

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Báo cáo đồ án 1
Color Compression

Môn học: Toán ứng dụng và thống kê cho CNTT

MTH00051_22CLC09

Sinh viên:

Nguyễn Hồ Đăng Duy
22127085
22CLC09

Giảng viên hướng dẫn:

Nguyễn Ngọc Toàn
Nguyễn Văn Quang Huy

Thành phố Hồ Chí Minh, tháng 7 năm 2024



Mục lục

1	Thông tin sinh viên	2
2	Ý tưởng thực hiện	2
2.1	Thuật toán K-means clustering	2
2.2	Sử dụng K-means để giảm màu ảnh	4
3	Mô tả các hàm	5
3.1	Các thư viện cần thiết	5
3.2	read_img(img_path)	5
3.3	Hàm show_img(img_2d)	5
3.4	save_img(img_2d, img_path)	6
3.5	convert_img_to_1d(img_2d)	6
3.6	kmeans(img_1d, k_clusters, max_iter, init_centroids='random')	6
3.6.1	Khởi tạo centroid	6
3.6.2	Vòng lặp chính (K-Means Iteration)	6
3.7	generate_2d_img(img_2d_shape, centroids, labels)	7
3.8	comparision(original, result, k_clusters, init_centroids)	7
4	Kết quả	8
4.1	init_centroids = 'random'	8
4.2	init_centroids = 'in_pixels'	10
5	Nhận xét kết quả	12
5.1	Thời gian chạy của thuật toán	12
5.2	Dộ chính xác của thuật toán	12
5.3	Kết luận	13
Tài liệu tham khảo		14
Công tác viên		14

1 Thông tin sinh viên

- Họ và tên: Nguyễn Hồ Đăng Duy
- MSSV: 22127085
- Lớp: 22CLC09

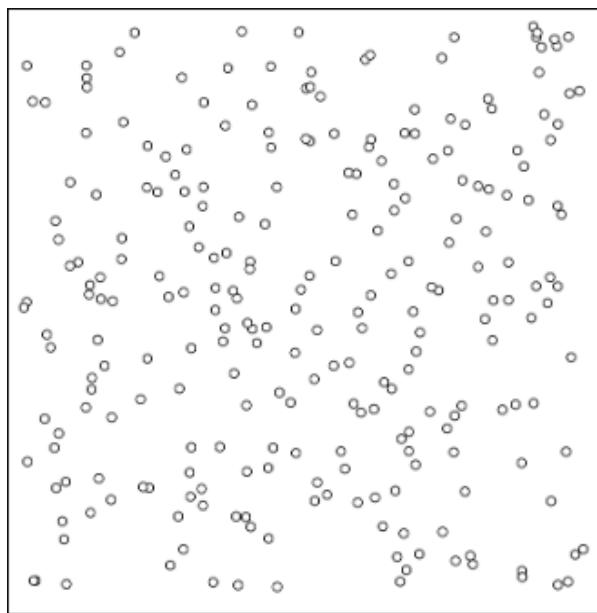
2 Ý tưởng thực hiện

2.1 Thuật toán K-means clustering

K-means clustering là một trong những thuật toán cơ bản nhất trong Unsupervised learning. Thuật toán này dùng để phân dữ liệu đầu vào thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau [4]

