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

BÁO CÁO ĐỒ ÁN - COLOR COMPRESSION

TOÁN ỨNG DỤNG VÀ THỐNG KÊ

HỌ VÀ TÊN: BÙI ĐỖ DUY QUÂN

MÃ SỐ SINH VIÊN: 21127141

LỚP: 21CLC02

Giảng viên hướng dẫn:
Phan Thị Phương Uyên

Ngày 17 tháng 7 năm 2023

Mục lục

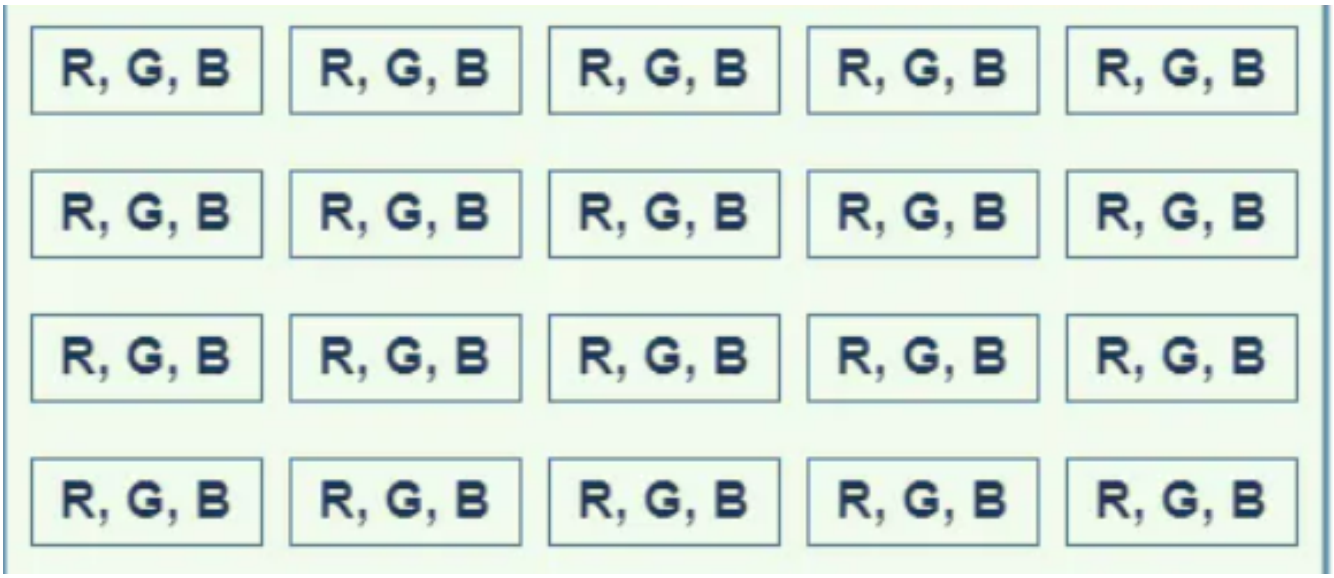
1 YÊU CẦU CỦA ĐỒ ÁN

- Yêu cầu cài đặt chương trình giảm số lượng màu cho ảnh bằng cách sử dụng thuật toán **K-Means**
- Các thư viện được sử dụng: **Numpy**, **PIL(đọc, ghi ảnh)**, **matplotlib(hiển thị ảnh)**

2 Ý TƯỞNG THỰC HIỆN

2.1 Điểm ảnh của 1 bức ảnh

- Qua các bài học trên lớp lý thuyết và lớp thực hành của môn, em đã biết được bản chất 1 bức ảnh có cấu tạo là 1 ma trận với các phần tử là giá trị của điểm ảnh (ma trận của các điểm ảnh).
- Tuỳ vào màu mà bức ảnh đó mà các điểm ảnh có giá trị khác nhau như thế nào. Như trong ví dụ của đồ án, ảnh xám có các điểm ảnh sẽ biểu thị 1 giá trị không âm có giá trị từ [0-255].
- Đối với ảnh màu, trong trường hợp này ta sẽ xét ảnh màu **RGB**, mỗi điểm ảnh của ma trận sẽ chứa 3 giá trị của lần lượt 3 loại màu (đỏ-**R**ed, xanh lá cây-**G**reen, xanh dương-**B**lue). Tức là ma trận của ảnh màu **RGB** sẽ là ma trận của các ma trận chứa 3 giá trị của màu đỏ, xanh lá cây, và xanh dương.



