

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



COLOR COMPRESSION

Môn học: Toán Ứng Dụng Và Thống Kê Cho Công Nghệ Thông Tin

MTH00057-23CLC08

Sinh viên thực hiện:

Nguyễn Phương Thảo
23127306
23CLC08

Giảng viên:

Vũ Quốc Hoàng
Trần Thị Thảo Nhi
Nguyễn Văn Quang Huy

Ngày 23 tháng 6 năm 2025

Mục lục

1	Ý tưởng thực hiện	2
1.1	Tổng quan về đồ án	2
1.2	Đầu vào, đầu ra của thuật toán	2
1.3	Mục tiêu	2
1.4	Ý tưởng giải quyết	2
2	Chi tiết thực hiện	3
2.1	Cấu trúc chương trình	3
2.2	Mô tả một số hàm chính	5
2.2.1	<code>kmeans(img_1d, k_clusters, max_iter, init_centroids)</code>	5
2.2.2	<code>generate_2d_img(img_2d_shape, centroids, labels)</code>	7
2.2.3	<code>compress_image(image_path, k, max_iter, init_method)</code>	7
2.3	Cải tiến bằng thuật toán K-Means++	7
2.4	Cải tiến bằng thuật toán Mini-Batch K-Means	8
3	Kết quả thực nghiệm	9
3.1	Ảnh phân cụm với các giá trị k khác nhau (khởi tạo ngẫu nhiên)	9
3.2	So sánh Mini-Batch K-Means và K-Means thường	10
3.3	So sánh các phương pháp khởi tạo centroid	11
4	Tài liệu tham khảo	13
5	Acknowledgement	14

1 Ý tưởng thực hiện

1.1 Tổng quan về đồ án

Trong thời đại kỹ thuật số hiện nay, nhu cầu lưu trữ và truyền tải hình ảnh một cách hiệu quả ngày càng tăng cao. Một trong những hướng tiếp cận phổ biến để giảm dung lượng ảnh mà vẫn giữ được chất lượng hiển thị là giảm số lượng màu sắc trong ảnh. Dựa trên ý tưởng này, đồ án thực hiện bài toán **nén ảnh màu** bằng thuật toán **K-Means clustering** – một phương pháp phân cụm không giám sát trong học máy.

1.2 Đầu vào, đầu ra của thuật toán

Đầu vào: một ảnh RGB bất kỳ và số cụm K do người dùng chỉ định. Ảnh được chuyển thành mảng kích thước $(n, 3)$ với n là số pixel.

Đầu ra: ảnh mới chỉ còn K màu, mỗi pixel mang màu của một centroid. Ảnh đầu ra có dung lượng nhỏ hơn và vẫn giữ được nội dung chính.

1.3 Mục tiêu

Mục tiêu của đồ án là xây dựng chương trình giảm số lượng màu trong ảnh bằng thuật toán K-Means, từ đó tạo ra ảnh đầu ra gọn nhẹ hơn nhưng vẫn giữ được nội dung thị giác chính.

Ngoài ra, đồ án còn hướng đến việc:

- Hiểu và cài đặt thuật toán K-Means.
- Thủ nghiệm với nhiều giá trị K khác nhau để đánh giá chất lượng ảnh sau nén.
- So sánh các phương pháp khởi tạo tâm cụm và đề xuất cải tiến bằng K-Means++ nhằm tăng hiệu quả và giảm thời gian hội tụ.

1.4 Ý tưởng giải quyết

Bài toán sử dụng thuật toán K-Means để gom các pixel trong ảnh thành K cụm màu, từ đó giảm số lượng màu. Ảnh đầu vào được chuyển thành mảng 2 chiều, mỗi pixel là một điểm RGB. Thuật toán lặp lại hai bước: (1) gán mỗi pixel vào cụm gần nhất, (2) cập nhật centroid bằng trung bình cụm.

Khi các centroid hội tụ hoặc đạt số vòng lặp tối đa, mỗi pixel được thay bằng màu của centroid tương ứng để tạo ra ảnh nén màu.

Trong quá trình thực hiện, em nhận thấy cách khởi tạo centroid ảnh hưởng đáng kể đến tốc độ hội tụ và chất lượng cụm. Nếu khởi tạo ngẫu nhiên, các centroid ban đầu có thể phân bố không đều, dẫn đến phân cụm kém. Để cải thiện, em sử dụng phương pháp K-Means++, giúp chọn centroid ban đầu một cách có chủ đích hơn, các centroid được phân bố đồng đều, từ đó tăng độ ổn định và giảm số vòng lặp.

