

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



Báo cáo thực hành Color Compression

Môn học: Toán ứng dụng và thống kê cho CNTT

MTH00057-23CLC04

Sinh viên thực hiện
Nguyễn Trần Thiên An

Giảng viên hướng dẫn:
Nguyễn Đình Thúc
Trần Hà Sơn
Nguyễn Ngọc Toàn
Nguyễn Văn Quang Huy

Ngày 25 tháng 6 năm 2025

Mục lục

1	Thông tin cá nhân và đồ án	2
1.1	Thông tin cá nhân	2
1.2	Giới thiệu đồ án	2
2	Ý tưởng thực hiện	2
2.1	Input/Output	2
2.2	Mục tiêu chính	2
2.3	Ý tưởng giải quyết	3
3	Chi tiết thực hiện	4
3.1	Cấu trúc	4
3.2	Mô tả	4
4	Kết quả thực nghiệm và kết luận	7
4.1	Cấu hình máy tính	7
4.2	Kết quả thực nghiệm	8
4.2.1	Thực nghiệm 1	8
4.2.2	Thực nghiệm 2	10
4.3	Kết luận	12
4.3.1	Hiệu suất thời gian của hai phương pháp khởi tạo	12
4.3.2	Chất lượng ảnh sau phân cụm màu	13
4.3.3	Nhận xét về sự khác biệt thời gian chạy giữa hai ảnh	13
4.3.4	Kết luận chung	14
5	Tài liệu	14
6	Acknowledgement	15

1 Thông tin cá nhân và đồ án

1.1 Thông tin cá nhân

Lớp	MSSV	Tên
23CLC04	23127315	Nguyễn Trần Thiên An

Bảng 1: Thông tin của cá nhân

1.2 Giới thiệu đồ án

Viết chương trình Python để giảm số lượng màu của ảnh bằng thuật toán **K-Means**. Chương trình phải đọc, hiển thị, lưu ảnh (ít nhất định dạng png và pdf), chuyển ảnh từ 2D sang 1D, gom nhóm màu bằng K-Means và tạo ảnh mới từ các màu trung tâm. Hàm main cho phép nhập tên file ảnh qua *input()*. Chỉ dùng các thư viện **NumPy**, **PIL** và **matplotlib**.

2 Ý tưởng thực hiện

Đồ án nhằm xây dựng một hệ thống nén ảnh dựa trên thuật toán **K-Means clustering** trong không gian màu RGB. Thay vì lưu toàn bộ màu gốc của ảnh (có thể chứa hàng triệu màu khác nhau), thuật toán sẽ gom các màu tương đồng về một số cụm màu chính (centroid), từ đó thay mỗi pixel bằng màu cụm tương ứng. Cách tiếp cận này giúp giảm kích thước ảnh mà vẫn giữ được đặc trưng thị giác ban đầu.

2.1 Input/Output

Input: một ảnh màu bất kỳ (định dạng JPG, PNG, v.v.)

Output: ảnh sau khi được nén màu bằng K-Means, với số lượng màu giảm còn k cụm chính (do người dùng nhập).

2.2 Mục tiêu chính

- Hiểu rõ quy trình nội tại của thuật toán **K-Means** thông qua việc tự triển khai toàn bộ các hàm (**KHÔNG** dùng thư viện có sẵn như `sklearn`).
- Ứng dụng vào xử lý ảnh để thực hiện bài toán **giảm số lượng màu** mà vẫn đảm bảo độ trung thực thị giác.
- Cho phép lựa chọn phương pháp khởi tạo cụm và số vòng lặp tối đa để kiểm soát chất lượng và hiệu suất nén ảnh.

2.3 Ý tưởng giải quyết

Bài toán đặt ra là làm sao để nén ảnh bằng cách **giảm số lượng màu sắc** xuất hiện trong ảnh gốc mà vẫn duy trì được chất lượng thị giác. Ý tưởng cốt lõi là sử dụng thuật toán **K-Means Clustering** để nhóm các pixel trong ảnh (biểu diễn trong không gian màu RGB) thành các cụm màu chính (centroids). Sau khi phân cụm, mỗi pixel trong ảnh sẽ được ánh xạ về màu của cụm gần nhất, từ đó tạo thành ảnh mới với số lượng màu ít hơn ban đầu.