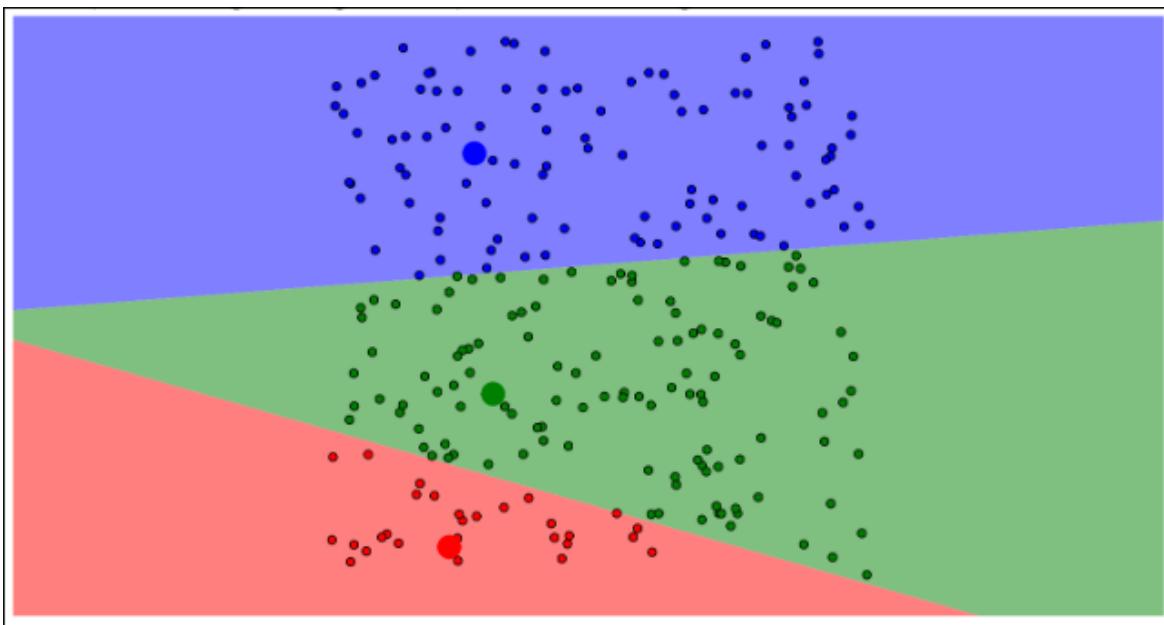
Ta có thể mô tả thuật toán trên qua các hình sau:

- Ban đầu ta có các điểm dữ liệu như sau



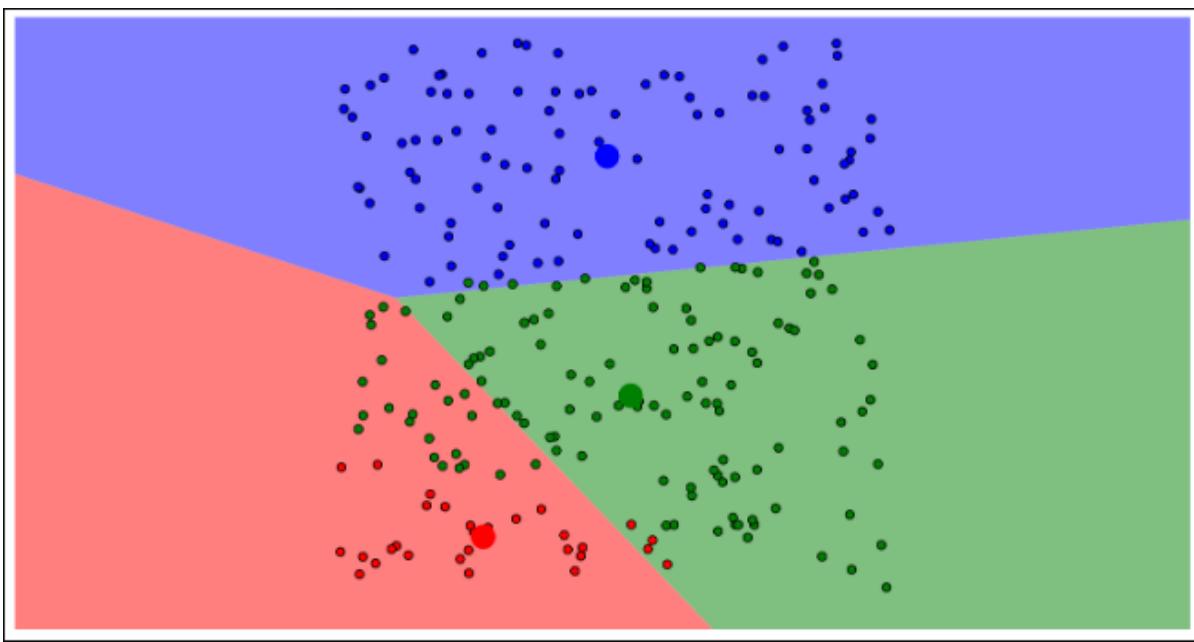
Hình 1: Điểm dữ liệu ban đầu

- Chọn số lượng nhóm K muốn phân loại dữ liệu. Thuật toán sẽ chọn K điểm bất kì trong tập dữ liệu để làm điểm trung tâm ban đầu cho từng nhóm. Tính toán khoảng cách của từng điểm đến từng trung tâm, sau đó gán điểm dữ liệu vào nhóm gần nhất (có khoảng cách gần nhất) với điểm trung tâm



