TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN 1
TOÁN ỨNG DỤNG VÀ THỐNG KÊ CHO CÔNG NGHỆ THÔNG TIN

TÊN ĐỒ ÁN:
COLOR COMPRESSION

LỚP: 21CLC05

Thông tin cá nhân:

21127099 Nguyễn Tấn Lộc

Giảng viên hướng dẫn:
Thầy Vũ Quốc Hoàng
Thầy Lê Thanh Tùng
Cô Phạm Thị Phương Uyên
Thầy Nguyễn Văn Quang Huy

MỤC LỤC

THÔNG TIN CÁ NHÂN

Họ tên: Nguyễn Tấn Lộc

Mã số sinh viên: 21127099

Lớp: 21CLC5

THÔNG TIN ĐỒ ÁN

Mã học phần: MTH00057

Tên học phần: Toán ứng dụng và thống kê cho công nghệ thông tin

Tên đồ án: Color Compression

KẾT QUẢ ĐỒ ÁN

1. Ý tưởng thực hiện

Thuật toán K-Means là một thuật toán **phân cụm dữ liệu** không giám sát, được sử dụng để **phân chia dữ liệu thành các cụm** (clusters) **dựa trên các đặc trưng** của chúng. Trong trường hợp của bài toán giảm số lượng màu trong ảnh, ta sử dụng thuật toán K-Means để phân chia các điểm ảnh thành các cụm dựa trên màu sắc của chúng.

Ý tưởng chung để giảm số lượng màu của ảnh sử dụng thuật toán K-Means như sau:

1. Đọc ảnh đầu vào và chuyển đổi thành mảng numpy để tiện xử lý.
2. Chuyển đổi ma trận ảnh thành một mảng 1D, trong đó mỗi phần tử là một pixel với các kênh màu tương ứng.
3. Áp dụng thuật toán K-Means để gom nhóm các pixel thành k_clusters nhóm.
4. Tính toán các centroid cho từng nhóm dựa trên các pixel trong nhóm đó.
5. Gán nhãn cho từng pixel dựa trên centroid gần nhất.
6. Thay thế mỗi pixel bằng centroid tương ứng của nó.
7. Chuyển đổi lại mảng 1D thành ma trận ảnh ban đầu với số lượng màu đã được giảm.
8. Hiển thị ảnh ban đầu và ảnh đã giảm số lượng màu để so sánh.

Như vậy nhìn chung, ta đã có một góc nhìn về thuật toán này, sau đây sẽ là các bước cụ thể cho thuật toán K-Means:

Bước 1: Đọc ảnh và chuyển đổi nó thành một mảng NumPy 1 chiều (1D) với mỗi phần tử là một điểm ảnh.

Bước 2: Khởi tạo các centroid (tâm cụm) ban đầu. Có hai cách để khởi tạo centroid: ngẫu nhiên hoặc lấy ngẫu nhiên từ các điểm ảnh trong ảnh gốc.

Bước 3: Gán nhãn cho mỗi điểm ảnh bằng cách tìm centroid gần nhất với nó.

Bước 4: Cập nhật lại vị trí của các centroid bằng cách tính trung bình cộng của tất cả các điểm ảnh thuộc cùng một cụm.

Bước 5: Lặp lại bước 3 và 4 cho đến khi thuật toán hội tụ hoặc đạt đến số lần lặp tối đa.

- Sau khi thuật toán hội tụ, có thể sử dụng nhãn của mỗi điểm ảnh để thay thế giá trị màu sắc của nó bằng giá trị màu sắc của centroid tương ứng giúp giảm số lượng màu trong ảnh.

2. Mô tả các hàm

- Em có tổng cộng 4 hàm như sau:
  - Hàm kmeans → Đây là hàm thiết kế thuật toán K-Means.
  - Hàm reduce_color → Đây là hàm giảm số lượng màu của ảnh.
  - Hàm save_image → Đây là hàm lưu ảnh tùy theo định dạng.
  - Hàm main → Đây là hàm thực hiện tất cả các tác vụ input, output và save ảnh.
- Mô tả hàm **kmeans**:
  - Về input, ta sẽ có các biến như sau:
    - **img_1d**: Một mảng numpy 1D có hình dạng (**height * width, num_channels)**, đại diện cho ảnh gốc (là ảnh mà người dùng nhập ten vào).
    - **k_clusters**: Số lượng cụm (clusters) mà thuật toán K-Means sẽ phân loại ảnh. → Ta dựa trên số lượng cụm này để nhận xét về kết quả và thời gian chạy thuật toán.
    - **max_iter**: Số lần lặp tối đa mà thuật toán sẽ thực hiện trước khi dừng. → Mặc định sẽ là 1000.
    - **init_centroids**: Phương pháp khởi tạo centroid ban đầu, có hai giá trị cho lựa chọn: **random** (lựa chọn ngẫu nhiên từ 0-255 theo size là shape (hình dạng) của ảnh gốc) hoặc **in_pixels**. (lựa chọn ngẫu nhiên các điểm ảnh **DUY NHẤT** trong mảng của ảnh gốc).
  - Về output, hàm sẽ trả về các biến như sau:
    - **centroids**: Mảng numpy có hình dạng **(k_clusters, num_channels)** → lưu trữ các centroid màu.
    - **labels**: Mảng numpy có hình dạng **(height * width,)** → lưu trữ nhãn cho từng điểm ảnh (ám chỉ số cụm mà điểm ảnh đó thuộc về).
  - Các bước trong thuật toán K-Means:
    - **Khởi tạo centroids ban đầu**: Dựa vào init_ centroids được chỉ định (**random** hoặc **in_pixels**), các centroid ban đầu sẽ được khởi tạo.