Ý tưởng thực hiện gồm các bước:

- Chuyển ảnh từ định dạng ban đầu (ma trận 3 chiều) sang dạng danh sách (2 chiều), trong đó mỗi hàng là một vector màu RGB.
- Khởi tạo k điểm màu ngẫu nhiên làm các centroid ban đầu. Có thể khởi tạo hoàn toàn ngẫu nhiên hoặc chọn từ các pixel có sẵn.
- Áp dụng thuật toán K-Means: lặp lại quá trình gán mỗi pixel vào cụm gần nhất (dựa trên khoảng cách Euclidean), sau đó cập nhật centroid bằng trung bình các pixel trong cụm.
- Lặp lại cho đến khi các centroid hội tụ (không thay đổi đáng kể) hoặc đạt số vòng lặp tối đa.
- Dựng lại ảnh mới bằng cách thay mỗi pixel bằng màu của cụm mà nó thuộc về.



Hình 1: Biểu đồ minh họa ý tưởng thực hiện

Phương pháp này giúp giảm số lượng màu (ví dụ từ hàng triệu màu xuống chỉ còn vài chục cụm hay vài cụm) phục vụ cho các mục đích khác trong quá trình xử lý ảnh.

3 Chi tiết thực hiện

3.1 Cấu trúc

STT	Hàm	Chức năng
1	<code>read_img()</code>	Đọc ảnh và chuyển thành mảng NumPy dạng RGB.
2	<code>show_img()</code>	Hiển thị ảnh bằng thư viện matplotlib.
3	<code>save_img()</code>	Lưu ảnh dưới định dạng mong muốn (PNG/PDF).
4	<code>convert_img_to_1d()</code>	Chuyển đổi ảnh 2D thành mảng 1D để phục vụ phân cụm.
5	<code>initialize_centroids()</code>	Khởi tạo centroids theo hai cách: <i>random</i> hoặc <i>in_pixels</i> .
6	<code>calculate_distances()</code>	Tính toán khoảng cách từ mỗi pixel đến các centroid.
7	<code>get_labels()</code>	Gán nhãn (cluster) cho mỗi pixel với centroid gần nhất.
8	<code>update_centroids()</code>	Cập nhật các centroid dựa trên <i>mean</i> điểm trong mỗi cụm.
9	<code>is_converged()</code>	Đọc ảnh và chuyển thành mảng NumPy dạng RGB.
10	<code>kmeans()</code>	Thuật toán K-Means chính, gọi các hàm trên để phân cụm.
11	<code>generate_2d_img()</code>	Dựng lại ảnh từ <i>labels</i> và giá trị <i>centroids</i> .
12	<code>main()</code>	<i>Hàm chính:</i> gọi các hàm, xử lý nhập xuất, hiển thị kết quả.

Bảng 2: Cấu trúc sơ bộ các hàm

3.2 Mô tả

1. `read_img(img_path)`

Hàm đọc ảnh từ đường dẫn và ép về định dạng RGB bằng `.convert('RGB')` để chuẩn hóa đầu vào. Kết quả được chuyển thành mảng NumPy kiểu `uint8` với 3 kênh màu, làm đầu vào cho các bước xử lý tiếp theo.

2. `show_img(img_2d)`

Hàm dùng `matplotlib` để hiển thị ảnh dưới dạng RGB. Được sử dụng sau khi ảnh đã được phân cụm và tái dựng để trực quan hóa kết quả.

3. `save_img(img_2d, img_path)`

Hàm lưu ảnh đã xử lý ra tệp với định dạng do người dùng chỉ định (PNG, JPG, PDF, v.v.), sử dụng `PIL.Image.fromarray()`.

4. `convert_img_to_1d(img_2d)`

Sau đọc ảnh, hàm này chuyển đổi ma trận ảnh 2D có kích thước (`height, width, 3`) thành mảng 1D dạng (`height * width, 3`) bằng `reshape()`. Việc này là cần thiết để thực hiện các phép toán vector hóa trong K-Means và tối ưu hóa hiệu năng.

5. initialize_centroids(img_1d, k_clusters, init_centroids)

Hàm khởi tạo k centroid đầu tiên – thành phần trọng yếu trong thuật toán K-Means. Có hai tùy chọn:

- 'random': sinh ngẫu nhiên centroid trong không gian RGB [0, 255] bằng `np.random.randint`.
- 'in_pixels': chọn ngẫu nhiên từ chính các pixel thật của ảnh bằng `np.random.choice()`.