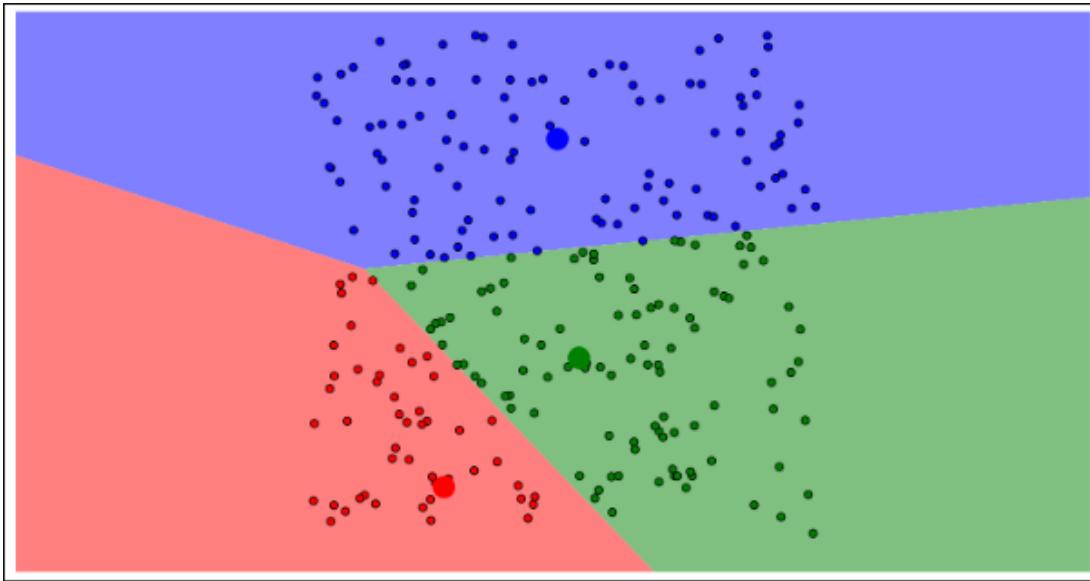
Hình 2: Chọn K nhóm và phân loại dữ liệu vào từng nhóm

- Tính toán lại các điểm trung tâm mới của mỗi nhóm bằng cách lấy trung bình của các điểm dữ liệu trong nhóm đó



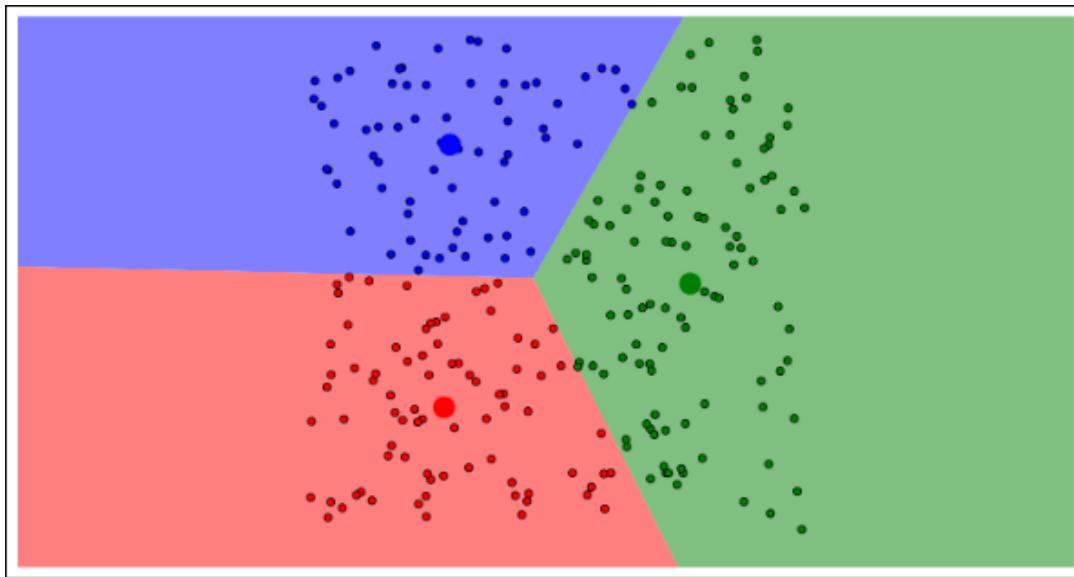
Hình 3: Tính toán và gán lại các điểm trung tâm mới