- **Trong điều kiện random**: em thực hiện việc khởi tạo các centroids bằng cách sử dụng hàm **np.random.randint** để tạo ra các giá trị ngẫu nhiên trong khoảng từ 0 đến 255.
    - Cụ thể, em tạo một ma trận centroids với kích thước **(k_clusters, img_1d.shape[1])**, trong **đó k_clusters** là số lượng các centroids cần khởi tạo và **img_1d.shape[1]** là số chiều của mỗi centroid, tương ứng với số kênh màu của ảnh.
    - Giải thích về img_1d.shape[1]: nó là truy cập vào phần tử thứ hai của tuple shape của ảnh(img_1d.shape), mà **img_1d** là 1 mảng numpy có hình dạng **(height * width, num_channels)** nên img_1d.shape[1] sẽ trả về **num_channels** → tức là số lương kênh màu của mỗi điểm ảnh trong mảng **img_1d**.
    - Hàm np.random.randint trả về một ma trận ngẫu nhiên có kích thước **(k_clusters, img_1d.shape[1])**, trong đó các phần tử nằm trong khoảng từ 0 đến 255 → Sẽ đảm bảo rằng các centroids được khởi tạo ngẫu nhiên và có giá trị trong khoảng màu từ 0 đến 255.
- **Trong điều kiện in_pixels:** em thực hiện việc khởi tạo các centroids bằng cách chọn ngẫu nhiên một số điểm ảnh **DUY NHẤT** từ mảng **img_1d**.
    - Cụ thể, em sử dụng hàm np.unique để tìm các điểm ảnh **duy nhất** trong mảng img_1d → Sẽ đảm bảo rằng mỗi điểm ảnh trong mảng unique_pixels chỉ xuất hiện một lần duy nhất.
    - Tiếp theo, chúng ta sử dụng np.random.choice để chọn ngẫu nhiên k_clusters chỉ mục từ unique_pixels mà không có sự thay thế (replace=False).
    - Cuối cùng, chúng ta gán centroids bằng các điểm ảnh tương ứng với chỉ mục đã chọn ngẫu nhiên từ unique_pixels.
    - Tuy nhiên, em thấy rằng sử dụng np.unique có thể không cần thiết lắm vì đã được em set replace=False trong np.random.choice, điều này đảm bảo rằng các chỉ mục được chọn sẽ không trùng lặp và không cần phải loại bỏ các giá trị trùng lặp từ unique_pixels.

- **Vòng lặp cho đến khi hội tụ hoặc đạt max_iter:** Thuật toán lặp lại cho đến khi các centroid hội tụ (khi khoảng cách giữa các centroid mới và cũ nhỏ hơn một ngưỡng được chỉ định, ở đây em chọn ngưỡng đó là 0.2) hoặc đạt đến số lần lặp tối đa (mặc định là 1000).
    - ***Đầu tiên***, em tính khoảng cách giữa mỗi pixel trong **img_1d** và các centroid trong **centroids.** Em dùng hàm np.linalg.norm để tính độ dài Euclidean giữa mỗi pixel và các centroid. **axis=2** tức là cho phép em tính toán trên trục thứ ba, là tính khoảng cách theo chiều của các kênh màu(chiều thứ hai trong mảng).
    - Vì mảng img_1d là mảng 1D, trong khi đó **centroids** là mảng 2D có kích thước **(k_clusters, num_channels)** nên em phải mở rộng **img_1d** thành mảng 2D bằng cách thêm một chiều mới (xem là chiều thứ hai). Kết quả là em sẽ có một mảng có kích thước (**height * width, 1, num_channels)**, trong đó mỗi pixel được sao chép và mở rộng thành một mảng có kích thước **(1, num_channels)**.
    - ***Tiếp theo***, em gán nhãn cho từng pixel trong ảnh dựa trên khoảng cách đã tính trước đó. Em dùng hàm np.argmin để trả về chỉ mục của centroid có khoảng cách nhỏ nhất cho mỗi pixel(tức là theo chiều dọc, trong mỗi hàng).
    - ***Tiếp theo***, em sẽ cập nhật centroid mới:
        - Trong trường hợp **random**:
            - Tạo một mảng new_centroids có cùng kích thước với centroids và tất cả các phần tử được khởi tạo bằng 0.
            - Với mỗi cluster j từ 0 đến k_clusters - 1, kiểm tra nếu có ít nhất một pixel được gán nhãn vào cluster j (tức là np.sum(labels == j) > 0) → đảm bảo nhóm này không rỗng, thì cập nhật giá trị của new_centroids[j] bằng trung bình của các pixel thuộc cluster j bằng cách sử dụng hàm np.mean(img_1d[labels == j], axis=0).

- Nếu không có pixel nào thuộc cluster j (nhóm rỗng), thì giá trị của new_centroids[j] được gán bằng [0, 0, 0]. Vì em gặp lỗi **RuntimeWarning: invalid value encountered in cast** xảy ra khi giá trị trung bình của một nhóm pixel là NaN (tức là không có pixel nào thuộc nhóm đó). Lỗi này có thể xảy ra nếu giá trị **k_clusters** lớn hơn số lượng pixel thực tế trong hình ảnh.

- Trong trường hợp **in_pixels**:
  - Tạo một mảng new_centroids có kích thước (k_clusters, num_channels) và mỗi phần tử được khởi tạo bằng trung bình của các pixel thuộc cùng cluster j, được tính bằng np.mean(img_1d[labels == j], axis=0), với j từ 0 đến k_clusters - 1.