Ngoài ra, để tăng tốc độ xử lý đối với ảnh có số lượng điểm ảnh lớn, em đã triển khai thêm phiên bản Mini-Batch K-Means. Thuật toán này sử dụng các mini-batch thay vì toàn bộ tập dữ liệu trong mỗi vòng lặp cập nhật centroid. Nhờ đó, thời gian tính toán được rút ngắn đáng kể mà chất lượng kết quả vẫn được đảm bảo ở mức chấp nhận được. Kỹ thuật này đặc biệt hiệu quả khi xử lý ảnh lớn hoặc khi cần thử nghiệm với nhiều giá trị k khác nhau.

2 Chi tiết thực hiện

2.1 Cấu trúc chương trình

Chương trình gồm các hàm chính được chia thành 4 nhóm theo chức năng:

- **Xử lý ảnh:**

- `read_img(img_path)`: Đọc ảnh từ đường dẫn, chuyển về không gian màu RGB và trả về ảnh dưới dạng mảng NumPy 3 chiều.
- `show_img(img_2s)`: Hiển thị ảnh 2D trên màn hình bằng `matplotlib`
- `save_img(img, path)`: Lưu ảnh ra tệp tin sau khi xử lý.
- `convert_img_to_1d(img_2d)`: Chuyển đổi ảnh từ dạng 2D ($H, W, 3$) sang dạng 1D ($H \times W, 3$) để thuận tiện cho việc xử lý từng pixel riêng lẻ trong thuật toán K-Means.

- **Thuật toán K-Means**

- `kmeans(img_1d, k_clusters, max_iter, init_centroids)`: Triển khai thuật toán K-Means.

- **initialize_centroids(img_1d, k_clusters, init_centroids):** Khởi tạo tâm cụm theo các phương pháp: random, in_pixels, kmeans++.
- **init_kmeans_plus_plus(img_1d, k_clusters):** Khởi tạo tâm cụm theo thuật toán K-Means++.
- **assign_clusters(img_1d, centroids):** Gán pixel vào cụm gần nhất.
- **update_centroids(img_1d, labels, k_clusters):** Tính lại centroid từ các điểm trong cụm.
- **generate_2d_img(img_2d_shape, centroids, labels):** Tái tạo ảnh 2D từ nhãn phân cụm và centroid.
- **compress_image(image_path, k_clusters, max_iter, method):** Thực hiện toàn bộ quy trình nén ảnh bằng K-Means.

- **Thuật toán Mini-Batch K-Means**

- **mini_batch_kmeans(img_1d, k_clusters, max_iter, batch_size, init_centroids):** Biến thể của K-Means sử dụng mini-batch để cập nhật centroid theo từng nhóm điểm ảnh nhỏ, giúp giảm thời gian xử lý.
- **compress_image_mini_batch(image_path, k_clusters, max_iter, init_method):** Thực hiện nén ảnh sử dụng thuật toán Mini-Batch K-Means.

- **Kiểm tra và phân tích các kết quả chạy thử**

- **test_k_values(image_path, k_list, max_iter, init_centroids):** Hàm dùng để kiểm tra và phân tích kết quả phân cụm khi thay đổi số lượng cụm k . Cụ thể:
 - * Chạy thuật toán K-Means với các giá trị k khác nhau trong danh sách k_list .
 - * Do và lưu lại thời gian xử lý tương ứng với mỗi giá trị k .
 - * Tái tạo và hiển thị ảnh phân cụm ứng với từng k , cùng với ảnh gốc để dễ dàng so sánh trực quan.
- **test_kmeans_vs_minibatch_multiple_k(...):** So sánh chất lượng ảnh và thời gian xử lý giữa thuật toán K-Means truyền thống và Mini-Batch K-Means với nhiều giá trị k .

- **Chạy chương trình**

- **main()**: Hàm chính cho phép nhập thông số từ người dùng và thực hiện nén ảnh bằng K-Means.
- **main_mini_batch()**: Hàm chính cho phép nhập thông số từ người dùng và thực hiện nén ảnh bằng Mini-Batch K-Means.

