ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

MÔN : TOÁN ỨNG DỤNG VÀ THỐNG KÊ

CHO CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN 1 :

COLOR COMPRESSION

GIẢNG VIÊN: NGUYỄN ĐÌNH THÚC

NGÔ ĐÌNH HY

NGUYỄN VĂN QUANG HUY

SINH VIÊN THỰC HIỆN: NGUYỄN THÁI ĐAN SÂM – 21127414 – 21CLC01

Mục lục

I. Ý tưởng thuật toán.....3

II. Cài đặt chương trình.....5

III. Các test case.....6

IV. Nhận xét và đánh giá.....14

V. Tham khảo.....15

Phần I. Ý tưởng thuật toán

Một bức ảnh có thể lưu trữ dưới ma trận của các điểm ảnh. Có nhiều loại ảnh được sử dụng trong thực tế, ví dụ: ảnh xám, ảnh màu,...

Đối với ảnh xám, một điểm ảnh sẽ là được biểu diễn bằng giá trị không âm.
Ảnh màu được sử dụng phổ biến là ảnh RGB, trong đó, mỗi điểm ảnh sẽ lưu trữ 3 thông tin kênh màu (mỗi kênh màu 1 byte) là: R (red - đỏ), G (green - xanh lá), B (blue - xanh dương).

Như vậy, số màu trong ảnh RGB có thể là $256^3 \approx 1.7 \times 10$. Với số lượng màu khá lớn, khi lưu trữ ảnh có thể sẽ tốn chi phí lưu trữ. Do đó bài toán đặt ra là giảm số lượng màu để biểu diễn ảnh sao cho nội dung ảnh được bảo toàn nhất có thể.

Để thực hiện giảm số lượng màu, ta cần tìm ra các đại diện có thể thay thế cho một nhóm màu. Cụ thể trong trường hợp ảnh RGB, ta cần thực hiện gom nhóm các pixel R^3 và chọn ra đại diện cho từng nhóm. Như vậy, bài toán trên trở thành gom nhóm các vec-tơ.

Như vậy, để giải quyết bài toán này, ta sử dụng thuật toán phân cụm K – Means để chọn ra những màu trung tâm (centroid) làm đại diện cho toàn bộ màu trong các pixel đó

1. Xử lý ảnh ban đầu

Biến ảnh màu thành ma trận các pixel, mỗi pixel là 1 list chứa 3 giá trị nguyên dương từ 0 đến 255 đại diện cho 3 màu RBG

2. Dùng thuật toán phân cụm K – Means

Đầu tiên, chọn số lượng màu cho bức ảnh theo ý muốn của người dùng, số lượng màu ở đây chính là k, dùng để phân cụm pixel

Chọn ngẫu nhiên k pixel để làm centroid, có thể chọn theo 2 cách : 1 là chọn ngẫu nhiên các pixel từ 0 đến 255, 2 là chọn các pixel từ trong chính ảnh gốc, lưu ý với trường hợp 1, pixel có thể không nằm trong ảnh

Sau đó tính khoảng cách Euclid từ các pixel x(x_R, x_B, x_G) đến các centroid C(C_R, C_B, C_G), với công thức như sau

$$\text{Distance} = \sqrt{(C_R - x_R)^2 + (C_G - x_G)^2 + (C_B - x_B)^2}$$

Chia các pixel thành từng cụm cùng với centroid gần nó nhất rồi cập nhật lại centroid bằng cách tính trung bình cộng các pixel trong từng cụm của nó

Lặp lại quá trình tính khoảng cách như đã nêu trên cho đến khi các pixel không có sự thay đổi cụm nữa, có thể giới hạn số lần lặp tránh trường hợp thuật toán chạy quá lâu

3. Giảm màu ảnh

Thay thế các pixel trong từng cụm bằng các centroid của cụm đó. Ta được bức ảnh mới có đúng k màu sắc như đã yêu cầu

Phần II. Cài đặt thuật toán

1. Các thư viện hỗ trợ

`numpy :` tính toán trên ma trận, dùng nhiều trong cài đặt thuật toán K - Means
`matplotlib.pyplot :` Đọc, xuất hình ảnh và đặt title cho ảnh
`PIL :` Mở file ảnh và biến đổi hình ảnh thành ma trận pixel

2. Các hàm trong chương trình

`def remake_img(img) :` Biến đổi hình ảnh thành 1 mảng các pixel mà mỗi pixel đại diện cho 1 giá trị RBG. Mảng sau khi biến đổi sẽ được trả về dưới dạng đầu ra của hàm.