Việc lựa chọn cách khởi tạo sẽ ảnh hưởng đến tốc độ hội tụ và chất lượng phân cụm. Hàm này được gọi đầu tiên trong `kmeans()`.

6. calculate_distances(img_1d, centroids, img_squared_norm)

Đây là hàm phụ trợ giúp tối ưu tính khoảng cách Euclidean giữa mỗi pixel và mỗi centroid bằng công thức:

$$\|x - c\| = \sqrt{\|x - c\|^2} = \sqrt{\|x\|^2 - 2x^T c + \|c\|^2}$$

Cài đặt sử dụng `dot_product` giữa pixel và centroid, nhằm tránh tính toán lặp và tăng tốc xử lý. Hàm này được gọi bên trong `get_labels()`.

7. get_labels(img_1d, centroids, img_squared_norm)

Hàm gán nhãn cho mỗi pixel bằng cách chọn chỉ số centroid gần nhất (tức có khoảng cách nhỏ nhất). Điều này được thực hiện bằng `np.argmin()` trên kết quả khoảng cách. Đây là bước chính trong mỗi vòng lặp của `kmeans()` để xác định từng pixel thuộc về cụm nào.

8. update_centroids(img_1d, k_clusters, labels, old_centroids)

Với các nhãn đã gán, centroid mới được tính bằng trung bình các pixel thuộc cụm đó. Nếu một cụm rỗng (không có pixel nào), chương trình sẽ giữ lại centroid cũ hoặc chọn lại ngẫu nhiên để tránh lỗi. Quá trình này dùng vòng `for` duyệt qua từng cụm.

9. is_converged(centroids, new_centroids)

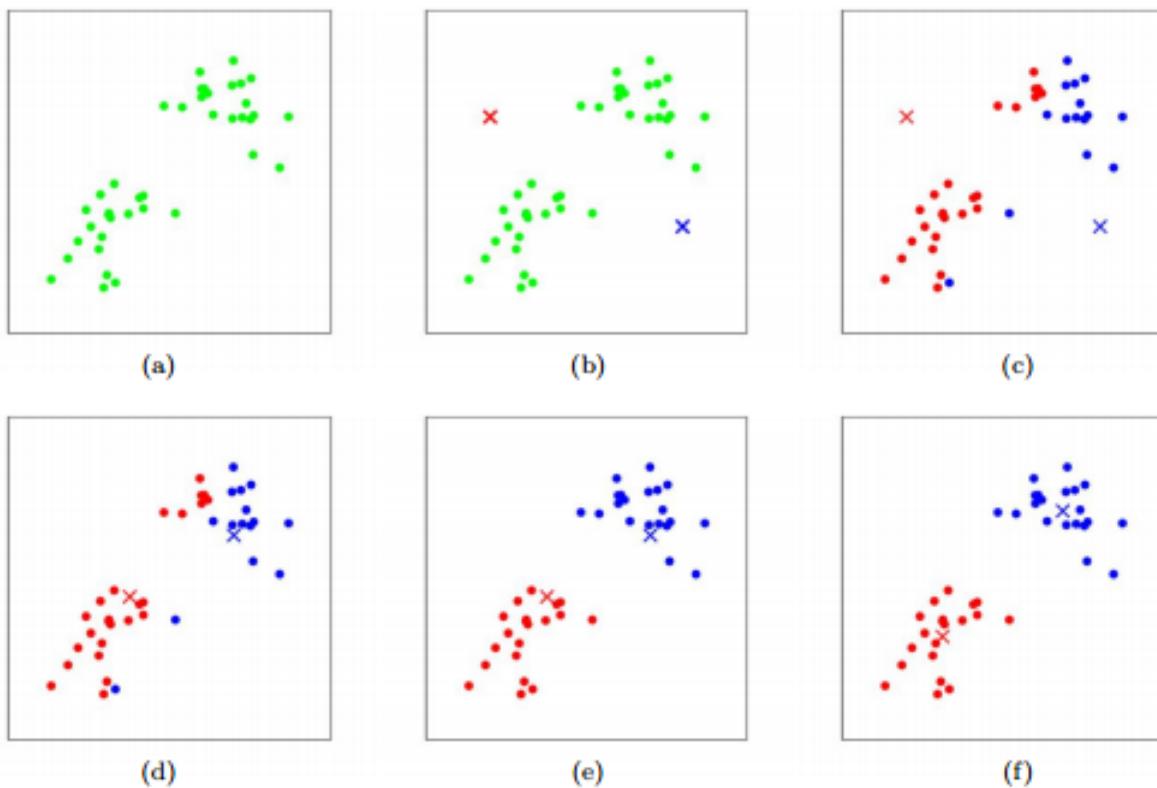
Hàm kiểm tra hội tụ của thuật toán bằng cách đo sai khác tuyệt đối lớn nhất giữa các centroid cũ và mới. Nếu sai khác nhỏ hơn 10^{-6} , hàm trả về `True` và vòng lặp trong `kmeans()` dừng sớm. Việc này giúp giảm chi phí tính toán khi thuật toán đã ổn định.

