UNIVERSITY OF SCIENCE – VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY

FACULTY OF INFORMATION TECHNOLOGY – HIGH-QUALITY PROGRAM

PROJECT REPORT - APPLIED MATHEMATICS AND STATISTICS FOR INFORMATION TECHNOLOGY

COLOR COMPRESSION

Class: 23CLC05

Theory Instructor: Vũ Quốc Hoàng

Practical Instructors: Trần Thị Thảo Nhi, Nguyễn Ngọc Toàn

Student Name: Lê Hồng Ngọc

Student ID: 23127236

Ho Chi Minh City, June 22, 2025

Contents

I. Implementation Idea

1. Project overview

- In today's digital age, images play a significant role in conveying information. The pixel is a fundamental and essential concept in digital image processing, representing the smallest unit that makes up an image on electronic devices. A pixel can be represented in RGB format as (x, y, z), where x, y, and z are non-negative integers ranging from 0 to 255. Therefore, the total number of possible colors in an RGB image can reach up to $256^3 = 16.777.216$ colors. With a large number of distinct colors, image storage becomes increasingly costly in terms of memory and bandwidth.
- This project aims to address that problem by applying the **K-Means Clustering** algorithm to group similar colors in the image into a limited number of clusters. The image is then reconstructed using only the representative colors of these clusters, effectively reducing the number of colors while still preserving its original content.

2. Input and Output

- The input is a file path to a color image in .png, .jpg or .jpeg format provided by the user, along with an integer k representing the maximum number of colors allowed in the image. The user also specifies the method for initializing the centroids either randomly or by selecting from the existing pixels in the image.
- The output includes the original image with its number of distinct colors displayed, and a new version of the image reduced to k colors. The new image is shown on the screen and can be saved in either .png or .pdf format.

3. Implementation idea

- The program reduces the number of colors in an image to k colors (with k specified by the user), thereby simplifying and compressing the image to reduce storage costs while still preserving the origcinal visual content.
- To achieve color reduction, the **K-Means clustering algorithm** is used to group pixels with similar colors into clusters. Each clusters is represented by a central color, called a **centroid**. After clustering, each pixel in the image is replaced with the color of its corresponding centroid.
- In essence, we cluster the pixels in the 3-dimensional space $\mathbb{R}^3$ and select a representative color for each group. Hence, the problem is transformed into one of clustering vectors.

II. Implementation Detail

1. Importing relevant libraries

- **NumPy**: A numerical computing library used to manipulate image data in matrix form; it supports vector and matrix operations.
- **PIL (Python Imaging Library)**: An image processing library used to read, write, convert, and save images.
- **matplotlib**: A library for creating 2D charts from array data, used here to display images.

2. Helper functions

2.1. read_img()

Reads an image from a given path and converts it into a NumPy array for processing.

- Input parameter: *img_path* – a string representing the image file path.
- Using *Image.open()* from the PIL library to open the image at the specified *img_path.*
- If the image is not in RGB forrmat, it is converted to RGB.
- The image is then converted into a NumPy array to represent the image data. This NumPy array is a 2D array (3-dimensional) with a shape of (height x width x 3), where 3 corresponds to the 3 RGB color channels.

2.2. show_img()

Displays an image from a NumPy array using the *matplotlib* library (*plt*).

- Input parameter: *img_2d* – a 2D array representing the image.
- *plt.imshow()*: displays the image array *img_2d* in matphotlib's graphical window.
- *plt.axis('off')*: hides the coordinate axes, showing only the image.
- *plt.show()*: displays the image window prepared by *plt.imshow()* so the user can view it on the screen.

2.3. save_img()

Saves an image to a specified file path using *matplotlib.pyplot.imsave()* with automatic format detection based on the file extension.

- Input parameters: *img_2d* – a 2D array representing the image and *img_path* – a string representing the image path (including extension, e.g., .png, .jpg or .pdf).
- The image array is first converted to type uint8 to ensure pixel values are in the valid range (0–255).

- The *plt.imsave()* function is then used to write the image to the specified file. The file format (e.g., PNG, PDF) is automatically inferred from the extension in *img_path*.
- *vmin=0* and *vmax=255* are specified to preserve full RGB value range.

2.4. convert_img_to_1d()

Converts the image from 2D to 1D format to facilitate clustering using the K-Means algorithm.