Hình 1: Ma trận các điểm ảnh của ảnh RGB

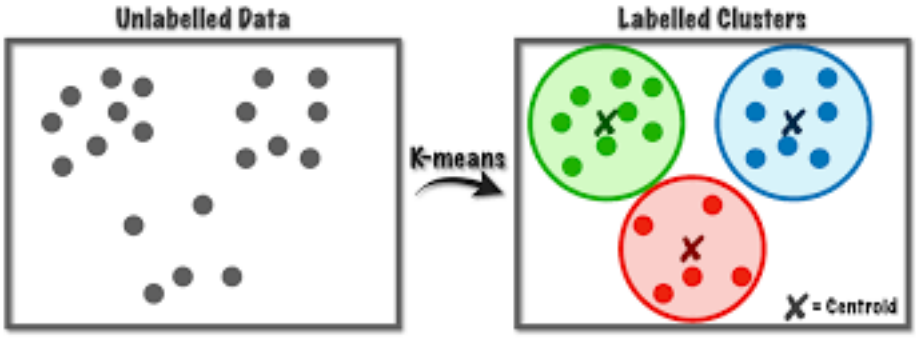
2.2 Đối chiếu kiến thức điểm ảnh vào yêu cầu đồ án

- Mục đích của đồ án này chính là làm GIẢM SỐ LƯỢNG MÀU BAN ĐẦU của ảnh xuống **K** màu mà người dùng mong muốn nhưng vẫn BẢO TOÀN NỘI DUNG CỦA ẢNH. Có thể thấy trong 1 ảnh có rất nhiều giá trị màu, vậy câu hỏi đặt ra là: ***Làm thế nào để có thể chọn được K màu đại diện để biểu diễn bức ảnh đó?***
- Như đã được phân tích trên lớp, trong ảnh sẽ có những màu có cùng loại màu với nhau nhưng có thể khác về mức độ biểu thị. Giả sử: trong bức ảnh sẽ có khu vực có màu xanh dương nhạt, màu xanh dương đậm,... Nhưng bản chất các màu này đều là loại xanh dương, vậy có thể phân tích rằng, việc chọn ra màu đại diện thực chất là chọn màu trung bình của các nhóm màu có trong bức ảnh đó.
- Từ phân tích trên, câu hỏi được đặt ra đó là: ***Làm sao để tạo nhóm và phân các màu về từng nhóm?*** Dựa vào gợi ý về thuật toán trong đồ án này thì thuật toán phân cụm K-Means (**K-Means clustering**) sẽ được áp dụng.

2.3 Thuật toán phân cụm K-Means

a Thuật toán phân cụm K-Means là gì?

Có thể tóm tắt thuật toán K-Means là kĩ thuật **lượng tử hoá vector**, mục đích của thuật toán là sẽ chia 1 tập hợp lớn dữ liệu thành các nhóm và các nhóm này sẽ là tập hợp những điểm mà có cùng số điểm (về giá trị, hoặc khoảng cách,...) gần nhất với điểm trung tâm của nhóm.



Hình 2: Hình ảnh thuật toán K-Means hoạt động

b Lý do áp dụng thuật toán này vào đồ án

- Thuật toán K-Means rất phù hợp cho mục đích giảm số lượng màu của 1 ảnh. Như đã phân tích ở trên về cách giảm số lượng màu, chúng ta phải thực hiện phân nhóm các điểm ảnh về nhóm màu mà mình muốn giảm xuống, sau đó chọn ra 1 đại điện màu trong nhóm đó để làm màu đại diện cho các màu khác trong nhóm. Nếu ta áp dụng thuật toán K-Means, ta sẽ xem dữ liệu cần phân nhóm là giá trị các điểm ảnh và số lượng nhóm cần chia là số lượng màu mà người dùng cần giảm xuống tới.
- Ban đầu chúng ta sẽ thực hiện chia ra **K** nhóm màu, chọn màu đại diện cho nhóm đó. Nếu chúng ta coi 1 điểm ảnh là 1 điểm trong toạ độ, thì ta sẽ quyết định chia màu về nhóm mà khoảng cách từ điểm ảnh này đến trung tâm của nhóm màu đó là nhỏ nhất.
- Từ đó, ta có thể chọn được giá trị trung bình của nhóm màu đó, và màu trung bình này sẽ là màu đại diện cho các màu thuộc vào nhóm màu này.