10. `kmeans(img_1d, k_clusters, max_iter, init_centroids='random')`

Đây là thuật toán chính thực hiện phân cụm màu cho ảnh, gồm các bước:

- Gọi `initialize_centroids()` để chọn k centroid ban đầu.
- Tính khoảng cách và nhãn bằng `get_labels()`.
- Cập nhật centroid qua `update_centroids()`.
- Dừng nếu hội tụ được phát hiện bằng `is_converged()`.

Kết quả trả về là mảng `centroids` và `labels`, phục vụ bước tái dựng ảnh sau này.



Hình 2: Biểu đồ minh họa thuật toán K-Means

11. `generate_2d_img(img_2d_shape, centroids, labels)`

Hàm này tái dựng lại ảnh từ các nhãn đã gán bằng cách thay mỗi pixel bằng màu của centroid tương ứng: `centroids[labels]`. Kết quả reshape lại thành kích thước ban đầu (`height, width, 3`) và được ép kiểu về `uint8`. Hàm thường được dùng sau khi đã phân cụm bằng `kmeans()`.

12. main()

Hàm `main()` (có thể tự định nghĩa thêm) có thể tổng hợp toàn bộ các bước xử lý:

- Đọc ảnh bằng `read_img()`.
- Chuyển đổi sang 1D bằng `convert_img_to_1d()`.
- Gọi `kmeans()` để phân cụm màu.
- Tái dựng ảnh bằng `generate_2d_img()`.
- Hiển thị ảnh bằng `show_img()` và lưu ảnh bằng `save_img()`.

4 Kết quả thực nghiệm và kết luận

4.1 Cấu hình máy tính

Thông tin máy tính sử dụng cho thử .

- Loại: Laptop
- Tên: MacBook Pro (13-inch, M1, 2020)
- Chip: Apple M1 chip
 - 8-core CPU with 4 performance cores and 4 efficiency cores
 - 8-core GPU
 - 16-core Neural Engine
- RAM: 16GB

4.2 Kết quả thực nghiệm

4.2.1 Thực nghiệm 1

Thời gian chạy của riêng thuật toán **K-Means**, hàm `kmeans()`:

STT	Kiểu	Thời gian ước tính
1	random với $k = 3$	$\approx 12s.$
2	random với $k = 5$	$\approx 15s.$
3	random với $k = 7$	$\approx 31s.$
4	in_pixels với $k = 3$	$\approx 20s.$
5	in_pixels với $k = 5$	$\approx 17s.$
6	in_pixels với $k = 7$	$\approx 25s.$