- Tính toán lại khoảng cách của từng điểm dữ liệu đến từng điểm trung tâm mới. Sau đó gán lại các điểm dữ liệu



Hình 4: Gán lại các điểm dữ liệu

- Thực hiện lặp đi lặp lại các bước trên cho đến khi không còn sự thay đổi giữa các điểm trung tâm thì dừng lại. Kết quả là nhóm dữ liệu đã được phân loại thành công



Hình 5: Kết thúc thuật toán K-means

2.2 Sử dụng K-means để giảm màu ảnh

Dựa vào các bước của thuật toán K-means như trên, ta có ý tưởng sử dụng thuật toán này để giảm màu của ảnh với mục tiêu là vẫn bảo toàn được nội dung của ảnh ban đầu như sau:[3]

1. Xử lý ảnh đầu vào: Chuyển đổi ảnh đầu vào từ kích thước 2D (height, width, channels) sang 1D (height x width, channels). Trong đó mỗi hàng là một vector màu.

2. Chọn số cluster (k): Số cluster được chọn phụ thuộc vào mức độ giảm màu mong muốn. Số cluster thường nhỏ hơn số màu gốc của ảnh.
3. Tạo ra k điểm centroids: Nếu `init_centroids` là `random` thì thuật toán sẽ tạo ra k điểm centroids ngẫu nhiên trong đoạn $[0; 255]$. Nếu `init_centroids` là `in_pixels` thì thuật toán sẽ chọn ngẫu nhiên k điểm dữ liệu từ `img_1d` để dùng như centroids.
4. Phân loại các điểm dữ liệu: Gán mỗi điểm dữ liệu vào cluster gần nhất dựa trên khoảng cách Euclid giữa điểm và centroids.
5. Cập nhật centroids: Tính toán lại vị trí của các centroids bằng cách lấy trung bình của tất cả các điểm dữ liệu trong cùng một cluster.
6. Lặp lại bước 4 và 5 cho đến khi không có sự thay đổi nào ở các centroids nữa hoặc đạt đến số lần lặp tối đa (`max_iter`).
7. Chuyển đổi ảnh: Sau khi đã hoàn thành thuật toán K-means, sử dụng các màu đã xác định ở các bước trước để tạo ra ảnh mới đã giảm màu.

3 Mô tả các hàm

3.1 Các thư viện cần thiết

Các thư viện được sử dụng trong đồ án là: numpy, PIL và matplotlib

- numpy: Cho phép làm việc hiệu quả với ma trận và mảng. Đặc biệt là dữ liệu ma trận và mảng lớn vì có tốc độ xử lý nhanh, cấu trúc hàm đơn giản.
- Image từ PIL: Được sử dụng để đọc ảnh.
- matplotlib.pyplot: Được sử dụng để hiển thị ảnh.

3.2 `read_img(img_path)`

Hàm `read_img` được thiết kế để có thể đọc hình ảnh từ `img_path` được truyền vào và đổi nó thành một mảng numpy 2D bằng lệnh `img_2d = np.array(im)` giúp dễ dàng thao tác trong các tác vụ xử lý hình ảnh về sau.

3.3 `Hàm show_img(img_2d)`

Hàm `show_img` dùng để hiển thị một hình ảnh từ mảng numpy 2D sử dụng thư viện matplotlib. Vì bài toán đặt ra là xử lý hình ảnh nên có thể dùng `plt.axis('off')` để tắt các trục tọa độ xung quanh hình ảnh, giúp hình ảnh hiển thị rõ ràng hơn.

3.4 save_img(img_2d, img_path)

Hàm **save_img** được dùng để lưu một hình ảnh từ mảng numpy 2D vào một đường dẫn cho output file:

1. Kiểm tra xem đường dẫn **img_path** có hợp lệ hay chưa. Nếu chưa thì chỉnh sửa để đảm bảo có thể lưu được hình ảnh.
2. Chuyển đổi mảng numpy **img_2d** ban đầu thành đối tượng hình ảnh của PIL bằng hàm *Image.fromarray()*.
3. Yêu cầu người dùng chọn kiểu tệp: Hàm có hỗ trợ người dùng lưu ảnh dưới các dạng jpg, jpeg, png hoặc pdf. Người dùng nhập kiểu tệp muốn lưu, nếu kiểu tệp không hợp lệ thì hàm sẽ in ra thông báo lỗi.
4. Lưu hình ảnh: Đường dẫn **img_path** được cập nhật để bao gồm tên file đã quy định sẵn và kiểu tệp vừa được chọn. Sau đó hình ảnh sẽ được lưu vào đường dẫn này bằng hàm *save()*

