.15 for Sample 2. This suggests that the images are much less compressed compared to the lower values of k, as 250 colors are used to represent each image. The file sizes are closer to the original size, but the image quality is significantly improved with minimal loss of detail. The lower compression ratios indicate that the compression is much less aggressive at this level of k, resulting in higher quality images with a trade-off of reduced compression.

IX. CONCLUSION

In this paper, the K-means clustering algorithm has been explored as an effective method for image compression. Through the application of the algorithm on two distinct types of images—a digital illustration and a photograph—we demonstrated the impact of different values of k on the compression ratio and image quality. The experimental results show a clear trade-off between compression efficiency and image quality. As the value of k increases, the number of colors retained in the image also increases, which leads to a decrease in the compression ratio but an improvement in image quality.

At lower values of k, such as 10, the images are highly compressed, leading to significant reductions in file size but with visible degradation in quality. As k increases, the compression becomes less aggressive, and the images maintain more of their original details, offering a better visual appearance. These results highlight the importance of choosing an optimal value for k based on the desired balance between file size and image fidelity.

Overall, K-means clustering provides a straightforward and efficient approach to image compression, with performance that can be fine-tuned by adjusting the number of clusters. However, future work could explore alternative clustering methods or hybrid approaches to further improve compression ratios while maintaining high image quality.

X. ACKNOWLEDGMENT

I would like to express my sincere gratitude to God Almighty for His guidance, who made the completion of this paper possible. I would also like to thank Dr. Ir. Rinaldi Munir, M.T., for his invaluable guidance and support during the IF2123 Aljabar Linier dan Geometri course. Additionally, the completion of this paper has been greatly supported by my dear family and friends, fellow students from the Teknik Informatika and Sistem dan Teknologi Informasi departments at ITB, Class of 2023.

REFERENCES

[1] S. Lloyd, ”Least squares quantization in PCM,” in IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129-137, March 1982, doi: 10.1109/TIT.1982.1056489.
[2] J. Pelleg, D. & Moore, A. (1999). ”Accelerating exact k -means algorithms with geometric reasoning”. Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. San Diego, California, United States: ACM Press. pp. 277–281. doi:10.1145/312129.312248. ISBN 9781581131437. S2CID 13907420.
[3] Hartigan, J. A. & Wong, M. A. (1979). Algorithm AS 136: A K-Means Clustering Algorithm. Applied Statistics, 28, 100–108. doi: 10.2307/2346830.
[4] Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., & Heming, J. (2022). K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. Information Sciences, 622, 178–210. doi: 10.1016/j.ins.2022.11.139.
[5] Anton, H., & Kaul, A. (2019). Elementary linear algebra. John Wiley & Sons.

APPENDIX A
COMPLETE IMPLEMENTATION OF THE COMPRESSION PROGRAM

The complete implementation of the program can be accessed at https://github.com/carasiae/compress_image

APPENDIX B
SOURCES OF SAMPLE IMAGES

1) Sample 1: A scene from a fictional visual novel starring Wikipe-tan, via Wikimedia. https://commons.wikimedia.org/wiki/File:Wikipe-tan_visual_novel_(Ren%27Py).png. Licensed under

the Creative Commons Attribution-Share Alike 3.0 Unported license.
2) Sample 2: MrBeast frontface 2021, via Wikimedia. https://commons.wikimedia.org/wiki/File:MrBeast_frontface_2021.png. Originally uploaded on YouTube under a CC license.

STATEMENT

I hereby declare that the paper I wrote is my own writing, not an adaptation, or translation of someone else's paper, and not plagiarized.

Bandung, 2 January 2025

Luqman

Muhammad Luqman Hakim