Bảng 3: Thời gian chạy hàm `kmeans()`



Hình 3: Ảnh gốc 4K với 137975 màu



Hình 4: **random** với $k = 3$



Hình 5: **in_pixels** với $k = 3$



Hình 6: **random** với $k = 5$



Hình 7: **in_pixels** với $k = 5$



Hình 8: **random** với $k = 7$



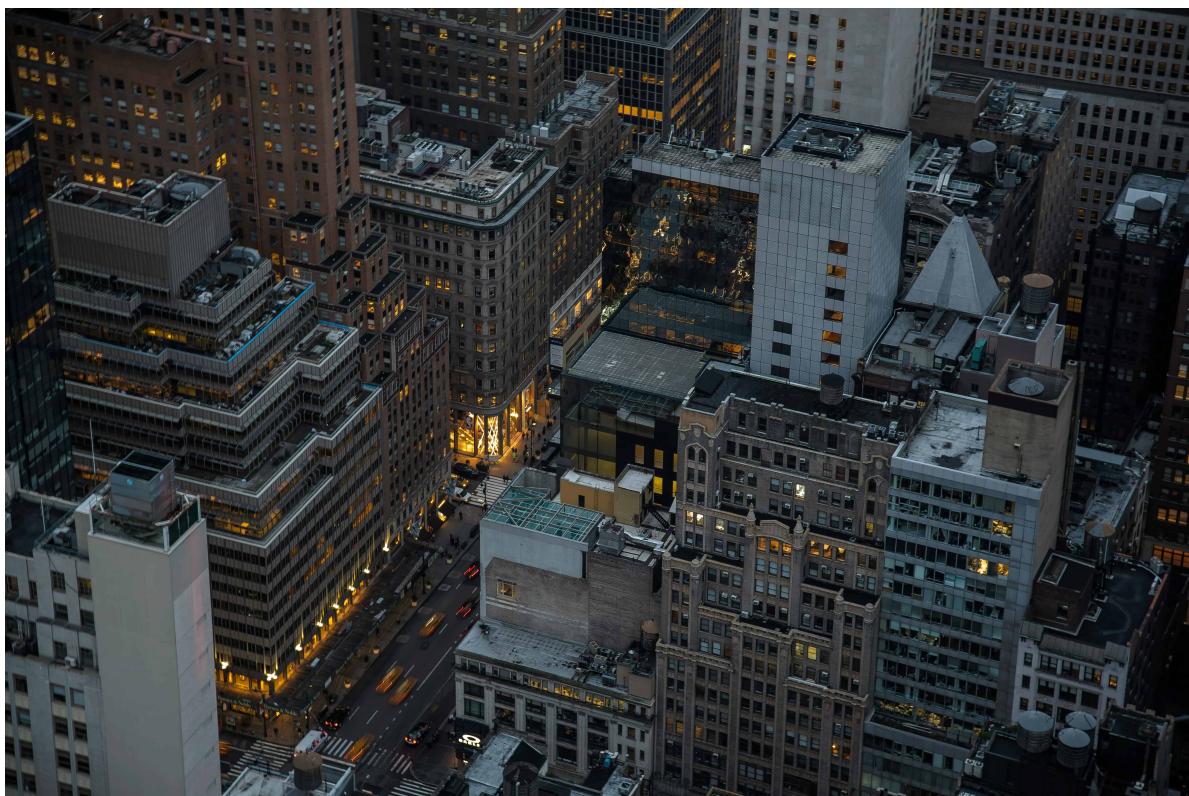
Hình 9: **in_pixels** với $k = 7$

4.2.2 Thực nghiệm 2

Thời gian chạy của riêng thuật toán **K-Means**, hàm `kmeans()`:

STT	Kiểu	Thời gian ước tính
1	<code>random</code> với $k = 3$	$\approx 52s.$
2	<code>random</code> với $k = 5$	$\approx 124s.$
3	<code>random</code> với $k = 7$	$\approx 82s.$
4	<code>in_pixels</code> với $k = 3$	$\approx 44s.$
5	<code>in_pixels</code> với $k = 5$	$\approx 125s.$
6	<code>in_pixels</code> với $k = 7$	$\approx 130s.$

Bảng 4: Thời gian chạy hàm `kmeans()`



Hình 10: Ảnh gốc 4K với 262284 màu



Hình 11: **random** với $k = 3$



Hình 12: **in_pixels** với $k = 3$



Hình 13: **random** với $k = 5$



Hình 14: **in_pixels** với $k = 5$



Hình 15: **random** với $k = 7$



Hình 16: **in_pixels** với $k = 7$

4.3 Kết luận

Phần này trình bày kết quả thực nghiệm và nhận xét khi áp dụng thuật toán **K-Means** (tự cài bằng NumPy) cho hai ảnh màu độ phân giải 4K, sử dụng hai phương pháp khởi tạo centroid khác nhau.

4.3.1 Hiệu suất thời gian của hai phương pháp khởi tạo

Hai phương pháp khởi tạo được sử dụng:

- **random**: khởi tạo centroid ngẫu nhiên trong không gian RGB.
- **in_pixels**: khởi tạo centroid từ tập điểm ảnh thực tế (chọn ngẫu nhiên từ dữ liệu).