2.2 Mô tả một số hàm chính

2.2.1 kmeans(img_1d, k_clusters, max_iter, init_centroids)

Hàm này thực hiện thuật toán K-Means trên dữ liệu ảnh 1 chiều.

Đầu vào:

- **img_1d**: mảng pixel dạng (`num_pixels, num_channels`).
- **k_clusters**: số cụm màu K cần phân.
- **max_iter**: số vòng lặp tối đa.
- **init_centroids**: phương pháp khởi tạo tâm cụm (`random, in_pixels, kmeans++`).

Cách hoạt động:

1. Sử dụng hàm `initialize_centroids(...)` để khởi tạo K centroid theo phương pháp được chỉ định.

- `random`: sinh ngẫu nhiên các vector RGB trong khoảng $[0, 255]$.
- `in_pixels`: chọn ngẫu nhiên K pixel từ ảnh gốc.
- `kmeans++`: gọi hàm `init_kmeans_plus_plus(...)` để chọn centroid phân bố đều.

2. Trong mỗi vòng lặp:

- `assign_clusters(img_1d, centroids)`: Hàm này gán mỗi pixel vào cụm gần nhất bằng cách sử dụng kỹ thuật **vector hóa** để tăng tốc độ xử lý thay vì dùng vòng lặp thủ công:

- `img_1d[:, np.newaxis]` chuyển shape của ảnh từ $(\text{num_pixels}, 3)$ thành $(\text{num_pixels}, 1, 3)$ nhằm chuẩn bị cho phép broadcast.
 - `centroids[np.newaxis, :]` có shape là $(1, k, 3)$, cũng để broadcast với mảng pixel.
 - Khi thực hiện phép trừ giữa hai mảng này, NumPy tạo ra một mảng hiệu có shape $(\text{num_pixels}, k, 3)$, trong đó mỗi phần tử là hiệu giữa một pixel và một centroid.
 - Hàm `np.linalg.norm(..., axis=2)` được dùng để tính khoảng cách Euclidean giữa từng pixel và từng centroid (trên không gian màu RGB).
 - Kết quả là một mảng 2 chiều có shape $(\text{num_pixels}, k)$, chứa khoảng cách từ từng pixel đến tất cả các centroid.
 - Cuối cùng, `np.argmin(..., axis=1)` được dùng để xác định chỉ số centroid gần nhất (có khoảng cách nhỏ nhất) cho mỗi pixel.
 - Nhờ việc vector hóa như vậy, quá trình gán cụm diễn ra nhanh và hiệu quả hơn nhiều so với sử dụng vòng lặp lồng nhau theo cách truyền thống.
- `update_centroids(img_1d, labels, k_clusters)`: Hàm này cập nhật lại các centroid dựa trên trung bình của các pixel được gán vào mỗi cụm.
 - Với mỗi cụm i , chọn các pixel có nhãn tương ứng từ `img_1d[labels == i]`.
 - Nếu cụm không rỗng, tính trung bình theo từng kênh màu để cập nhật centroid mới.
 - Nếu một cụm không có điểm nào, chương trình sẽ chọn một pixel ngẫu nhiên từ ảnh làm centroid thay thế, tránh lỗi chia cho 0.
3. `np.allclose(...)`: Kiểm tra hội tụ của thuật toán. Nếu các centroid mới và cũ không khác biệt đáng kể, thuật toán sẽ dừng sớm để tiết kiệm thời gian tính toán.
4. Kết thúc và trả về centroid cùng nhãn cụm.

Đầu ra:

- `centroids`: mảng $(K, 3)$ chứa các màu đại diện.
- `labels`: mảng kích thước $(\text{num_pixels},)$ chứa chỉ số cụm của từng pixel.

2.2.2 generate_2d_img(img_2d_shape, centroids, labels)

Tái tạo ảnh 2D từ nhãn cụm và màu centroid:

- Gán mỗi pixel màu của centroid tương ứng.
- Chuyển từ mảng 1D về ảnh 2D theo kích thước gốc.

2.2.3 compress_image(image_path, k, max_iter, init_method)

Hàm tổng hợp giúp thực hiện toàn bộ quy trình nén màu ảnh, trả về ảnh gốc và ảnh đã nén

1. Đọc ảnh từ đường dẫn.
2. Chuyển ảnh về dạng 1D.
3. Áp dụng K-Means để phân cụm màu.
4. Tái tạo ảnh mới chỉ chứa đúng K màu.