3.5 convert_img_to_1d(img_2d)

Hàm **convert_img_to_1d** được dùng để chuyển một hình ảnh từ dạng 2D sang dạng 1D, giúp dễ dàng xử lý và phân tích hình ảnh trong thuật toán K-means clustering. Hàm *reshape()* được sử dụng để chuyển ảnh 2D có dạng (height, width, channels) sang 1D (height x width, channels).

3.6 kmeans(img_1d, k_clusters, max_iter, init_centroids='random')

Hàm **kmeans** được thiết kế để thực hiện thuật toán K-Means clustering trên một hình ảnh, chia các pixel của hình ảnh thành các cụm dựa trên màu sắc. Kết quả trả về các centroids và labels của mỗi pixel. Thuật toán sẽ được chia thành các bước nhỏ sau:

3.6.1 Khởi tạo centroid

Kiểm tra **init_centroids** để có thể khởi tạo centroid phù hợp

- Nếu là *random* thì centroid sẽ được khởi tạo với các giá trị ngẫu nhiên từ 0 đến 255 cho mỗi kênh màu.
- Nếu là *in_pixels*, các centroid được khởi tạo bằng cách chọn ngẫu nhiên các pixel từ hình ảnh gốc.
- Nếu không phải 2 giá trị trên, hàm sẽ trả về thông báo lỗi.

3.6.2 Vòng lặp chính (K-Means Iteration)

1. **Tính khoảng cách và gán cụm:** Khoảng cách từ mỗi pixel đến mỗi centroid được tính toán. Sau đó mỗi pixel được gán vào cụm có centroid gần nhất:

- *img_1d[:, None]* sử dụng kỹ thuật broadcasting của numpy để thêm một chiều dữ liệu mới vào mảng **img_1d**. Từ đó có thể dễ dàng tính toán khoảng cách giữa mỗi pixel với tất cả centroid. Kết quả là **img_1d** có dạng (height x width, 1, channels).

- Phép tính `np.linalg.norm(..., axis=2)` dùng để tính toán chuẩn Euclidean dọc theo trục 2 (trục của các kênh màu). Kết quả là một mảng **distances** có dạng (height x width, k_clusters) trong đó mỗi phần tử biểu diễn khoảng cách từ một điểm dữ liệu tới một centroid.
- Tìm chỉ số của centroid gần nhất bằng `np.argmin(distances, axis=1)`. Hàm sẽ tìm chỉ số của giá trị nhỏ nhất dọc theo trục 1 (trục của các centroid) hay nói cách khác, nó tìm chỉ số của centroid gần nhất cho mỗi điểm dữ liệu. Kết quả là một mảng **labels** có dạng (height x width), trong đó mỗi phần tử là chỉ số của centroid gần nhất cho điểm dữ liệu tương ứng.

2. **Cập nhật centroids:** Tính toán lại các centroid mới dựa trên các điểm dữ liệu đã được gán nhãn vào mỗi cụm (cluster):

- Đầu tiên là một vòng lặp trên các clusters, mỗi lần lặp tương ứng với việc tính toán centroid mới cho cụm k.
- Kiểm tra số lượng điểm dữ liệu trong mỗi cụm bằng `np.sum(labels == k)`.
- Kiểm tra cụm rỗng: Nếu kết quả kiểm tra phía trên là rỗng (`sum=0`) thì **centroids[k]** được giữ nguyên, nghĩa là centroids của cụm này không thay đổi. Ngược lại, **centroids[k]** sẽ được tính toán bằng cách tính giá trị trung bình của các điểm dữ liệu được gán vào cụm k dọc theo trục 0 (trục của điểm dữ liệu), tức là tính trung bình của từng kênh màu.
- Sau khi kết thúc tính toán, tạo một mảng numpy mới từ các centroid cho mỗi cụm. Kết quả là mảng **new_centroids** có hình dạng (k_clusters, channels).