c Cách thức hoạt động của thuật toán K-Means trong đồ án

1. Chọn ra **K** màu trung tâm là đại diện cho các nhóm.
2. Phân các màu trong ảnh về từng nhóm bằng cách tìm khoảng cách giữa từng màu và màu trung tâm của từng nhóm. Khoảng cách nào nhỏ nhất thì màu đó sẽ phân về nhóm đó.
3. Thực hiện cập nhật màu trung tâm của nhóm bằng cách tính trung bình của các màu trong nhóm đó.
4. Quay lại thực hiện từ bước 2 cho tới khi thoả mãn 1 trong 2 điều kiện dừng.

Chúng ta sẽ có 2 điều kiện dừng thuật toán:

1. Thực hiện cho tới khi đủ số lượng lặp mà chúng ta qui định.
2. Điểm trung tâm của các nhóm sau khi được cập nhật không có thay đổi so với điểm trung tâm trước đó.

3 CÀI ĐẶT CHƯƠNG TRÌNH

Trong đồ án này, chương trình được thiết lập trên tập tin Jupyter Notebook (.ipynb)

3.1 Kĩ thuật

Để cài đặt chương trình cho dễ hiểu, dễ thay đổi và đi đúng hướng phân tích thì sẽ có những lưu ý như sau về kĩ thuật:

- Chương trình sẽ được chia thành các hàm nhỏ: Hàm hỗ trợ và hàm thực hiện thuật toán.
- Về bản chất, thông tin trong ảnh RGB là một ma trận 3 chiều các điểm ảnh, nhưng chúng ta sẽ đối xử và coi nó như là 1 ma trận 2 chiều các điểm ảnh (tức là 1 phần tử chính là 1 mảng chứa giá trị của 3 màu) để có thể dễ hiểu như những gì đã phân tích ở trên.

3.2 Mô tả các hàm

a Các hàm hỗ trợ

- Hàm đọc các điểm ảnh.
  - Input: Tên file ảnh.
  - Ouput: Ma trận **n** chiều các điểm ảnh của bức ảnh đó, với **n** là số kênh màu của điểm ảnh.

```
def read_Image(filename):
```

- Hàm in 2 ảnh để phục vụ cho việc so sánh (ảnh gốc và ảnh đã giảm màu).
  - Input: Ma trận điểm ảnh của ảnh 1, Ma trận điểm ảnh của ảnh 2, số lượng cụm K
  - Ouput: Hàm không trả về giá trị, nhưng in 2 ảnh ra màn hình kế nhau.

```
def show_Image_side_by_side(image1, image2, k_clusters):
```

- Hàm để thay đổi hình dạng của ma trận ảnh từ 2 chiều sang 1 chiều.
  - Input: Ma trận 2 chiều của ảnh.
  - Output: Ma trận 1 chiều của ảnh.

```
def reshape_2D_to_1D(image2d):
```

- Hàm tìm các mảng màu duy nhất trong ảnh. Hàm này có chức năng sẽ lọc những phần tử màu lặp lại của từng phần tử màu
  - Input: Ma trận 1 chiều của ảnh.
  - Output: Ma trận 1 chiều của các màu sau khi đã lọc những phần tử trùng.

```
def Unique_colors(image1d):
```

- Hai hàm khởi tạo các điểm trung tâm theo 2 cách: 'Random' và 'In_Pixel':
  1. Random: Ngẫu nhiên chọn **K** màu trung tâm.
     - Input: K cụm, số lượng kênh màu.

       ```
       def init_Random_Centroids(K_clusters, numchannels):
       ```
  2. In_Pixel: Ngẫu nhiên chọn **K** màu trung tâm mà các màu này có trong ảnh.
     - Input: K cụm, ma trận 1 chiều các màu sau khi lọc màu trùng.

       ```
       def init_Random_InPixel(K_clusters, UniqueColors):
       ```
  - Output: Cả 2 hàm đều trả về ma trận chứa các màu và được coi là các màu trung tâm của K nhóm.
- Hàm cấu trúc lại ảnh. Sau khi có được các màu trung tâm thích hợp nhất và mảng lưu thông tin màu nào sẽ thuộc nhóm nào (nhãn dán) thì sẽ tạo lại 1 hình mới.
  - Input: mảng các màu trung tâm, mảng các nhãn dán của từng điểm ảnh, ma trận 1 chiều của ảnh gốc, ma trận 2 chiều của ảnh gốc.