`def color_img(centroids, labels, img) :` "Tô màu" lại cho hình ảnh bằng cách thay thế từng giá trị pixel bằng centroids và labels mà nó thuộc về. Sau đó đưa hình ảnh về kích thước ban đầu (Reshape) và trả về hình ảnh sau khi biến đổi dưới dạng đầu ra của hàm

`def k_means(img_1d, k_clusters, max_iter, init_centroids) :` Áp dụng thuật toán K – Means, nhận vào 1 mảng pixel (img_1d), số clusters (k_clusters), số lần lặp cho thuật toán (max_iter) và kiểu centroid đã chọn

- **Bước 1**: Với kiểu centroid người dùng đã chọn, hàm sẽ tiến hành tạo centroid. Có 2 kiểu centroid (**`random`** : từ 0 đến 255 và **`in_pixels`** : chọn ngẫu nhiên pixel từ hình ảnh)
- **Bước 2**: Sau đó tiến hành tính trung bình cộng giá trị khoảng cách Euclide giữa các pixel và centroid, trong mỗi vòng lặp, sau khi tính xong, khoảng cách nhỏ nhất nhất và k_cluster tương ứng sẽ được lưu lại nếu như centroid gần đó được tìm thấy. k_cluster đó sẽ được gán cho pixel bằng cách "dán nhãn (labels)"
- **Bước 3**: Sau khi tính xong, tiến hành cập nhật lại centroid dựa theo các pixel mới được gán ở bước trên, lặp đi lặp lại cho đến khi đạt đến số lần lặp tối đa (max_iter), sau khi tính toán xong, hàm sẽ trả về 2 mảng centroid và label

`def main() :` Cho phép người dùng nhập tên ảnh (lưu ý nhập cả đuôi tên file, vd: my_image.jpg), số k_cluster (số màu), số max_iter (số lần lặp), chọn kiểu centroid (centroid_type), tiếp theo là các bước xử lí hình ảnh (mở ảnh, xử lí ảnh, tính toán K_Means) và cuối cùng là hiển thị và chọn định dạng ảnh muốn lưu (png, pdf)

Phần III. Các test case

1.In_pixel centroid

Trường hợp 1.1 : Số màu cố định, số lần lặp thay đổi

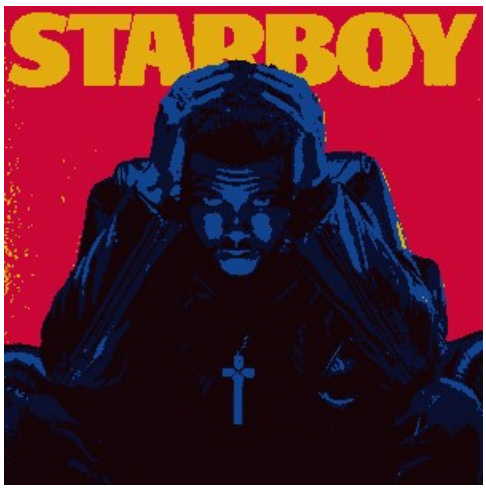
Ảnh 1:

Độ phân giải: 300 x 300 pixel

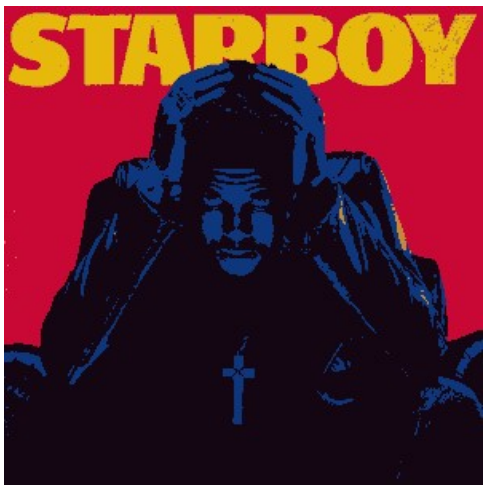
Số màu: 5

Ảnh gốc

Lặp 1 lần



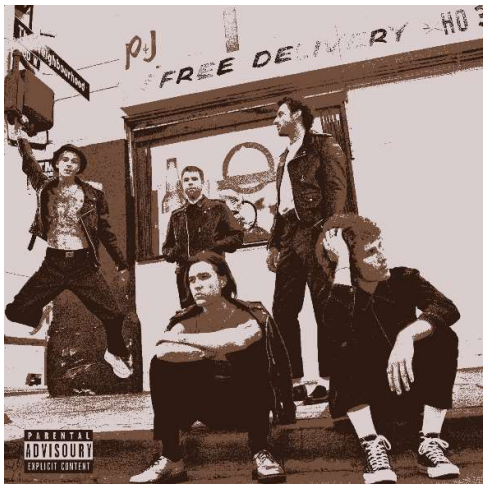
Lặp 5 lần



Lặp 7 lần

Ảnh 2:
Độ phân giải: 1400 x 1400 pixel
Số màu: 3

Ảnh gốc



Lặp 1 lần

Lặp 5 lần

Lặp 20 lần

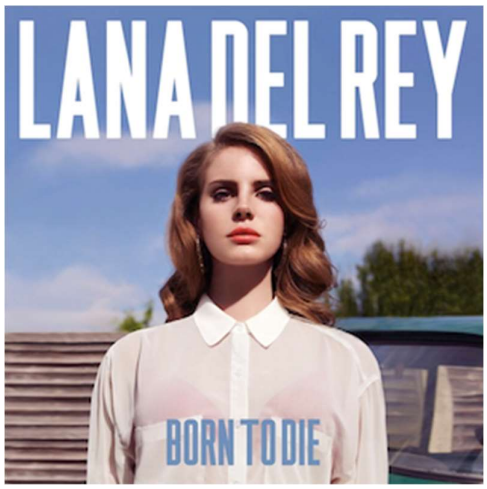
Nhận xét : Có thể thấy, số lần lặp (số lần phân cụm) càng ít thì ảnh cho ra càng mờ, nhòe, các chi tiết sẽ không rõ ràng hoặc có thể những chi tiết nhỏ sẽ bị mất đi, như ở Ảnh 1 với 5 màu và 1 lần lặp, ảnh cho ra không thể thấy rõ mặt người. Nên nếu tăng số lần lặp thì ảnh cho ra sẽ càng rõ và nét hơn, tuy nhiên thời gian xử lí cũng sẽ tăng lên theo đó.

Trường hợp 1.2 : Số màu thay đổi, số lần lặp cố định

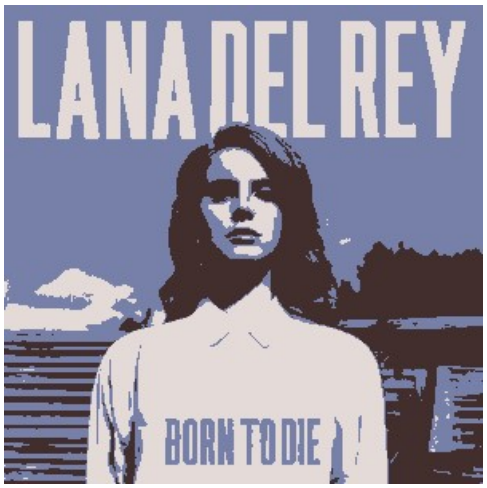
Ảnh 3:

Độ phân giải: 300 x 300 pixel

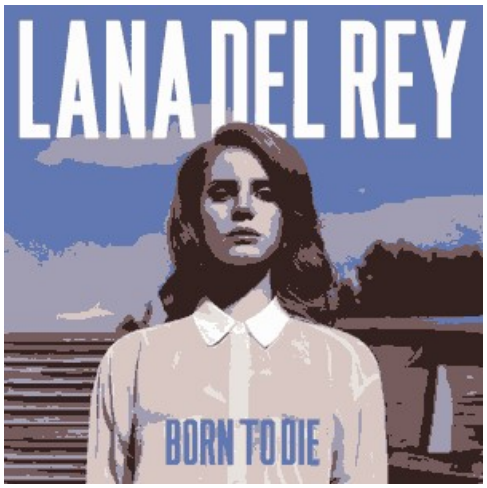
Số lần lặp: 5



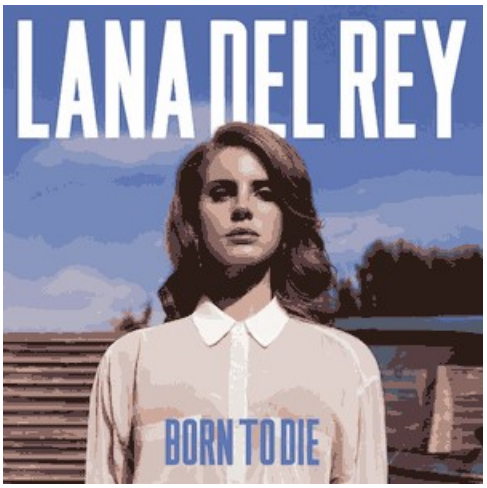
Ảnh gốc



3 màu



7 màu



20 màu

Ảnh 4:
Hệ màu RBG
Độ phân giải: 300 x 300 pixel
Số lần lặp: 10

Ảnh gốc

5 màu

10 màu

20 màu

Nhận xét : Với cùng số lần lặp, số lượng màu càng ít (khoảng từ 3 đến 5), ảnh sẽ bị nhòe đi đáng kể, các chi tiết cũng sẽ không rõ hoặc mất đi. Khi tăng số lượng màu lên, bức ảnh cho ra sẽ có chất lượng cao, gần giống với ảnh gốc (Ảnh 4 với 20 màu). Tuy nhiên điều này cũng sẽ làm tăng thời gian thực hiện thuật toán lên.