➔ Theo em thấy, hai trường hợp này khác nhau ở cách cập nhật centroids mới sau khi gán nhãn cho các pixel. Trong trường hợp **random**, mỗi centroid sẽ được cập nhật bằng trung bình của các pixel trong cluster tương ứng (nếu có), trong khi trong trường hợp **in_pixels**, các centroid được cập nhật bằng trung bình của tất cả các pixel trong cùng một cluster.

- ***Bước cuối cùng là xét hội tụ,*** ở bước này em dùng try – except để xử lý trường hợp khi không thể so sánh hai mảng trực tiếp bằng np.allclose(). Nếu không so sánh trực tiếp được thì em sẽ chuyển đổi hình dạng thành mảng 2D để so sánh.

- Mô tả hàm **reduce_color:**
  - Về input, các biến sẽ giống hàm **kmeans, chỉ có thêm 1 biến là image_path tức là tên ảnh gốc**
  - Về output, sẽ trả về biến **reduce_image** là biến sau khi đã sử dụng thuật toán K-means và chuyển đổi về dạng ảnh từ mảng.
  - Các bước trong hàm này:

- ***Đầu tiên,*** em load ảnh từ đường dẫn **image_path** bằng cách sử dụng **Image.open**.
- ***Tiếp theo,*** em chuyển ảnh thành một mảng numpy bằng cách sử dụng **np.array**.
- ***Sau đó,*** em reshape kích thước của mảng ảnh để có thể áp dụng thuật toán K-Means. Kích thước ban đầu của mảng ảnh là (height, width, num_channels), và sau khi reshape, nó sẽ trở thành một mảng 1D có kích thước (height * width, num_channels).
- ***Tiếp theo,*** ap dụng thuật toán K-Means (hàm **kmeans**) lên mảng 1D của ảnh → đưa ra kết quả là các centroid và nhãn của từng pixel.
- Ánh xạ từng pixel trong ảnh gốc tới centroid gần nhất của nó → được thực hiện bằng cách lấy giá trị của centroid tương ứng với nhãn của pixel đấy.
- ***Tiếp đó,*** em reshape lại mảng ảnh đã giảm số lượng màu về kích thước ban đầu (height, width, num_channels).
- Chuyển mảng ảnh về định dạng PIL Image bằng cách sử dụng **Image.fromarray**.
- Trả về ảnh đã giảm số lượng màu.

→ Nói ngắn gọn, hàm **reduce_color** của em sẽ nhận đầu vào là một đường dẫn tới ảnh, số lượng cụm (clusters) mong muốn, số lần lặp tối đa cho thuật toán K-Means và phương pháp khởi tạo centroids ban đầu. Sau khi đã qua xử lý thì hàm sẽ trả về ảnh đã giảm số lượng màu theo yêu cầu.

- Mô tả hàm **save_image**:
  - Về input, sẽ có 2 biến là **img (biến này em sẽ gán biến ảnh đã xử lý)** và **name (biến này là tên của ảnh).**
  - Về output, hàm này sẽ tự động lưu ảnh bằng hàm save.
  - Các bước trong hàm này:
    - Em xác định các định dạng tệp tin hợp lệ (valid_extensions), ví dụ như **'.pdf', '.png', '.jpg'.**
    - Em xác định định dạng tệp tin của tên tệp đầu ra (name) bằng cách lấy phần mở rộng từ vị trí cuối cùng của chuỗi tên.
    - So sánh định dạng tệp tin với danh sách định dạng hợp lệ. Nếu định dạng tệp tin nằm trong danh sách hợp lệ, hàm tiếp tục thực hiện

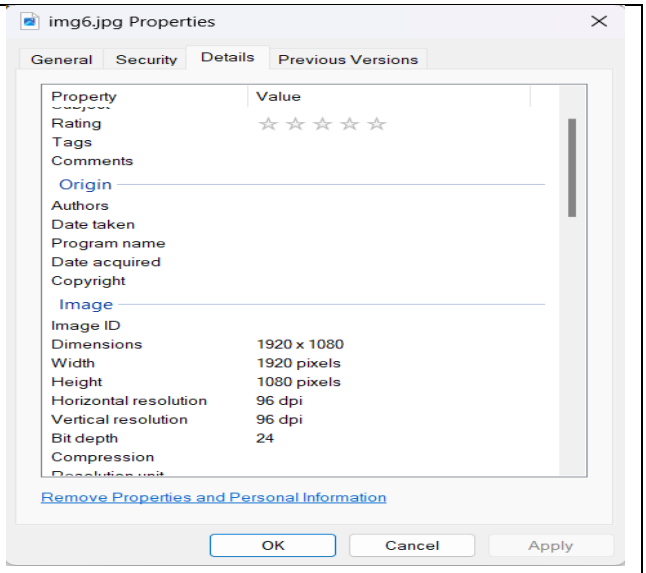
bước lưu ảnh. Ngược lại, hàm chuyển đổi định dạng tệp tin về .png (một định dạng tệp tin hợp lệ) và tiếp tục thực hiện bước lưu ảnh.

- Lưu ảnh (img) với tên tệp đầu ra (name) bằng cách sử dụng phương thức save của đối tượng ảnh.
- Kết thúc hàm.

- Mô tả hàm **main:**
  - Các bước trong hàm này:
    - Yêu cầu người dùng nhập tên tệp ảnh để xử lý (image_path).
    - Yêu cầu người dùng nhập số lượng clusters để xử lý.
    - Yêu cầu người dùng nhập max_iter để xử lý.
    - Yêu cầu người dùng chọn init_method là 'random' hoặc 'in_pixels'.
    - Yêu cầu người dùng nhập tên tệp để lưu ảnh sau khi giảm màu (save_name).
    - Gọi hàm reduce_color(image_path, k_clusters, max_iter, init_method) để giảm màu ảnh.
    - Hiển thị ảnh gốc và ảnh đã giảm màu bên cạnh nhau bằng cách sử dụng các trục (axs) trong một hình vẽ.
    - Hiển thị đồ họa bằng thư viện matplotlib.
    - Gọi hàm save_image(reduced_image, save_name) để lưu ảnh đã giảm màu với tên tệp đầu ra đã nhập từ người dùng.