Thực nghiệm 1 (ảnh thiên nhiên):

- Số màu ban đầu: **137,975** màu
- Thời gian trung bình:
 - **random**: dao động từ **12s** đến **31s**
 - **in_pixels**: dao động từ **17s** đến **25s**
- Quan sát:
 - Với $k = 3$: **random** nhanh hơn.
 - Với $k = 5$: **in_pixels** nhanh hơn.
 - Với $k = 7$: **random** chậm nhất (**31s**).

Thực nghiệm 2 (ảnh thành phố ban đêm):

- Số màu ban đầu: **262,284** màu
- Thời gian trung bình:
 - **random**: từ **52s** đến **124s**
 - **in_pixels**: từ **44s** đến **130s**
- Quan sát:
 - **in_pixels** nhanh hơn ở $k = 3$
 - Nhưng chậm hơn rõ rệt ở $k = 5$ và $k = 7$

Tạm kết: Không có phương pháp nào luôn vượt trội. **random** có thể nhanh hơn với k nhỏ và ảnh ít chi tiết, nhưng **in_pixels** thường ổn định hơn trên ảnh độ phân giải cao và phức tạp.

4.3.2 Chất lượng ảnh sau phân cụm màu

Thực nghiệm 1 (ảnh phong cảnh):

- Với $k = 3$:
 - `random`: ảnh mất quá nhiều chi tiết, khó phân biệt nền trời và vùng núi.
 - `in_pixels`: tái hiện màu tốt hơn, phân lớp rõ giữa các tầng mây – núi.
- Với $k = 5$ và $k = 7$: cả hai phương pháp đều cải thiện thị giác rõ rệt. Đặc biệt, `in_pixels` ở $k = 7$ cho kết quả gần với ảnh gốc nhất.

Thực nghiệm 2 (ảnh thành phố ban đêm):

- Với $k = 3$: cả hai phương pháp đều khiến ảnh bị giảm chi tiết đáng kể.
- Với $k = 5$: `in_pixels` thể hiện chi tiết tốt hơn 1 ít.
- Với $k = 7$: hầu như không có sự khác biệt quá nhiều giữa `in_pixels` và `random` khi cả 2 đều giữ được tông màu vàng sáng của đèn, tạo độ tương phản rõ.

Tạm kết: Khi số cụm k tăng, chất lượng ảnh tăng rõ rệt. Càng tăng k cho kết quả thị giác ổn định hơn, đặc biệt trên ảnh có nhiều chi tiết nhỏ và phổ màu đa dạng.

4.3.3 Nhận xét về sự khác biệt thời gian chạy giữa hai ảnh

Dựa trên kết quả thực nghiệm, thời gian chạy thuật toán K-Means trên ảnh 2 (ảnh thành phố) **dài hơn** đáng kể so với ảnh 1 (ảnh phong cảnh). Có thể giải thích hiện tượng này qua các yếu tố sau:

1. Số lượng màu khác nhau (số điểm cần phân cụm):

- Ảnh 1 có khoảng **137975** màu khác nhau.
- Ảnh 2 có tới **262284** màu — gần gấp đôi ảnh 1.

Do thuật toán phải tính khoảng cách từ mỗi điểm ảnh đến các centroid ở mỗi vòng lặp, số màu lớn hơn làm tăng đáng kể số phép tính, dẫn đến thời gian chạy dài hơn.

2. Độ phức tạp nội dung ảnh:

- Ảnh 1 là cảnh thiên nhiên, màu sắc chuyển tiếp mượt, ít chi tiết.
- Ảnh 2 là ảnh thành phố, chứa nhiều chi tiết nhỏ, cạnh sắc nét, ánh sáng phức tạp.

Mức độ chi tiết cao làm tăng số cụm cần thiết để tái hiện ảnh một cách chính xác, đồng thời gây khó khăn cho việc hội tụ nhanh của thuật toán.

3. Phân bố không gian màu RGB:

Ảnh phong cảnh dễ gom cụm hơn, trong khi ảnh thành phố có các màu sắc phân bố rời rạc và phức tạp (ánh sáng, cửa sổ, bóng tối...), khiến thuật toán cần nhiều vòng lặp để hội tụ.