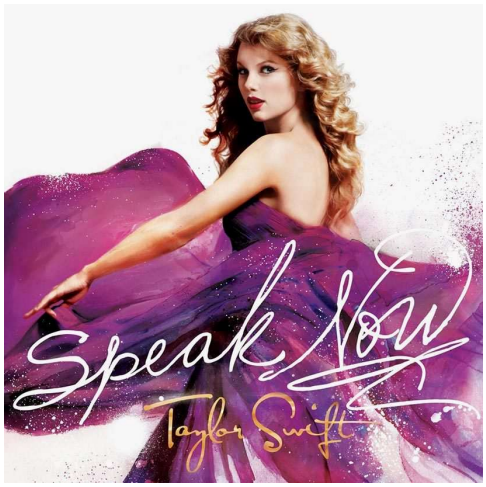
2.Random centroid

Trường hợp 2.1 : Số màu cố định, số lần lặp thay đổi

Ảnh 5:

Độ phân giải: 820 x 820 pixel

Số màu: 5



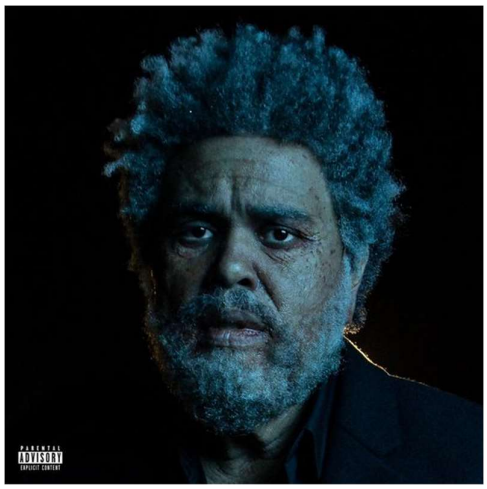
Ảnh gốc

Lặp 1 lần

Lặp 3 lần

Lặp 7 lần

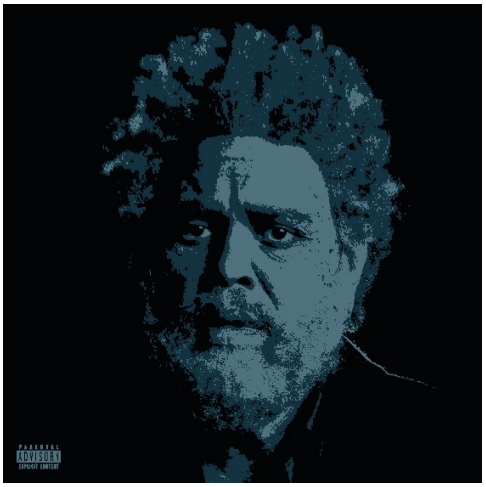
Ảnh 6:
Độ phân giải: 564 x 564 pixel
Số màu: 3



Ảnh gốc

Lặp 3 lần

Lặp 7 lần



Lặp 20 lần

Nhận xét : Có thể thấy số lần lặp không ảnh hưởng nhiều đến chất lượng bức ảnh, thay vào đó màu sắc sẽ là yếu tố quyết định nhiều hơn. Ở mỗi lần lặp, các bức ảnh cho ra đều có chất lượng như nhau(độ sắc nét, chi tiết,...) nhưng có sự khác biệt rõ rệt về màu sắc và có trường hợp không thể nhìn ra màu ở ảnh gốc. Lấy ví dụ Ảnh 5(lặp 7 lần), ta không thể nhìn ra màu tóc của cô gái nhưng trường hợp(lặp 1 lần)thì có.

Trường hợp 2.2 : Số màu thay đổi, số lần lặp cố định

Ảnh 7:
Độ phân giải: 1920 x 2560 pixel (width x height)
Số lần lặp: 3

Ảnh gốc

3 màu

5 màu

7 màu

Ảnh 8:
Độ phân giải: 800 x 800 pixel
Số lần lặp: 5

Ảnh gốc

3 màu

5 màu

10 màu

<u>Nhận xét</u> : Với cùng số lần lặp, tăng số lượng màu lên thì bức ảnh cho ra sẽ càng rõ ràng cả về màu sắc và chi tiết, có thể dễ dàng nhìn ra được nhiều chi tiết hơn. Điều này dĩ nhiên cũng sẽ tiêu tốn nhiều thời gian xử lí hơn.