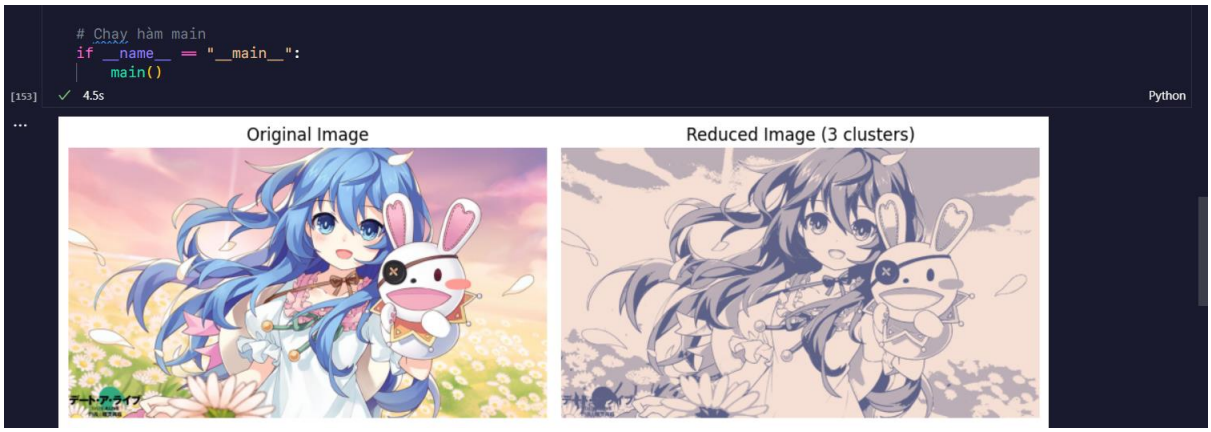
3. Hình ảnh kết quả + Thời gian chạy

- Hình ảnh minh chứng thời gian chạy của em để chứng minh thời gian chạy, nên nó sẽ khác với hình ảnh đã lưu (tức là ảnh đã qua xử lý).
- Thời gian chạy không bao gồm cả thời gian lưu ảnh và thời gian xử lý xuất ảnh ra 1 cell output nên sẽ chênh lệch ~2s. Nếu có lưu ảnh thời gian xuất sẽ chậm hơn ~3s.
- Thời gian chạy còn phụ thuộc vào cấu hình máy tính (vi xử lý, ram, GPU, …)

	
Hình ảnh được sử dụng	*Thông số hình ảnh được sử dụng*

- *Trường hợp random*:
    - *Với k_clusters = 3*:

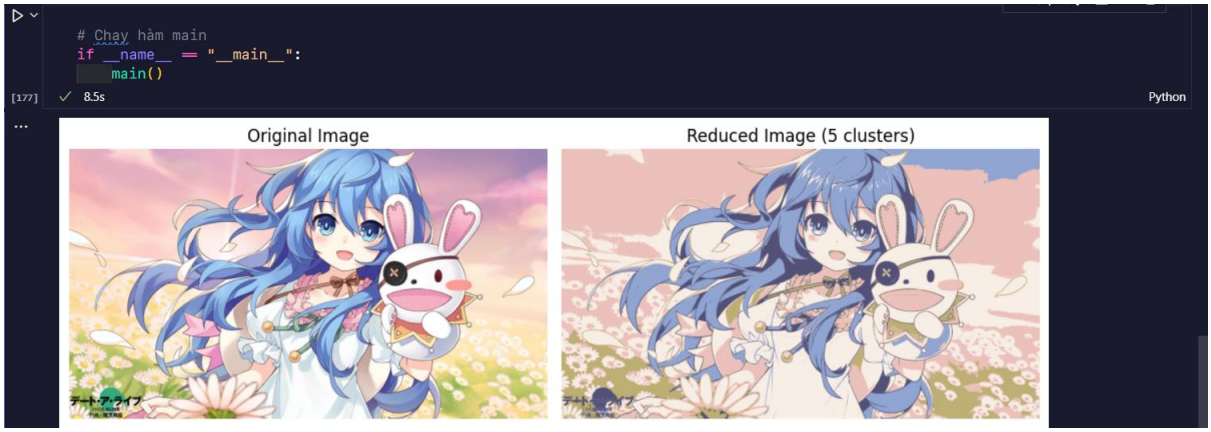
Ảnh đã qua xử lý với k=3



Minh chứng thời gian chạy bằng 4.5s

- *Với k_clusters = 5*:

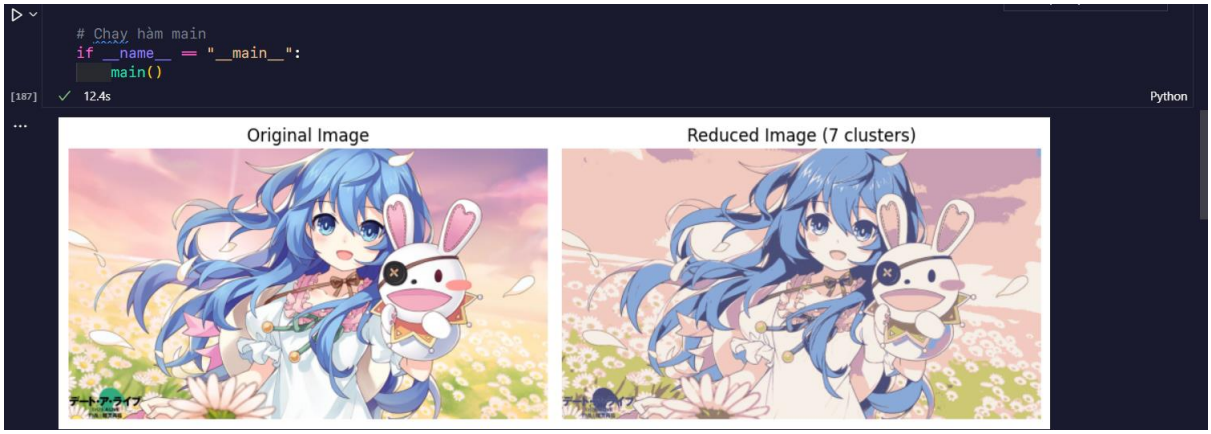
Ảnh đã qua xử lý với k=5



Minh chứng thời gian chạy bằng 8.5s

- *Với k_clusters = 7*:

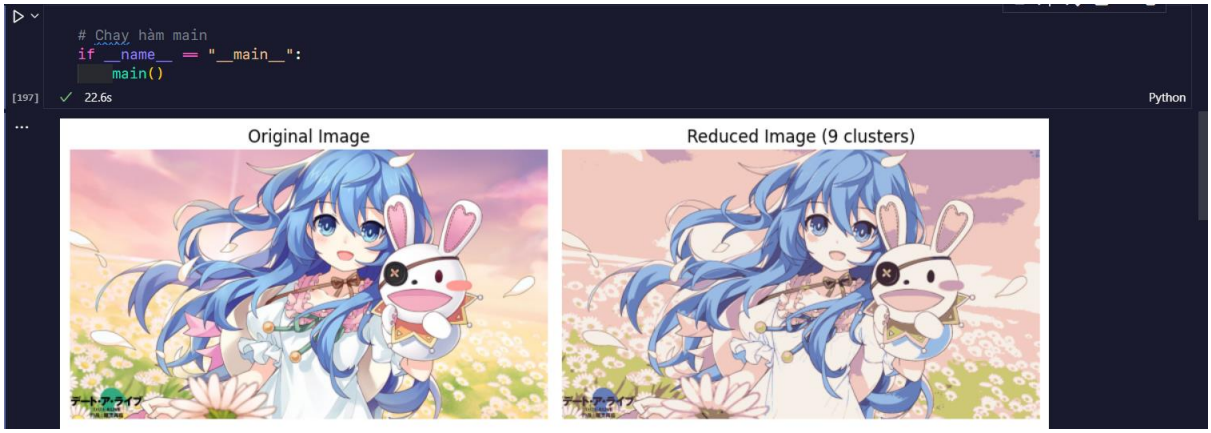
Ảnh đã qua xử lý với k=7



Minh chứng thời gian chạy bằng 12.4s

- *Với k_clusters = 9*:

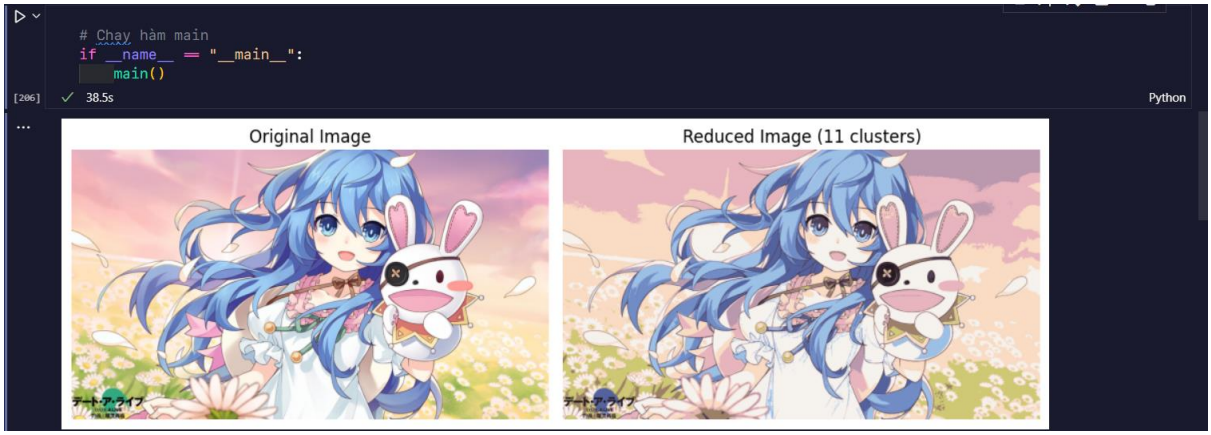
Ảnh đã qua xử lý với k=9



Minh chứng thời gian chạy bằng 22.6s

- *Với k_clusters =11*:

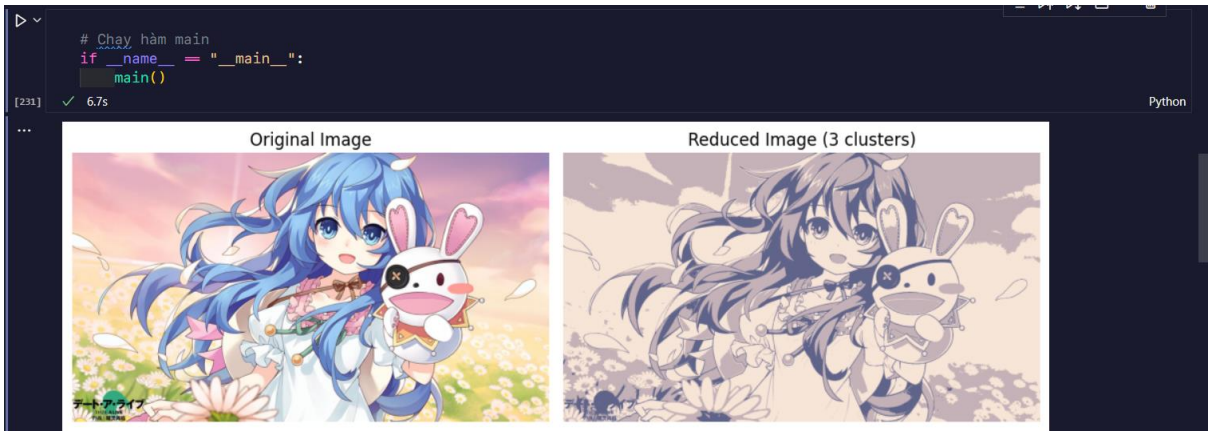
Ảnh đã qua xử lý với k=11



Minh chứng thời gian chạy bằng 38.5s

- *Trường hợp in_pixels*:
    - *Với k_clusters =3:*

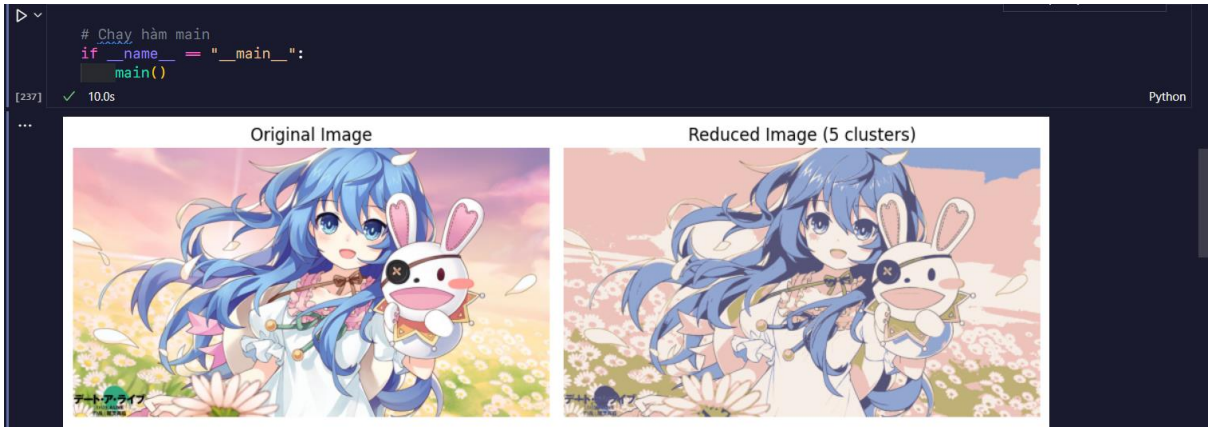
Ảnh đã qua xử lý với k=3



Minh chứng thời gian chạy bằng 6.7s

- *Với k_clusters =5:*

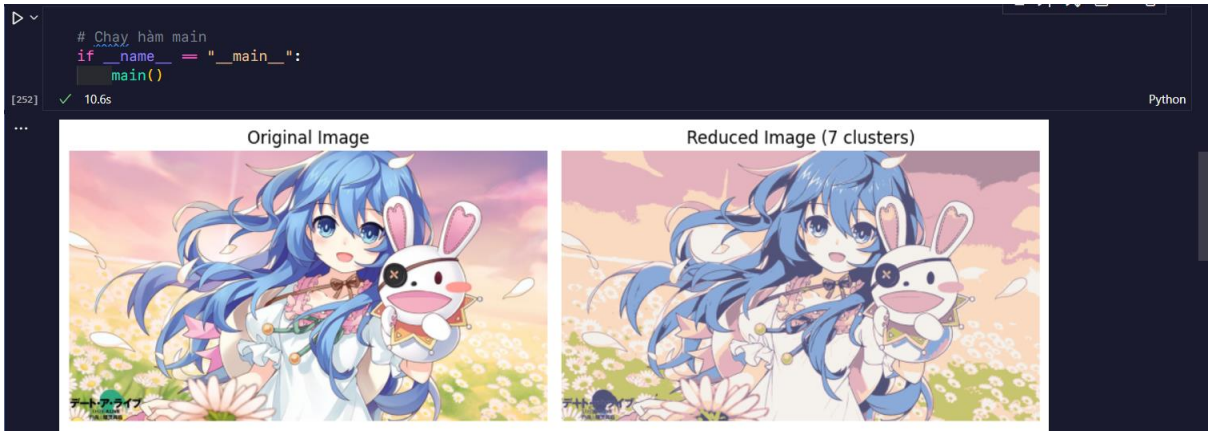
Ảnh đã qua xử lý với k=5



Minh chứng thời gian chạy bằng 10.0s

- *Với k_clusters =7:*

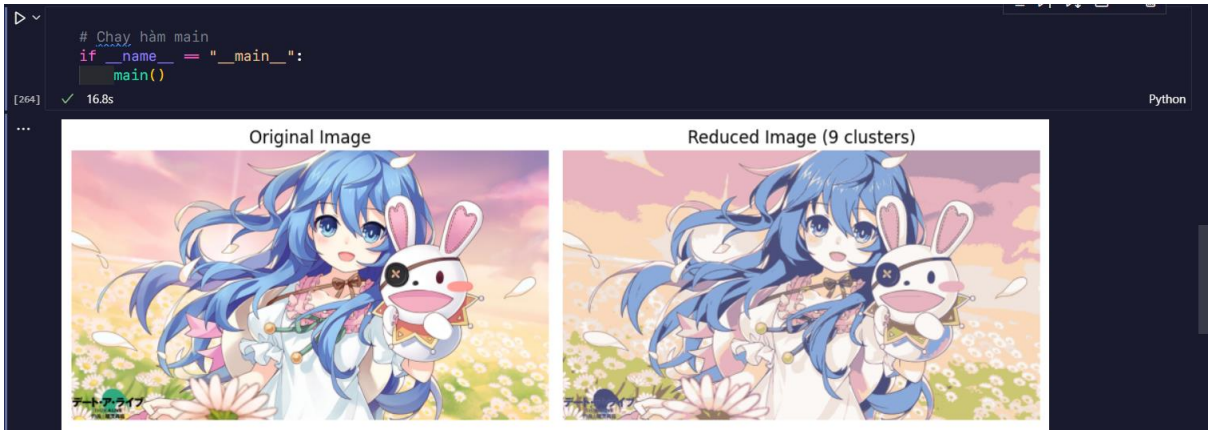
Ảnh đã qua xử lý với k=7



Minh chứng thời gian chạy bằng 10.6s

- *Với k_clusters =9:*

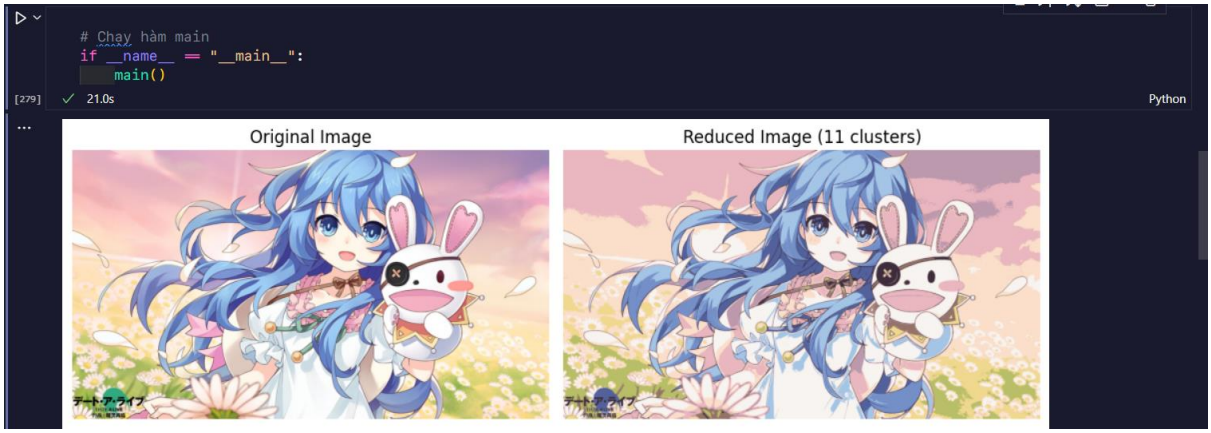
Ảnh đã qua xử lý với k=9



Minh chứng thời gian chạy bằng 16.8s

- *Với k_clusters =11:*

Ảnh đã qua xử lý với k=11



Minh chứng thời gian chạy bằng 21.0s

4. Nhận xét kết quả trên

	Random	In Pixels	Thời gian tính bằng giây
K=3	4.5	6.7	
K=5	8.5	10.0	
K=7	12.4	10.6	
K=9	22.6	16.8	
K=11	38.5	21.0	

- Dựa vào bảng kết quả bên trên, ta có thể thấy:
    - ***Đầu tiên***, thời gian chạy của **random** nhanh hơn so với thời gian chạy của **in_pixels**. Lí do là vì việc khởi tạo **random** các centroid không phụ thuộc vào dữ liệu ảnh và không đòi hỏi tính toán phức tạp, nó đơn giản chỉ là chọn từ 0 đến 255. Ưu điểm của khởi tạo **random** là tốc độ chạy rất nhanh, đặc biệt khi số lượng **k_clusters** lớn và kích thước ảnh lớn. Mặc dù vậy, khởi tạo **random** cũng có điểm yếu riêng, đó là nó có thể dẫn đến việc các centroid khởi tạo ban đầu không tốt (vì nó lựa chọn **random** từ 0 đến 255 các điểm ảnh), vì điều này nên đôi khi phải cần