Kết luận: Ảnh 2 có số lượng màu lớn hơn, cấu trúc màu phức tạp hơn và độ chi tiết cao hơn, là các nguyên nhân chính dẫn đến việc thuật toán K-Means phải thực hiện nhiều vòng lặp hơn để cập nhật centroid chính xác, đặc biệt khi k lớn (5 hoặc 7), khiến thời gian chạy thuật toán K-Means dài hơn đáng kể so với ảnh 1.

4.3.4 Kết luận chung

1. **Hiệu quả phân cụm:** K-Means có thể giảm đáng kể số lượng màu mà vẫn giữ được cấu trúc ảnh. Phương pháp `in_pixels` thường cho kết quả thị giác tự nhiên hơn.
2. **Thời gian tính toán:**
 - Phương pháp `random` nhanh hơn ở ảnh đơn giản hoặc k nhỏ, nhưng dễ lặp lại thuộc vào khởi tạo. `in_pixels` ổn định hơn nhưng không phải lúc nào cũng nhanh hơn.
 - Thời gian thực thi của thuật toán chịu ảnh hưởng mạnh bởi đặc điểm của ảnh đầu vào.
 - Ảnh 1 (thiên nhiên) có số màu ít hơn và màu sắc phân bố mượt, giúp thuật toán hội tụ nhanh.
 - Ảnh 2 (đô thị ban đêm) có gần gấp đôi số màu, nhiều chi tiết nhỏ và màu sắc rực rỡ hơn trong không gian RGB, làm tăng khối lượng tính toán và số vòng lặp hội tụ.
 - Vì vậy, ảnh 2 có thời gian chạy lâu hơn đáng kể so với ảnh 1, bất kể phương pháp khởi tạo nào được sử dụng.
3. **Tổng quan:** Việc hiện thực thuật toán K-Means bằng NumPy mà KHÔNG dùng thư viện sẵn có như `scikit-learn` nhưng vẫn đạt hiệu quả cả về tốc độ và thị giác.

5 Tài liệu

- [1] Mahesh Huddar, [K Means Clustering Algorithm | K Means Solved Numerical Example Euclidean Distance by Mahesh Huddar](#), YouTube, 2023. [14/06/2025].
- [2] NeuralNine, [K-Means Clustering From Scratch in Python \(Mathematical\)](#), YouTube, 2021. [15/06/2025].
- [3] AssemblyAI, [How to implement K-Means from scratch with Python](#) YouTube, 2022. [15/06/2025].
- [4] V. H. Tiệp, [K-means Clustering](#), Machine Learning Cơ Bản, Jan. 2017. [15/06/2025].
- [5] Numpy Developers, [numpy.random.randint — NumPy v2.1 Manual](#). [15/06/2025].
- [6] Numpy Developers, [numpy.random.choice — NumPy v2.1 Manual](#). [15/06/2025].
- [7] Numpy Developers, [numpy.argmax — NumPy v2.1 Manual](#). [15/06/2025].
- [8] GeeksforGeeks, [How to Convert Images to NumPy Array](#). [15/06/2025].
- [9] Programiz, [NumPy Fancy Indexing](#). [18/06/2025].
- [10] Chris Piech, [K Means](#), Stanford. [25/06/2025].
- [11] Simon Berger, [Silhouette Of Mountains](#), Pexels. [24/06/2025].
- [12] Peter Olex, [Aerial View Of City Buildings](#), Pexels. [25/06/2025].

6 Acknowledgement

Trước hết, em xin chân thành cảm ơn thầy **Nguyễn Ngọc Toàn** đã tận tình giảng giải thuật toán trên lớp, giúp em hiểu rõ hơn về **K-Means Clustering**, cũng như hỗ trợ cả lớp trong việc lập trình Python tại các buổi Lab. Em cũng xin gửi lời cảm ơn đến bạn **Trần Phụng Định** đã gợi ý cho em áp dụng công thức tính khoảng cách giữa pixel và centroid, góp phần giúp thuật toán chạy hiệu quả và nhanh hơn. Bên cạnh đó, em đã nhận được sự hỗ trợ từ các công cụ như **ChatGPT** và **Gemini** trong việc tìm kiếm tài liệu liên quan đến thuật toán K-Means. Các công cụ này cũng hỗ trợ em giải thích cú pháp Python thông qua các ví dụ cụ thể khi em gặp khó khăn trong quá trình tham khảo tài liệu, đồng thời giúp em chỉnh sửa câu văn và văn phong trong các phần **2.3**, **3.2** và **4.3** của báo cáo.