PHẦN IV. Kết luận và đánh giá

1. **Chất lượng hình ảnh**
   - Tùy vào số lượng màu sắc mà độ chi tiết, sắc nét của mỗi bức ảnh sẽ khác nhau. Với số lượng màu sắc nhỏ, khi thực hiện thuật toán xong ảnh có thể sẽ bị mất đi vài chi tiết, vài mảng màu sẽ không giống như ảnh gốc. Với số lượng màu lớn, ảnh sẽ chi tiết hơn, các mảng màu chính xác hơn và khi số lượng màu đủ lớn, ảnh cho ra sẽ gần giống như ảnh gốc, không có nhiều sự khác biệt.
   - Số lần lặp (số lần phân cụm) quyết định độ chi tiết cho bức ảnh, có thể thấy rõ ràng nhất ở những trường hợp có centroid in_pixel (số màu cố định, số lần lặp thay đổi), số lần lặp càng nhiều, ảnh cho ra sẽ càng chi tiết.
   - Tuy nhiên không thể tránh khỏi những sự sai lệch về màu sắc của bức ảnh khi thực hiện thuật toán. Ví dụ ở trường hợp Random centroid, thuật toán chọn ngẫu nhiên các centroid có màu gần như giống nhau nên khi kết quả cho ra, ta có thể dễ dàng thấy các bức ảnh có màu rất hạn chế, có trường hợp không thể nhìn ra màu ảnh gốc (như đã nhận xét). Nguyên nhân là do ở bước chọn centroid ngẫu nhiên, thuật toán có thể đã chọn những centroid không quá khác biệt nên mới gây ra sai lệch này, nếu chọn centroid đủ tốt, hình ảnh cho ra có thể sẽ khả quan hơn.

2. **Thời gian thực hiện thuật toán**

   Thời gian thực hiện thuật toán phụ thuộc chủ yếu vào thuật toán K – Means. Bên cạnh đó, các yếu tố sau cũng ảnh hưởng ít nhiều đến thời gian thực hiện thuật toán
   - Độ phân giải của ảnh : Độ phân giải càng cao, thời gian xử lí càng lâu
   - Số lượng màu sắc : Khi tăng số lượng màu sắc, thuật toán sẽ mất nhiều thời gian để tính toán hơn
   - Số lần lặp (phân cụm) : Số lần lặp càng lớn, thuật toán chạy càng lâu

3. **Kích thước lưu trữ**

   Màu sắc là yếu tố ảnh hưởng đến kích thước lưu trữ do nó là số lượng thông tin để biểu thị hình ảnh. Qua các lần xử lý hình ảnh, nhận thấy các file ảnh có sự chênh lệch về hình ảnh thì sẽ có sự chênh lệch về kích thước, còn chênh lệch số lần lặp thì không có sự khác biệt quá rõ rệt.

PHẦN V. Tham khảo

[1] Lab02 – Ma trận trong Python, Ngô Đình Hy, Nguyễn Văn Quang Huy

[2] Howard Anton, Chris Rorres. Elementary Linear Algebra - Applications Version. Wiley, 11th Edition, Chapter 3, Chapter 10.

[3] K-means Clustering – Machine Learning cơ bản
https://machinelearningcoban.com/2017/01/01/kmeans/

[4] Machine Learning – K–means – w3schools
https://www.w3schools.com/python/python_ml_k-means.asp

[5] In Depth: k-Means Clustering | Python Data Science Handbook
https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html

[6] Image Segmentation using K Means Clustering – GeeksforGeeeks
https://www.geeksforgeeks.org/image-segmentation-using-k-means-clustering/

[7] Elbow Method for optimal value of k in Kmeans – GeekforGeeks
https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

[8] Image compression & Color Quantization using K Means | k-means Clustering in Unsupervised ML – Youtube Data Science World
https://www.youtube.com/watch?v=shu4pYQb-ps&ab_channel=DataScienceWorld

[9] Image Segmentation with K-Means Clustering in Python – Youtube NeuralNine
https://www.youtube.com/watch?v=X-Y91ddBqaQ&ab_channel=NeuralNine

[11] how k means algorithm achieve color compression – stackoverflow
https://stackoverflow.com/questions/58712386/how-k-means-algorithm-achieve-color-compression

[11] color-compression – Github Topics
https://github.com/topics/color-compression