2.3 Cải tiến bằng thuật toán K-Means++

Một trong những yếu tố ảnh hưởng lớn đến hiệu quả của thuật toán K-Means là cách khởi tạo các tâm cụm ban đầu. Nếu chọn ngẫu nhiên, các centroid có thể bị phân bố không đều, dẫn đến hội tụ chậm hoặc rơi vào cực trị cục bộ. Để cải thiện vấn đề này, phương pháp K-Means++ được đề xuất nhằm chọn các centroid ban đầu một cách có chủ đích hơn.

Nguyên lý hoạt động của hàm init_kmeans_plus_plus(img_1d, k_clusters):

- Đầu tiên, chọn ngẫu nhiên một điểm dữ liệu từ tập đầu vào làm centroid đầu tiên.
- Với mỗi điểm còn lại trong tập dữ liệu, tính khoảng cách bình phương đến centroid gần nhất trong các centroid đã được chọn.
- Ta chọn centroid tiếp theo theo một cơ chế ngẫu nhiên có trọng số: mỗi điểm sẽ được gán một xác suất lựa chọn, tỉ lệ thuận với bình phương khoảng cách của nó đến centroid gần nhất. Điểm nằm càng xa các centroid hiện tại thì càng có cơ hội được chọn làm centroid tiếp theo.
- Quá trình này được lặp lại cho đến khi đã chọn đủ K centroid.

Ưu điểm của K-Means++ so với khởi tạo ngẫu nhiên:

- **Centroid phân bố hợp lý hơn:** Việc chọn tâm cụm theo xác suất có điều kiện giúp các centroid ban đầu được phân bố cách xa nhau, đại diện tốt hơn cho cấu trúc dữ liệu tổng thể.
- **Giảm số vòng lặp hội tụ:** Do cụm được khởi tạo hợp lý ngay từ đầu, thuật toán cần ít lần cập nhật hơn để đạt trạng thái hội tụ, giúp tiết kiệm thời gian tính toán.
- **Hạn chế rơi vào cực trị cục bộ:** Kết quả phân cụm ít phụ thuộc vào việc chọn ngẫu nhiên ban đầu, tăng tính ổn định giữa các lần chạy khác nhau.

Với cải tiến này, K-Means++ giúp khắc phục một trong những điểm yếu quan trọng của K-Means và được áp dụng rộng rãi trong các bài toán phân cụm thực tế.

2.4 Cải tiến bằng thuật toán Mini-Batch K-Means

Một nhược điểm lớn của thuật toán K-Means truyền thống là chi phí tính toán và lưu trữ cao khi áp dụng cho tập dữ liệu lớn, đặc biệt là ảnh có độ phân giải cao. Mỗi vòng lặp yêu cầu duyệt qua toàn bộ tập dữ liệu để gán cụm và cập nhật centroid, khiến thời gian chạy tăng đáng kể. Để giải quyết vấn đề này, thuật toán **Mini-Batch K-Means** được đề xuất như một phiên bản cải tiến hiệu quả hơn.

Nguyên lý hoạt động của hàm: `mini_batch_kmeans(img_1d, k_clusters, max_iter, batch_size, init_centroids)`

- Thay vì sử dụng toàn bộ dữ liệu trong mỗi vòng lặp, Mini-Batch K-Means chỉ sử dụng một **mini-batch** nhỏ (ví dụ: 100–1000 pixels) được chọn ngẫu nhiên từ tập dữ liệu.
- Mỗi vòng lặp gồm 2 bước:
 1. Gán mỗi điểm trong mini-batch vào cụm gần nhất.
 2. Cập nhật centroid của các cụm dựa trên điểm trung bình của các điểm trong cụm đó.
- Sau khi lặp qua nhiều mini-batch, thuật toán tiến tới hội tụ tương tự K-Means truyền thống.