    ```
    def reconstruct_Image(centroids, labels, img_1d, img_2d):
    ```

b Các hàm thực hiện thuật toán

Trong phần này sẽ giải thích chi tiết về thuật toán và cách cài đặt các hàm

b.1 Hàm gán nhãn dãn cho từng điểm ảnh

- Input: ma trận 1 chiều của ảnh, mảng các điểm trung tâm
- Ouput: Trả về mảng các nhãn dán của các điểm ảnh trong hình.

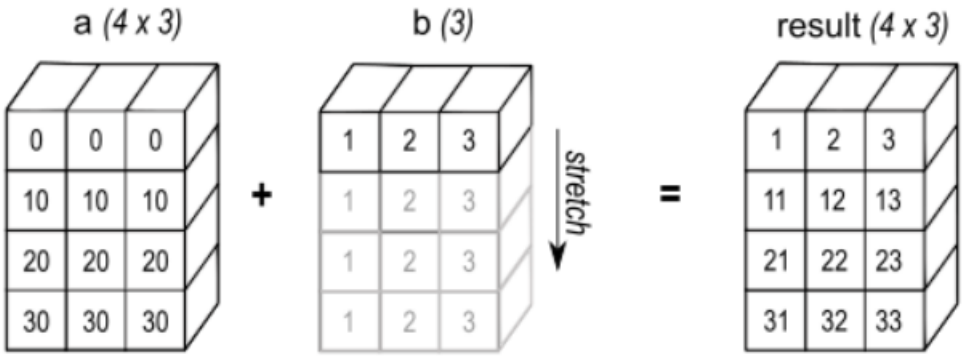
```
def get_Labels(img_1d, centroids):
```

- Ý tưởng cho hàm này chính là sẽ tính khoảng cách của mỗi điểm ảnh với tất cả các điểm trung tâm và chọn trung tâm nào có khoảng cách ngắn nhất thì điểm ảnh sẽ thuộc về nhóm của trung tâm đó.
- Tuy nhiên trong quá trình cài đặt, nếu áp dụng theo lý thuyết là sẽ sử dụng vòng lặp, chọn từng điểm ảnh và tính khoảng cách thì sẽ tốn rất nhiều thời gian, ảnh hưởng tốc độ của thuật toán **K-Means**. Chính vì vậy, thay vì sử dụng vòng lặp thông thường, ta sẽ tính toán khoảng cách ngay trên xử lý ma trận và phương pháp **Broadcasting** là công cụ mạnh mẽ.

- **Broadcasting** là cơ chế mạnh cho phép thực thi các phép toán số học trên các mảng numpy có kích thước khác nhau. Chúng ta có 1 ma trận nhỏ và 1 ma trận lớn hơn, và ta muốn sử dụng mảng nhỏ nhiều lần để thao tác trên mảng lớn.
- Có thể thấy mục đích **Broadcasting** và của thuật toán này rất giống nhau. Bản chất việc tính khoảng cách của 2 điểm **khoảng cách Euclid** trong không gian **n** chiều:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2}$$

  Tức là ta sẽ tính hiệu của từng điểm ảnh với các màu trung tâm, vậy ta coi ma trận lớn là ma trận các điểm ảnh, ma trận nhỏ là mảng các màu trung tâm, và phép toán sử dụng sẽ là phép trừ.



Hình 3: Ý tưởng tương tự như phép cộng trong hình

- Tuy nhiên lại xuất hiện 1 vấn đề. Chúng ta chỉ đang coi ma trận các điểm ảnh là 1 chiều, thực chất trong phần xử lý code thì là ma trận 2 chiều: **(a, 3)** với a là số lượng điểm ảnh, 3 là số kênh màu trong ảnh RGB. Trong khi đó ma trận chứa các màu trung tâm có kích thước là **(K, 3)** với K là số lượng nhóm và có thể $K > 1$. Tức là với số chiều của 2 ma trận như vậy thì sẽ không thể thực hiện được phép toán ma trận.
- Vấn đề này có thể được giải quyết nếu chúng ta coi lại **Hình 1**, nếu chung ta duỗi ma trận các màu trung tâm thành kích thước **(1, K, 3)** và ma trận điểm ảnh thành kích thước **(a, 1, 3)** thì sẽ thế nào? Đây chính là sức mạnh của **Broadcasting** khi cho phép thực hiện phép tính ma trận khi có 1 chiều là bằng 1.
- Vậy chúng ta giải quyết bằng cách thêm 1 chiều ảo vào hai ma trận, và để thêm chiều mới vào có nhiều cách nhưng trong thuật toán này sẽ sử dụng kĩ