- Input parameter: *img_2d* – a 2D array representing the image.
- Obtain the height and width of the image by retrieving the number of rows and columns of the array *img_2d*.
- Uses reshape() to convert the 2D array with shape (height x width x 3) into a 1D array with shape (height * width x 3), where each row of the new array represents a single pixel.

2.5. kmeans()

The function implements the K-Means algorithm to cluster image pixels into k groups (*k_clusters*), where each group is represented by a color called a *centroid*. A new image is then created using only these centroid colors, which helps reduce the image size while preserving the original visual content.

- Input parameters:
  - *img_1d*: a 1D array representing the image with shape (number of pixels x 3)
  - *k_clusters*: the number of clusters to form, i.e., the number of colors to reduce the image to.
  - *max_iter*: the maximum number of iterations for the algorithm.
  - *init_centroids*: the method for initializing centroids (random/in_pixels)
- Centroid initialization:
  - If random: generate *k_clusters* centroids with random RGB values in the range from 0 to 255.
  - If in_pixels: randomly select *k_clusters* pixels from the image as the initial centroids.
- Main loop (up to *max_iter* iterations):
  - Distance calculation: compute the square Euclidean distance between each pixel and each centroid across the 3 color channels. Leverages NumPy's powerful *broadcasting* mechanism to perform operations efficiently between arrays of different shapes.
  - *np.argmin()*: identifies the index of the nearest centroid for each pixel, assigning each pixel to the closest cluster.

        - Centroi update: for each cluster, calculate the mean of all assigned pixels and use this as the new centroid value.
        - *np.allclose()*: checks if the updated centroids are significantly different from the previous ones. If not, the algorithm stops early.
    - Return values:
        - *centroids*: an array containing the final centroids, representing the color of each cluster after pixel clustering.
        - *labels*: an array containing the cluster labels indicating which cluster each pixel belongs to.

2.6. generate_2d_img()

Reconstruct the 2D image from the result of the K-Means algorithm by replacing each pixel with the centroid it belongs to.

- Input parameters:
    - *img_2d_shape*: a 2D array representing the original image shape.
    - *centroids*: an array containing the representative color of each cluster.
    - *labels*: an array of labels indicating which centroid each pixel belongs to.
- Create a new 1D array by mapping each label in *labels* to its corresponding centroid – this is the magic of NumPy indexing.
- Reshape the resulting 1D array back to the original image shape using *reshape()*.
- Convert the NumPy array to *uint8* so that the color image can be displayed correctly.

2.7. count_unique_colors()

An additional helper function to count the number of distinct colors in the original image

- Input parameter: *img_2d* – a 2D array representing the original image.
- Convert the image from 2D to 1D format, where each row represents a single pixel with 3 color channels.
- *np.unique()*: find all unique pixels, which correspond to the different colors in the image.
- Return the number of rows in the *unique_colors* array, which is the total number of distinct colors in the image.

3. Main function

The *main()* function is the core of the color reduction program using the K-Means algorithm. It calls the helper functions defined above to form a complete processing pipeline and also provides a simple interactive interface for users.

- Requires the user to input the image file path and the desired number *k* of colors for color reduction.
- Counts the number of distinct colors in the original image.
- Reads the image and converts it from a 2D array to a 1D array for processing.
- Executes the K-Means algorithm to group similar colors into *k* clusters.
- Reconstructs a new image based on the result of the clustering.
- Displays both the original image and the color-reduced image so that users can easily observe the difference.
- Allows users to save the resulting image with a custom filename.

III. Results and Comment

1. Results

I selected the the image below as a sample for testing the K-Means algorithm implemented in the previous section.

Figure: Canola fields image

The image has the following properties:

Dimension	1280 x 853
Number of unique colors	172.105
Size	397 KB

The implemented K-Means algorithm was applied to compress the image by reducing the number of colors, using various values of the number of clusters: $k = 3$, 5, 7, 10, 20, 50, with the maximum number of iterations *max_iter* = 100.
The results for color reduction using 2 centroid initialization methods: *random* and *in_pixels* – are shown below:

- With $k = 3$ clusters:

Initialization Method	Output Image	Size	Best execution time
random		87 KB	2.7s
in_pixels		87 KB	3.4s

- With $k = 5$ clusters:

Initialization Method	Output Image	Size	Best execution time
random		93 KB	4.8s

in_pixels		93 KB	6.4s

- With $k = 7$ clusters:

Initialization Method	Output Image	Size	Best execution time
random		155 KB	12.6s
in_pixels		208 KB	13.2s

- With $k = 10$ clusters:

Initialization Method	Output Image	Size	Best execution time

random		178 KB	15.6s
in_pixels		274 KB	26.9s

- With $k = 20$ clusters:

Initialization Method	Output Image	Size	Best execution time
random		392 KB	1m 17s

in_pixels		508 KB	1m 32.2s

- With $k = 50$ clusters:

Initialization Method	Output Image	Size	Best execution time
random		612 KB	5m 02s
in_pixels		902 KB	5m 26.9s

2. Comments

Based on the experimental results, we have the following comments:

- Image quality:
  - The visual quality between the two centroid initialization methods: *random* and *in_pixels* shows no significant difference. However, in terms of fine details, the *in_pixels* method tends to perform slightly better, as the

initial centroids are sampled directly from the original image pixels, allowing for a more representative starting point in clustering.

  - With $k = 3$, the image loses a significant amount of detail, but the overall structure remains recognizable. With $k = 5$, $k = 7$, the image becomes more natural and easier to recognize. At $k = 10$ and $k = 20$, the color reduction is no longer clearly visible, and image details become much sharper. When k reaches 50 or more, the image becomes almost identical to the original.
  - ⇨ As k increases, more colors are retained, making the image more detailed and similar to the original. Fine details such as the flower field and the sky are more clearly restored. However, from $k = 20$ onward, improvements in visual quality begin to slow down compared to the increasing computational cost.

- Image size: For smaller values of k (up to 10), color reduction often leads to a smaller file size. However, for higher values of k (20 and above), the resulting image file can actually become larger than the original. The *in_pixels* centroid initialization method tends to produce larger files compared to the *random* method.
- Execution time: increases with k, especially for values of k greater than or equal to 20, where the runtime can reach several minutes. In practice, the *random* initialization method consistently runs faster than *in_pixels* method, mainly because it does not require selecting centroid values directly from the original image.

IV. References

1. Harris, C. R., et al. (2020). *NumPy*. https://numpy.org
2. Hunter, J. D., et al. (2024). *Matplotlib: Visualization with Python*. https://matplotlib.org
3. Clark, A. (2024). *Pillow (PIL Fork) Documentation* (v11.2.1). https://pillow.readthedocs.io
4. Wikipedia contributors. (2024, June). *K-means clustering*. Wikipedia. https://en.wikipedia.org/wiki/K-means_clustering
5. Huddar, M. (2020). *K Means Clustering Algorithm | K Means Solved Numerical Example Euclidean Distance* [Video]. YouTube. https://www.youtube.com/watch?v=4b5d3muPQmA
6. GeeksforGeeks. (2019). *K-means Clustering – Introduction*. https://www.geeksforgeeks.org/k-means-clustering-introduction
7. Project AI. (2023). *Sử dụng K-Means để thay đổi số lượng màu của ảnh*. https://projectai.com.vn/su-dung-k-means-de-giam-mau-anh
8. Pixabay. (n.d.). *Wilamowice Rapeseed Canola Fields - Free photo*. https://pixabay.com/photos/wilamowice-rapeseed-field-flowers-7585880
9. Uri2803. (2023). *CORLOR-COMPRESSION: K-Means color compression for images* [Source code]. GitHub. https://github.com/Uri2803/CORLOR-COMPRESSION

V. Acknowledgement

To complete this project, I would like to express my sincere gratitude to my friends, instructors, and AI tools that supported me throughout the implementation process:

- **Tran Cong Minh** – a student from the Applied Mathematics and Statistics class 23CLC05, Student ID 23127007 – who helped me implement the *save_img()* function and the step for finding the nearest centroid for each pixel (clustering step in the *kmeans()* function).
- **DeepSeek** – an AI tool that assisted me in debugging the *kmeans()* function and contributed to writing the helper function *count_unique_colors()*.
- **ChatGPT** – an AI tool that guided me in structuring this report and provided suggestions for writing the conclusion and evaluation sections.
- Finally, I would like to express my sincere thanks to the instructors of the Applied Mathematics and Statistics course: **Mr. Vu Quoc Hoang, Ms. Tran Thi Thao Nhi,** and **Mr. Nguyen Ngoc Toan**, who provided me with valuable knowledge that I was able to apply to this project.