Ưu điểm:

- **Tăng tốc độ huấn luyện:** Vì chỉ sử dụng một phần dữ liệu mỗi vòng, thuật toán xử lý nhanh hơn đáng kể, phù hợp với các bài toán lớn như phân cụm ảnh có kích thước lớn.
- **Giảm yêu cầu bộ nhớ:** Không cần tải toàn bộ dữ liệu vào bộ nhớ tại mỗi vòng lặp.
- **Hiệu quả tương đương K-Means:** Trong hầu hết các trường hợp, Mini-Batch K-Means vẫn cho kết quả phân cụm rất gần với K-Means truyền thống, trong khi tiết kiệm thời gian đáng kể.

Nhược điểm: Do tính ngẫu nhiên của các mini-batch, kết quả có thể dao động nhẹ giữa các lần chạy, nhưng có thể được khắc phục bằng cách tăng số vòng lặp.

Kết luận: Mini-Batch K-Means là một lựa chọn tối ưu khi cần cân bằng giữa độ chính xác và hiệu suất, đặc biệt hữu ích trong các ứng dụng xử lý ảnh hoặc phân cụm dữ liệu lớn.

3 Kết quả thực nghiệm

3.1 Ảnh phân cụm với các giá trị k khác nhau (khởi tạo ngẫu nhiên)



Hình 1: Ảnh sau phân cụm với các giá trị k khác nhau (khởi tạo ngẫu nhiên)
Unsplash

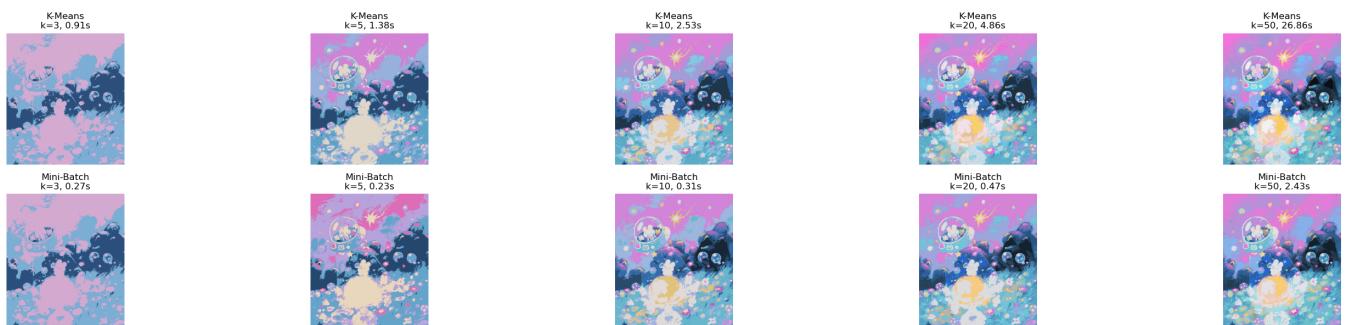
Nhận xét:

- $k = 3$, ảnh bị mất gần như toàn bộ chi tiết, chỉ còn các mảng màu lớn phân vùng rõ rệt.
- $k = 5$ và $k = 10$, mức độ chi tiết trong ảnh được cải thiện đáng kể, ánh sáng và vùng pháo hoa dần rõ nét.

- $k = 20$, ảnh tái hiện được nhiều vùng màu rực rỡ, ảnh sáng pháo hoa đa dạng hơn, gần giống ảnh gốc. Tuy nhiên vẫn tồn tại hiện tượng phân lớp màu.
- Thời gian xử lý tăng rõ rệt khi tăng k , đặc biệt khi từ $k = 10$ trở lên.

Kết luận: Tăng số cụm k giúp ảnh chi tiết và giống ảnh gốc hơn, nhưng sẽ tốn nhiều thời gian xử lý hơn. Với ảnh có độ phức tạp cao như cảnh pháo hoa, cần $k \geq 10$ để giữ được phần lớn đặc trưng màu sắc và chi tiết. Tuy nhiên, nếu k quá lớn sẽ gây hiện tượng phân lớp màu và tăng thời gian tính toán đáng kể. Do đó, cần cân nhắc giữa độ chi tiết mong muốn và thời gian xử lý khi chọn giá trị k .

3.2 So sánh Mini-Batch K-Means và K-Means thường



Hình 2: So sánh kết quả phân cụm giữa K-Means và Mini-Batch K-Means

Nhận xét:

- Ảnh kết quả giữa K-Means và Mini-Batch khá tương đồng về mặt thị giác với cùng giá trị k , tức Mini-Batch vẫn giữ được chất lượng phân cụm tốt.
- Tuy nhiên, Mini-Batch giúp giảm thời gian xử lý đáng kể.
 - $k = 50$, K-Means thường mất tới **26.86s** trong khi Mini-Batch chỉ mất **2.43s**, nhanh hơn hơn 10 lần.
 - $k = 20$, thời gian giảm từ **4.86s** xuống **0.47s**.