	4x3				2x3	
0	0	10		1	2	3
10	10	10	-	4	5	6
20	20	20				
30	30	30				

Hình 4: Không thể thực hiện trừ 2 ma trận với kích thước như vậy

4x1x3		1x2x3			4x2x3	
[0, 10, 10]		[1, 2, 3]	[4, 5, 6]		[-1, 8, 7]	[-6, 5, 4]
[10, 10, 10]	-			=	[9, 8, 7]	[6, 5, 4]
[20, 20,20]					[19, 18, 17]	[16, 15, 14]
[30, 30, 30]					[29, 28, 27]	[26, 25, 24]

Hình 5: Có thể thực hiện phép trừ khi có 1 chiều bằng 1

thuật **Slicing**.

- Kĩ thuật **Slicing** được sử dụng truy xuất các phần tử từ **start:end** trong mảng, nhưng nếu chúng ta truy xuất từ hay tới **None** thì ma trận sẽ thêm 1 chiều ảo nữa. Ngoài ra ta có thể sử dụng **numpy.reshape** hay **numpy.newaxis**
- Việc còn lại là ta chỉ cần bình phương các phần tử, tính tổng và tìm ra xem khoảng cách nào nhỏ nhất thì sẽ gán nhãn vào màu đó. Chúng ta sẽ bỏ qua phần lấy căn bậc hai vì tổng số dương $s_1 > s_2$ thì khi lấy căn bậc hai chắc chắn kết quả so sánh không thay đổi.

- Khi thực hiện áp dụng vào cài đặt thì sẽ như sau:
  - Thực hiện trừ 2 ma trận và ép kiểu để tránh tràn số:

    ```
    diff = (img_1d[:,None] - centroids[None,:]).astype('int64')
    ```

  - Bình phương tất cả các phần tử:

    ```
    diff_square = diff**2
    ```

  - Tính tổng các phần tử trong từng mảng con, chính vì vậy ta phải cho **axis=-1** để tính tổng ở các ma trận nhỏ ở chiều cuối cùng:

    ```
    sum_all = np.sum(diff_square, axis=-1)
    ```

  - Tìm các nhóm nào có khoảng cách nhỏ nhất với các điểm ảnh:

    ```
    return np.argmin(sum_all)
    ```

b.2 Hàm cập nhật các màu trung tâm

- Input: ma trận 1 chiều của ảnh, ma trận nhãn dán, số kụm màu.
- Output: mảng các màu trung tâm mới.

  ```
  def update_Centroids(img_1d, labels, K_clusters):
  ```

- Ý tưởng của hàm này cũng khá dễ hiểu. Đầu tiên chúng ta sẽ thực hiện cộng dồn tất cả các màu thuộc trong cùng 1 nhãn dãn và đếm số lượng màu trong 1 nhóm đó.
- Việc còn lại chỉ là thực hiện tìm màu trung bình trong nhóm màu đó bằng cách lấy tổng số màu chia cho số lượng màu trong ảnh. Vậy mỗi màu trung tâm có thể được tính theo công thức:

$$c_j^{(\mu+1)} = \frac{1}{|C_j^{(\mu)}|} \Sigma_{x_i \in C_j^{(\mu)}} x_i, j = 1...p$$

- Với $c_j^{(\mu+1)}$ là màu trung tâm mới, $C_j^{(\mu)}$ là tập hợp các màu thuộc trung tâm đó và $|C_j^{(\mu)}|$ là số lượng màu trong nhóm.

b.3 Hàm cho phép thuật toán kết thúc

- Input: mảng các màu trung tâm cũ, mảng các màu trung tâm mới
- Ouput: True hoặc False

```
def stop_Condition(old_centroids, new_centroidsm, threshold=1e-5):
```