nhiều vòng lặp hơn để đạt được kết quả tốt, kết quả tối ưu nhất. Đối với khởi tạo dựa trên các điểm trong ảnh gốc (tức là **in_pixels**) thì lại tốn nhiều thời gian chạy hơn một chút nên đây chính là nhược điểm chính của khởi tạo **in_pixels**. Lí do là vì nó phải chọn các centroid từ các pixel ngẫu nhiên trong ảnh mà còn phải là chọn **DUY NHẤT** nên phải yêu cầu tính toán và so sánh nhiều hơn. Dù vậy điểm mạnh của **in_pixels** chính là khởi tạo các centroid từ các pixel thực tế trong ảnh có thể tạo ra các centroid khởi tạo ban đầu tốt hơn, đối với số lượng **k_cluster** lớn thì càng tốt.
    - Đối với màu sắc của ảnh đã qua xử lý, thì ta có thể thấy do khởi tạo **random** có các centroid được khởi tạo ngẫu nhiên trong phạm vi từ 0-255 nên việc màu sắc của ảnh được xử lý theo cách không được đồng nhất và không liên quan trực tiếp đến màu sắc ban đầu của ảnh. Em thấy kết quả xử lý của khởi tạo **random** có thể mang lại sự đa dạng và ngẫu nhiên trong màu sắc của ảnh nhưng một số chỗ, một số khu vực nhỏ trong ảnh có thể có màu sắc khác nhau và nó không liên tục, đồng đều.
    - Đối với **in_pixels**, vì centroid được khởi tạo từ điểm ảnh của ảnh gốc nên nó sẽ đảm bảo được việc màu sắc của các centroids sẽ gần giống nhất với màu sắc ban đầu của ảnh. Khi ra được kết quả thường sẽ giữ lại được nhiều đặc trưng riêng của ảnh - đặc trưng về màu sắc ban đầu của ảnh. Màu sắc sẽ được chia ra từng khu vực, mỗi khu vực có màu sắc tương tự nhau sẽ được gom lại thành các cụm màu, giúp duy trì tính chất màu sắc chính thống của ảnh.