3. **Kiểm tra điều kiện dừng:** Nếu các centroid không thay đổi so với lần lặp trước, vòng lặp sẽ dừng lại. Nếu không, vòng lặp sẽ tiếp tục thực hiện tiếp tục các bước trên cho đến khi thỏa điều kiện dừng hoặc đạt `max_iter`.

3.7 generate_2d_img(img_2d_shape, centroids, labels)

Hàm **generate_2d_img** được dùng để tạo lại hình ảnh 2D từ các centroid màu và labels của mỗi pixel. Kết quả là một hình ảnh mới mà mỗi pixel có màu tương ứng với centroid của cụm mà nó được gán vào.

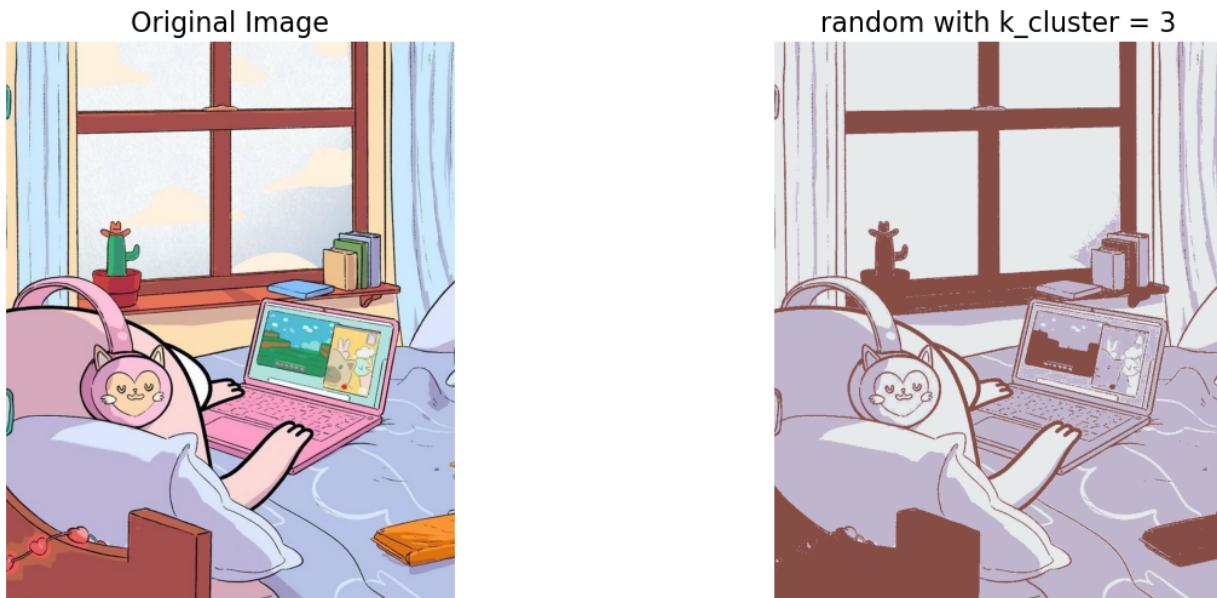
3.8 comparision(original, result, k_clusters, init_centroids)

Hàm **comparision** được thiết kế để hiển thị và so sánh trực quan giữa hình ảnh gốc và hình ảnh kết quả sau khi thực hiện thuật toán K-Means. Điều này giúp người dùng dễ dàng nhìn thấy sự khác biệt giữa hai hình ảnh.

4 Kết quả

Tất cả các kết quả dưới đây đều được thực hiện với **max_iter = 100**

4.1 init_centroids = 'random'



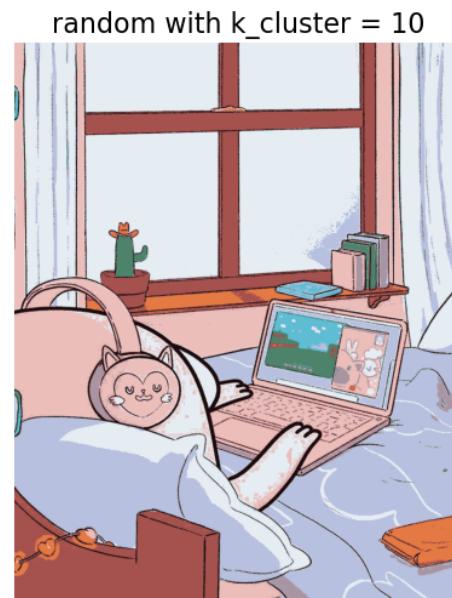
Hình 6: $k_cluster=3$



Hình 7: $k_cluster=5$



Hình 8: k_cluster=7



Hình 9: k_cluster=10

	Tổng thời gian chạy (10 lần)	Trung bình
k_cluster = 3	22.4 giây	2.24 giây
k_cluster = 5	27.3 giây	2.73 giây
k_cluster = 7	43.6 giây	4.36 giây
k_cluster = 10	87.3 giây	8.73 giây