- Như đã được giới thiệu trong phần giải thích **Thuật toán K-Means**, ngoài việc sử dụng số lần vòng lặp tối đa để kết thúc thuật toán thì chúng ta có thể kiểm tra sự thay đổi của của mảng các màu trung tâm, nếu mảng màu trung tâm mới không có sự thay đổi thì sẽ mảng màu trung tâm đó đã được tìm thấy và có thể két thúc thuật toán sớm.
- Hàm sẽ thực hiện kiểm tra độ dài của 2 mảng màu trung tâm cũ và mới, và so sánh với 1 ngưỡng thấp nhất gần bằng 0 (**threshold=1e-5**). Nếu khoảng cách giữa 2 mảng màu này nhỏ hơn hoặc bằng ngưỡng này, chứng tỏ mảng màu trung tâm mới không thay đổi gì và kết thúc thuật toán:

$$\left|c^{(\mu+1)} - c^{(\mu)}\right| \leq 1e-5$$

- Với $c^{(\mu+1)}, c^{(\mu)}$ lần lượt là mảng màu trung tâm mới và cũ.

c Cài đặt thuật toán K-Means

- Sau khi đã chuẩn bị được các hàm hỗ trợ cần thiết cho thuật toán, ta sẽ thực hiện thuật toán K-Means như sau:
- Input: ma trận ảnh 1 chiều, số cụm màu, số vòng lặp tối đa, kiểu khởi tạo các màu trung tâm.
- Output: mảng các màu trung tâm, mảng nhãn dán của từng điểm ảnh.