Kết luận: Mini-Batch K-Means là một cải tiến phù hợp với ảnh có kích thước lớn hoặc yêu cầu xử lý thời gian thực. Mặc dù không cập nhật centroids theo toàn bộ ảnh, kết quả vẫn đảm bảo ổn định và gần tương đương K-Means thường.

3.3 So sánh các phương pháp khởi tạo centroid



Hình 3: Khởi tạo random



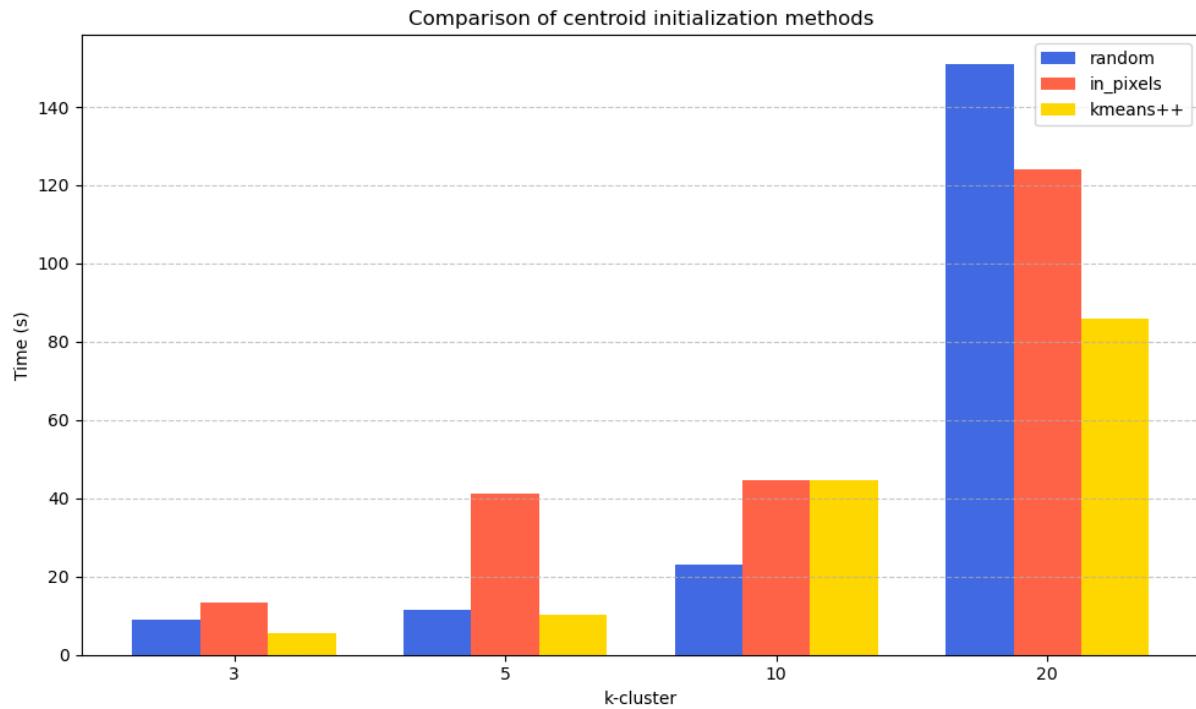
Hình 4: Khởi tạo in_pixels



Hình 5: Khởi tạo kmeans++
Unsplash

Nhận xét:

- $k = 3$ và $k = 5$: `kmeans++` cho thời gian chạy thấp nhất, hội tụ nhanh nhất.
- $k = 10$: Cả ba phương pháp có thời gian gần bằng nhau.
- $k = 20$: `random` mất thời gian lớn (150.82s), trong khi `kmeans++` nhanh hơn đáng kể (85.93s).



Hình 6: Biểu đồ so sánh thời gian chạy của các phương pháp khởi tạo centroid

Biểu đồ thể hiện thời gian chạy của thuật toán K-Means với ba phương pháp khởi tạo centroid khác nhau: `random`, `in_pixels`, và `kmeans++`, được thử nghiệm trên cùng một ảnh với các giá trị $k = 3, 5, 10, 20$.