4.2 init_centroids = 'in_pixels'



Hình 10: k_cluster=3



Hình 11: k_cluster=5



Hình 12: k_cluster=7



Hình 13: k_cluster=10

	Tổng thời gian chạy (10 lần)	Trung bình
k_cluster = 3	12.8 giây	1.28 giây
k_cluster = 5	27.3 giây	2.73 giây
k_cluster = 7	39.6 giây	3.96 giây
k_cluster = 10	66.7 giây	6.67 giây

5 Nhận xét kết quả

5.1 Thời gian chạy của thuật toán

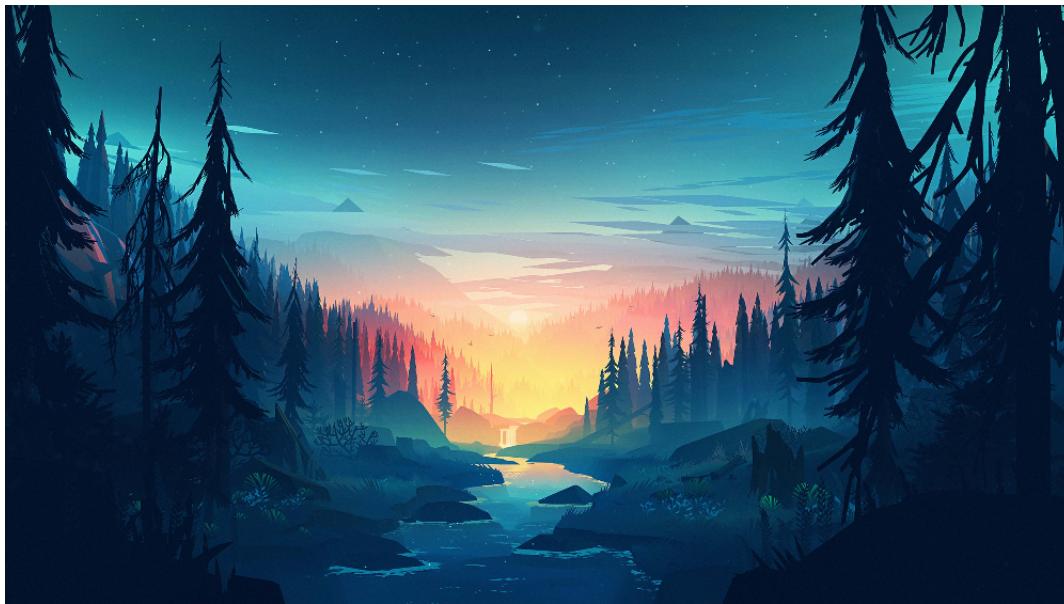
Thời gian chạy của thuật toán phụ thuộc vào các yếu tố sau:

- **Kích thước ảnh:** Khi ảnh càng lớn, thuật toán sẽ phải xử lý một lượng lớn các điểm ảnh, dẫn đến thời gian chạy càng lâu.
- **Số lượng màu cần giảm:** Nếu số lượng màu ban đầu trong ảnh lớn thì việc giảm số lượng về k_cluster màu sẽ phải mất nhiều thời gian hơn.
- **Độ chính xác:** Nếu yêu cầu một độ chính xác cao hơn cho kết quả giảm màu, thuật toán sẽ phải thực hiện nhiều vòng lặp để tìm ra cách phân nhóm tối ưu hơn. Điều này cũng ảnh hưởng đến thời gian chạy của thuật toán.
- **Khả năng tính toán của máy tính:** Thời gian chạy của thuật toán cũng phụ thuộc vào tốc độ xử lý của máy tính. Nếu một máy tính có cấu hình cao, CPU mạnh hơn thì sẽ thực hiện thuật toán nhanh hơn so với máy tính có cấu hình thấp.
- **Cách chọn centroids:** Như đã thấy trong mục kết quả, việc chọn centroids bằng cách **random** hoặc **in_pixels** cũng ảnh hưởng đến tốc độ thực hiện của thuật toán. Kết quả của việc chọn centroids bằng **in_pixels** sẽ được xử lý nhanh hơn