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids):
    1.Khởi tạo các màu trung tâm theo 2 dạng: 'random' hoặc 'in_pixels'.
    2.Khởi tạo mảng màu duy nhất từ ma trận ảnh.
    3.Cho đến khi hết số lần lặp:
        4.Tạo mảng nhãn dán cho các điểm ảnh.
        5.Lưu mảng các màu trung tâm hiện tại.
        6.Cập nhật mảng màu trung tâm.
        7.So sánh sự thay đổi của mảng màu trung tâm.
            + Không thay đổi: Chuyển tới bước 8.
            + Thay đổi: Quay trở về bước 4.
    8. Trả về mảng các màu trung tâm, mảng nhãn dán.
```

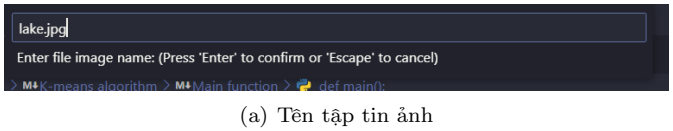
4 THỰC THI CHƯƠNG TRÌNH CHÍNH

4.1 Các thông tin đầu vào được yêu cầu

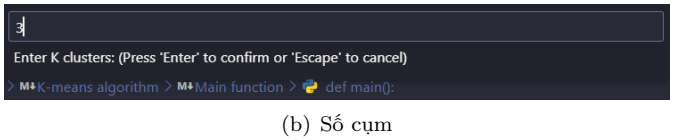
- Khi bắt đầu chạy chương trình, người dùng sẽ được yêu cầu nhập các thông tin đầu vào như sau:
  - **Tên tập in ảnh** (bao gồm đuôi file: png, jpg) và tập tin ảnh phải cùng chung thư mục với tập tin chương trình.
  - **Số lượng cụm màu**. Số lượng cụm màu càng lớn, chương trình thực thi có thể sẽ chậm hơn, tốt nhất số cụm màu từ [3-256] để đảm bảo chất lượng hình sau khi nén.
  - **Số lượng vòng lặp.**
  - **Loại khởi tạo các mảng màu trung tâm**: **'random'** hoặc **'in_pixels'**. Người dùng cần nhập đúng tên của loại khởi tạo thì chương trình mới được thực thi.
  - **Định dạng để lưu hình sau khi nén**: **pdf** và **png**. Người dùng cần nhập đúng tên định dạng để chương trình chạy đúng.
- Sau khi chạy và nhập đúng, đủ các thông tin đầu vào thì chương trình sẽ thực thi nén ảnh. Nén ảnh xong, tập tin ảnh mới sẽ được lưu với tên **Result.a** với a là định dạng mà người dùng muốn lưu khi nhập ban đầu.Đồng thời dưới chương trình cũng sẽ xuất ra màn hình 2 loại hình: ảnh gốc và ảnh sau khi nén để cho người dùng dễ so sánh sự thay đổi.

4.2 Hình ảnh khi chạy chương trinh

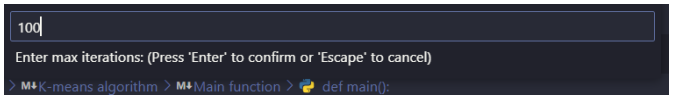
a Nhập thông số đầu vào



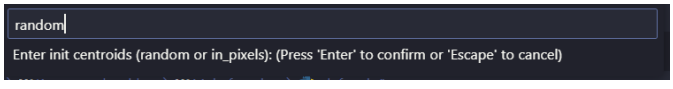
(a) Tên tập tin ảnh



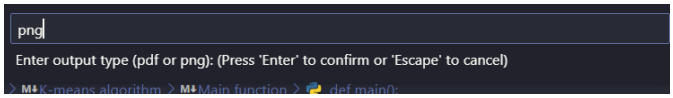
(b) Số cụm



(c) Số vòng lặp

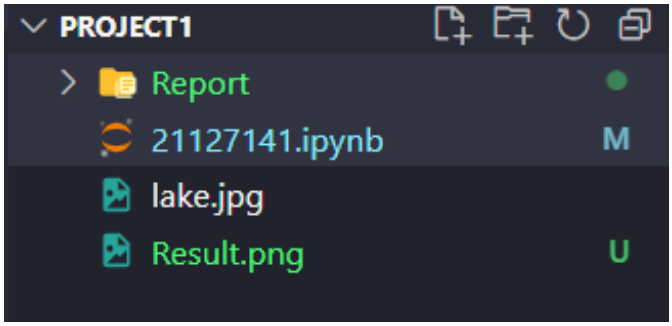


(d) Loại khởi tạo màu trung tâm

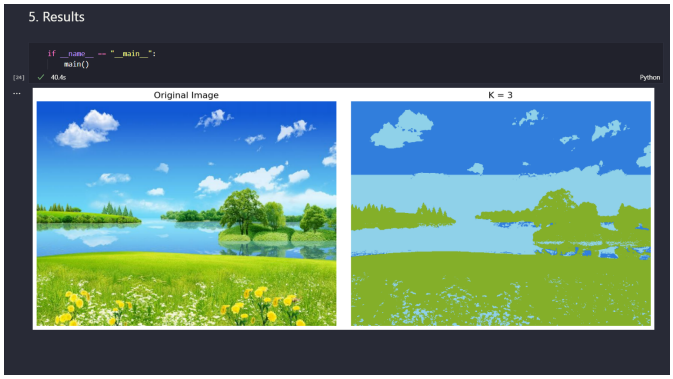


(e) Định dạng tập tin kết quả

b Kết quả của chương trình



(f) Vị trí của tập tin kết quả

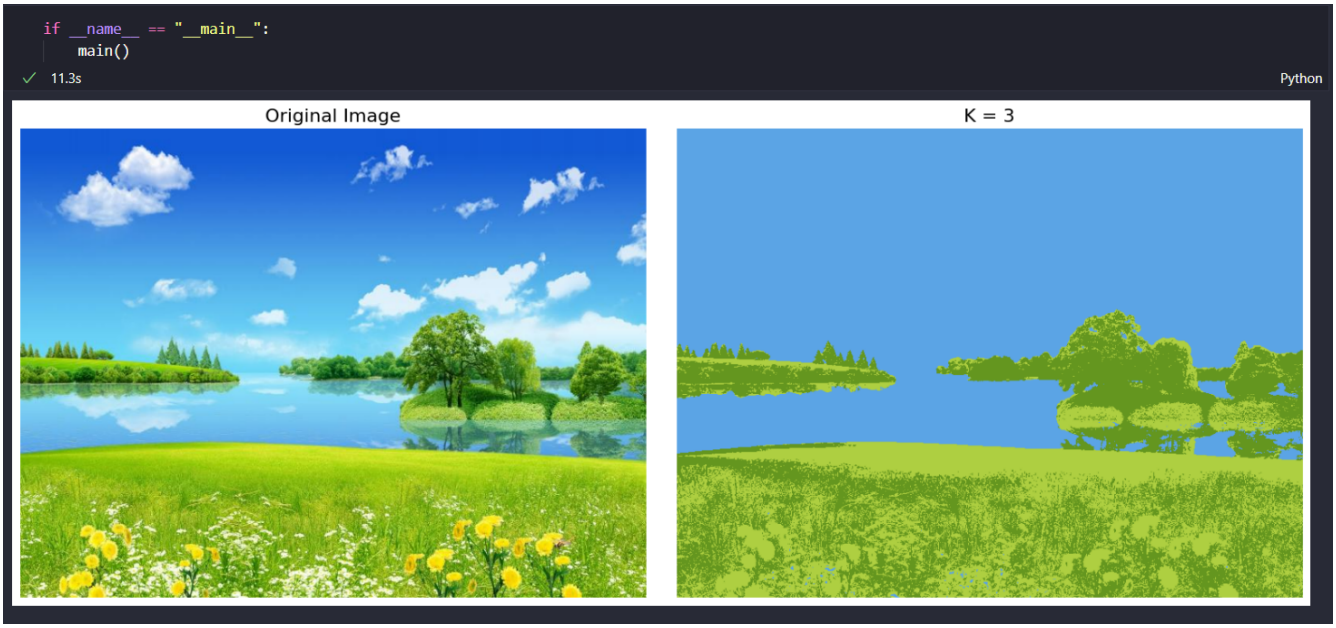


(g) Kết quả in ra màn hình

5 Nhận xét các trường hợp của thuật toán

- Trong phần này sẽ thực hiện chạy chương trình với số lượng màu nén xuống lần lượt là: 3, 5, 7.