→ Nhìn tổng quan thì phương pháp khởi tạo **random** thường nhanh hơn so với phương pháp khởi tạo **in_pixels**. Tuy nhiên, phải cần lưu ý ở chỗ thời gian chạy có thể khác nhau tùy thuộc vào kích thước ảnh và số lượng **k_clusters**. Đối với các ảnh lớn và số lượng **k_clusters** lớn thì phương pháp **random** có thể được ưu tiên hơn vì nó sẽ giảm thời gian chạy. Nhưng mà nếu yêu cầu kết quả phải tốt, phải đẹp và gần giống ảnh gốc

từ các centroid ban đầu thì phương pháp **in_pixels** lại được sử dụng (mặc dù nó tốn nhiều thời gian hơn).

- ***Tiếp theo***, em sẽ so sánh về thời gian của từng cluster. Vì việc chọn **k_clusters** trong thuật toán này(K-Means) có ảnh hưởng rất nhiều đến quá trình phân cụm và ảnh hưởng đến kết quả cuối cùng.

5. Điểm yếu bản thân

- Sau khi thực hiện đồ án này, em nhận ra được một số vấn đề mình cần cải thiện:
  - Về phần thuật toán, em cảm thấy hàm **kmeans** cần được tối ưu hơn nữa để có thể xuất ra kết quả với thời gian ít hơn.
  - Về phần trình bày code, em không biết nên phân chia từng phần nhỏ thành những hàm nhỏ hay không nên em để chung vào một hàm → Về mặt này thì em cảm thấy mình cần phân chia code sao cho hợp lí nhất.

→ Cuối cùng, sau khi thực hiện đồ án này, em cảm ơn cô Phạm Thị Phương Uyên, thầy Vũ Quốc Hoàng, thầy Nguyễn Văn Quang Huy, thầy Lê Thanh Tùng đã cho em được làm đồ án này để học hỏi được thêm kinh nghiệm. Em cảm ơn thầy cô rất nhiều. Về phần báo cáo, em đã chia từng mục nhỏ thành Bookmark để thầy cô đọc dễ dàng. Một lần nữa em gửi mọi điều tốt đẹp đến với thầy cô.

TÀI LIỆU THAM KHẢO

- *Tài liệu của giáo viên*
- *Image compression & Color Quantization using K Means | k-means Clustering in Unsupervised ML by Data Science World from Youtube*: https://www.youtube.com/watch?v=shu4pYQb-ps&t=560s&pp=ygUdY29sb3IgY29tcHJlc3Npbmcgd2l0aCBrbWVhbnM%3D
- *KMeans Clustering Algorithm for colors Detection in images by Shivansh Amattya from Youtube*: https://www.youtube.com/watch?v=wULZ49c1atg&pp=ygUdY29sb3IgY29tcHJlc3Npbmcgd2l0aCBrbWVhbnM%3D
- *Python k-means clustering colors - micro demo by Katherine Cottrell from Youtube:* https://www.youtube.com/watch?v=I41LsU8KGA4&pp=ygUdY29sb3IgY29tcHJlc3Npbmcgd2l0aCBrbWVhbnM%3D
- *CS 320 May 1 (Part 1) - K-Means for Color Segmentation by Tyler Caraza-Harter from Youtube:* https://www.youtube.com/watch?v=fOd8Rl9BC2k&t=519s&pp=ygUdY29sb3IgY29tcHJlc3Npbmcgd2l0aCBrbWVhbnM%3D

- *K-means Clustering From Scratch In Python [Machine Learning Tutorial] by Data Quest from Youtube:* https://www.youtube.com/watch?v=lX-3nGHDhQg&pp=ygUdY29sb3IgY29tcHJlc3Npbmcgd2l0aCBrbWVhbnM%3D
- *kmeans-experiments by TassaraR from Github:* https://github.com/TassaraR/kmeans-experiments.git
- *Machine-Learning-University-of-Washington-Clustering-Retrieval by aspiringguru from Github:* https://github.com/aspiringguru/Machine-Learning-University-of-Washington-Clustering-Retrieval.git
- *Image compression using K-means clustering from Geeksforgeeks:* https://www.geeksforgeeks.org/image-compression-using-k-means-clustering/
- *Image Compression using K-Means Clustering by Satyam Kumar from TowardsDataScience:* https://towardsdatascience.com/image-compression-using-k-means-clustering-aa0c91bb0eeb
- *Beginners Guide to Image Compression using K-Means Clustering by Himanshu Sharma from Analyticsindiamag:* https://analyticsindiamag.com/beginners-guide-to-image-compression-using-k-means-clustering/
- *Image Compression With K-Means from Epoch:* https://epoch.aisingapore.org/2023/02/image-compression-with-k-means/

---HẾT---