Kết luận: Thời gian xử lý của thuật toán K-Means tăng theo số cụm k , nhưng mức độ tăng còn phụ thuộc mạnh vào phương pháp khởi tạo centroid. Qua các thực nghiệm và kết quả phân cụm, có thể thấy:

- **Random:** Thời gian chạy không ổn định, có thể hội tụ chậm nếu các centroid khởi tạo quá gần nhau. Ảnh kết quả dễ bị sai lệch màu nếu khởi tạo không hợp lý.
- **In-pixels:** Có cải thiện hơn so với random do chọn từ các màu có thực trong ảnh, tuy nhiên vẫn có khả năng chọn các pixel gần nhau nên không đảm bảo tốt sự phân bố ban đầu.
- **K-Means++:** Mặc dù bước khởi tạo mất nhiều thời gian hơn, nhưng nhờ chọn centroid ban đầu cách đều nhau, quá trình hội tụ nhanh hơn rõ rệt. Tổng thời gian xử lý vẫn thấp hơn đáng kể so với hai phương pháp còn lại, đặc biệt khi số cụm lớn ($k = 20$).

4 Tài liệu tham khảo

1. Shubham Saxena, *K-Means Clustering - Introduction*, GeeksforGeeks, <https://www.geeksforgeeks.org/machine-learning/k-means-clustering-introduction/>, truy cập ngày 16/06/2025.
2. W3Schools, *Python Machine Learning - K-means Clustering*, https://www.w3schools.com/python/python_ml_k-means.asp, truy cập ngày 16/06/2025.
3. Tiep Vu, *Thuật toán K-Means – Machine Learning cơ bản*, Machine Learning Cơ Bản, <https://machinelearningcoban.com/2017/01/01/kmeans/>, truy cập ngày 16/06/2025.
4. Shubham Saxena, *K-Means Algorithm*, GeeksforGeeks, <https://www.geeksforgeeks.org/machine-learning/ml-k-means-algorithm/>, truy cập ngày 20/06/2025.
5. CafeDev Team, *Tự học Machine Learning – Thuật toán K-Means*, CafeDev, <https://cafedev.vn/tu-hoc-ml-thuat-toan-k-mean/>, truy cập ngày 20/06/2025.
6. Shubham Saxena, *Mini-Batch K-Means Clustering Algorithm*, GeeksforGeeks, <https://www.geeksforgeeks.org/machine-learning/ml-mini-batch-k-means-clustering-algorithm/>, truy cập ngày 22/06/2025.

5 Acknowledgement

- Em xin chân thành cảm ơn cô Trần Thị Thảo Nhi đã giao một đề tài thú vị, giúp em có cơ hội tìm hiểu sâu về thuật toán K-Means và ứng dụng thực tế của nó. Em cũng rất biết ơn sự chỉ dẫn tận tình của cô trong suốt quá trình thực hiện đồ án, nhờ đó em đã hiểu rõ hơn về thuật toán, nâng cao kỹ năng lập trình cũng như cách trình bày một báo cáo khoa học.
- Trang **GeeksforGeeks** là nguồn tham khảo chính để em tìm hiểu về thuật toán K-Means và cách triển khai trong Python.
- **ChatGPT** đã hỗ trợ em trong suốt quá trình phát triển và viết báo cáo, cụ thể:
 - Gợi ý và cải tiến hàm `test_k_values()` và `test_kmeans_vs_minibatch_multiple_k` giúp kiểm tra, so sánh thuật toán với nhiều giá trị k , và hỗ trợ trực quan hóa kết quả qua biểu đồ.
 - Viết chú thích code bằng tiếng Anh ngắn gọn và rõ ràng, giúp dễ bảo trì.
 - Gợi ý cấu trúc và trình bày LaTeX cho báo cáo, giúp em sắp xếp nội dung khoa học, dễ theo dõi và chuyên nghiệp.

Nhờ vào sự hỗ trợ từ giảng viên, tài liệu tham khảo và công cụ AI như ChatGPT, em đã hoàn thiện đồ án đúng tiến độ và nâng cao kỹ năng lập trình, phân tích và trình bày khoa học.