- Trong 3 trường hợp màu này, số vòng lặp cao nhất sẽ được khai báo là 100.
- Hình được sử dụng trong lần kiểm thử này có kích thước 600x800.
- Cách khởi tạo các màu trung tâm sẽ là 'random'.
- Thông số của máy tính chạy chương trình: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 1.38 GHz 16.0 GB RAM

1. Đối với hình khi nén xuống còn 3 màu, chắc chắn hình ảnh sẽ khác rất nhiều so với ảnh gốc vì chỉ còn 3 màu chính: xanh lá đậm, xanh lá nhạt, xanh dương nhạt. Các hình ảnh cơ bản nhất như mây, hoa hay mặt nước là không thể nhận ra được. Các đường nét của các sự vật cũng nhìn rất đơn giản.



Hình 6: Kết quả K = 3

2. Đối với hình khi còn 5 màu thì số lượng màu đã được tăng thêm, giờ đây ta có thể nhận thấy màu trắng của mây và màu xanh dương đậm ở vùng trời trên cùng. Ở màu này cho chúng ta nhận biết một chút về bầu trời, tuy nhiên đường nét của sự vật vẫn chưa rõ nét.

Hình 7: Kết quả K = 5

3. Hình khi có 7 màu thì không thấy nhiều thay đổi lớn, có thêm 2 màu nữa làm rõ màu hoa vàng và phần bóng của cây ở mặt mước. Tuy đường nét vẫn cơ bản nhưng các màu cũ được tô thêm nhiều hơn, giúp người dùng hình dung được hình tốt hơn.

Hình 8: Kết quả K = 7

6 KẾT LUẬN

- Trong nhiều lần chạy các trường hợp khác về số lượng cụm giảm và số lần lặp tối đa thì có thể nhận xét như sau:
  - Số cụm nhỏ và số lần lặp nhỏ: Hình sẽ rất khó nhìn và gần như không thể nhận biết được so với hình gốc.
  - Số cụm lớn và số lần lặp nhỏ: Hình xử lý nhanh, có thể nhận biết được hình với ảnh gốc, nhưng có nhiều điểm ảnh lỗi.
  - Số cụm nhỏ và số lần lặp lớn: Hình xử lý nhanh, người dùng có thể hình dung sơ được hình nhưng vẫn rất sơ sài màu.
  - Số cụm nhiều và số lần lặp nhiều: Hình sát với hình gốc, ít thấy điểm ảnh sai, nhưng thời gian xử lý lâu.
- Có thể thấy, thuật toán K-Means thích hợp để có thể sử dụng trong việc nén ảnh xuống số màu nhất định. Tốc độ chạy của thuật toán này tối ưu nhất sẽ là việc lựa chọn số cụm cần giảm xuống và điều kiện dừng để người dùng đạt được hình có thể nhận biết tốt nhất có thể.

7 Tài liệu tham khảo

- Pseudo code K-Means
- Giải thích K-Means
- Kĩ thuật Broadcasting
- Slicing với phần tử None
- Công thức toán học và các phương pháp cập nhật điểm màu trung tâm
- Tài liệu các hàm trong thư viện numpy
- Thư viện matplotlib hỗ trợ xuất ảnh và lưu ảnh
- Thư viện Pillow hỗ trợ đọc ảnh