5.2 Độ chính xác của thuật toán

Độ chính xác của thuật toán phụ thuộc vào các yếu tố sau:

- **Số lượng k_clusters:** Khi tăng số lượng nhóm k, thuật toán sẽ gom cụm các điểm ảnh có màu tương tự hơn và tạo ra được một bức ảnh có màu sắc tương đồng với ảnh gốc hơn. Càng giảm số lượng nhóm, bức ảnh chỉ còn lại các màu cơ bản, mất đi các chi tiết nhỏ. Do đó độ chính xác của thuật toán K-Means phụ thuộc nhiều vào việc lựa chọn số lượng k_clusters với ảnh cần xử lý.
- **Khởi tạo các điểm centroids ban đầu:** Thuật toán K-Means có thể rơi vào các điểm cực tiểu địa phương [4], dẫn đến kết quả gom cụm không tối ưu. Để giải quyết vấn đề này, có thể khởi tạo các điểm trung tâm ban đầu dựa trên một thuật toán hoặc phương pháp khác để tìm ra các điểm ảnh gần nhất với các màu trong ảnh.
- **Đặc điểm của ảnh:** Thuật toán K-means thường không hiệu quả khi áp dụng cho các ảnh có những vùng phức tạp hoặc chứa các chi tiết nhỏ. Với những ảnh có nhiễu mạnh, hay đối tượng có chi tiết nhỏ, K-means có thể tạo ra hiệu ứng “bể pixel” hoặc mất các chi tiết nhỏ.
- **Số lần lặp của thuật toán (max_iter):** Số lần lặp của thuật toán càng cao thì độ chính xác của ảnh cũng sẽ cao theo. Ví dụ với cùng k_cluster = 7, init_centroids='random' nhưng khi thay đổi max_iter thì 2 bức ảnh sau sẽ ra 2 màu hoàn toàn khác nhau.



Hình 14: Ảnh gốc



Hình 15: So sánh 2 kết quả dựa trên max_iter

5.3 Kết luận

Sẽ có sự đánh đổi giữa thời gian chạy và độ chính xác của hình ảnh thu được sau khi thực thi thuật toán K-Means. Việc chọn giá trị $k_clusters$ nhỏ có thể làm cho thời gian chạy nhanh hơn nhưng phải hy sinh một số chi tiết của hình ảnh, trong khi giá trị $k_clusters$ lớn hơn sẽ được nhiều chi tiết hơn nhưng cần thời gian xử lý nhiều hơn trong việc tính toán. Đặc biệt là với các file hình ảnh có kích thước lớn.

Tài liệu

- [1] Bala, P. C. (2022, February 15). Learn K-Means clustering by quantizing color images in Python. Hackernoon.com (Last visited June 17, 2024). From <https://hackernoon.com/learn-k-means-clustering-by-quantizing-color-images-in-python>
- [2] Các bước của thuật toán k-Means Clustering — Deep AI KhanhBlog. (n.d.) (Last visited June 15, 2024). From https://phamdinhkhanh.github.io/deepai-book/ch_ml/KMeans.html
- [3] Image compression using K-means clustering. GeeksforGeeks (Last visited June 17, 2024). From <https://www.geeksforgeeks.org/image-compression-using-k-means-clustering/>
- [4] Tiệp, V. H. (2017, January 1). Bài 4: K-means Clustering. Tiep Vu's Blog (Last visited June 17, 2024.). From <https://machinelearningcoban.com/2017/01/01/kmeans/> (Last visited June 17, 2024.)
- [5] Wikipedia contributors. (2024, May 19). k-means++. Wikipedia, The Free Encyclopedia (Last visited June 15, 2024). From <https://en.wikipedia.org/wiki/K-means%2B%2B>

Cộng tác viên

Đồ án 1 Color Compression được thực hiện với sự giúp đỡ của 2 cộng tác viên

1. Chat GPT: <https://chatgpt.com/>
2. GitHub Copilot: <https://github.com/features/copilot>