

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG HCM**



**BÁO CÁO**  
**PROJECT 1 – COLOR COMPRESSION**

**Sinh viên: Nguyễn Quốc Thắng**

**MSSV: 22127385**

**Lớp: 22CLC01**

## MỤC LỤC

1. Ý tưởng.....	3
2. Các hàm Đọc ảnh – Hiển thị ảnh – Lưu ảnh – Chuyển ảnh sang 1D.....	3
3. Thuật toán K-Means.....	4
4. Tạo ảnh mới từ tâm cụm và nhãn.....	6
5. Kết quả thử nghiệm.....	7
6. Tài liệu tham khảo.....	8

## 1. Ý tưởng:

*Quy trình Color Compression:*

1. Sử dụng hàm **read\_img** đọc ảnh từ đường dẫn
2. Sử dụng hàm **show\_img** hiển thị ảnh đã đọc.
3. Sử dụng hàm **convert\_img\_to\_1d** chuyển đổi ảnh sang dạng 1D.
4. Thực hiện thuật toán **K-means** trên ảnh 1D với số lượng cụm **k\_clusters** nhập vào và số lần lặp tối đa **max\_iter**.
5. Sử dụng hàm **generate\_2d\_img** tạo ảnh mới từ các tâm cụm và nhãn thu được sau khi áp dụng **K-means**.
6. Hiển thị ảnh mới sau khi đã nén màu sắc.
7. Thực hiện lưu ảnh vừa tạo (jpg hoặc pdf).

## 2. Các hàm Đọc ảnh – Hiển thị ảnh – Lưu ảnh – Chuyển ảnh sang 1D

### a. Đọc ảnh :

**Def read\_img( img\_path)**

- Đầu vào **img\_path** là đường dẫn tới ảnh.
- Sử dụng hàm **imread()** từ thư viện **matplotlib**.
- Chuyển đổi thành mảng 2D (bao gồm chiều cao, chiều rộng và kênh màu) và trả về mảng 2D.

### b. Hiển thị ảnh :

**Def show\_image (img\_2D)**

- Hàm này nhận một mảng 2D (**img\_2d**) làm đầu vào.
- Sử dụng hàm **imshow()** từ thư viện **matplotlib** để hiển thị ảnh
- Hàm **plt.show()** được gọi để thực sự hiển thị ảnh trên một cửa sổ mới. Điều này cho phép người dùng xem ảnh đã được xử lý hoặc đọc vào.

### c. Lưu ảnh :

**Def save\_as\_img (img\_2d, img\_path)**

- Sử dụng hàm **imsave** từ thư viện **matplotlib.image** (được giả định là đã được nhập với tên **mpimg**) để lưu mảng ảnh vào đường dẫn chỉ định.

**Def save\_as\_pdf (img\_2d, img\_path)**

- Sử dụng **plt.imshow** từ **matplotlib.pyplot** để hiển thị ảnh từ mảng 2D.
- **plt.savefig** với các tham số **bbox\_inches='tight'**, **pad\_inches=0** để lưu ảnh vào tệp PDF mà không có khoảng trắng xung quanh ảnh.

### d. Chuyển ảnh sang dạng 1D:

**Def convert\_img\_to\_1d(img\_2d)**

- Hàm nhận vào một mảng biểu diễn ảnh 2D.

- Sử dụng **np.array(img\_2d)** để đảm bảo img\_2d là một mảng numpy.
- Lấy kích thước của ảnh (**height, width, channels**) thông qua thuộc tính **shape** của mảng.
- Sử dụng **np.reshape** để chuyển đổi mảng 2D thành một mảng 1D. Kích thước mới của mảng là (**height \* width, channels**), nghĩa là mỗi hàng trong mảng mới sẽ tương ứng với một pixel trong ảnh gốc, và mỗi cột chứa giá trị của các kênh màu tại pixel đó.
- Trả về mảng 1D đã được chuyển đổi.

### 3. Thuật toán K-Means:

Thuật toán K-means là một thuật toán phân cụm (clustering) phổ biến, được sử dụng để phân chia dữ liệu thành các nhóm (cụm) dựa trên các đặc điểm của chúng. Thuật toán K-means được trình bày qua các bước sau:

**Def kmeans(img\_1d, k\_clusters, max\_iter, init\_centroids='random')**

#### Bước 1: Khởi tạo.

**Def kmeans\_init\_centers(X, k)**

Chọn **k\_clusters** điểm làm tâm cụm ban đầu. Có 2 phương pháp khởi tạo tâm cụm được mô tả qua tham số **init\_centroids**.

- **Random (Ngẫu nhiên):** Trong trường hợp này, các tâm cụm được khởi tạo một cách ngẫu nhiên trong không gian màu của dữ liệu X. Cụ thể, giá trị của mỗi tâm cụm được chọn ngẫu nhiên từ khoảng từ 0 đến 255 (giả sử dữ liệu là hình ảnh và giá trị màu nằm trong khoảng này) cho mỗi kênh màu. Phương pháp này đơn giản nhưng có thể không hiệu quả nếu dữ liệu có cấu trúc phức tạp.
- **In\_pixels (Trong các điểm dữ liệu):** Trong phương pháp này, k điểm dữ liệu được chọn ngẫu nhiên từ tập dữ liệu X để làm tâm cụm ban đầu. Điều này đảm bảo rằng các tâm cụm ban đầu luôn nằm trong phạm vi của dữ liệu thực tế, có thể giúp thuật toán hội tụ nhanh hơn so với việc chọn tâm cụm một cách hoàn toàn ngẫu nhiên.

#### Bước 2: Phân cụm.

**Def kmeans\_assign\_labels(X, centers)**

Với mỗi điểm trong dữ liệu (img\_1d), tính khoảng cách đến tất cả các tâm cụm và gán điểm đó vào cụm có tâm gần nó nhất.

- **X[:, None] - centers:** Trước hết, chúng ta mở rộng chiều của **X** bằng cách thêm một trục mới (None hoặc **np.newaxis** để tạo ra một trục mới) để có thể thực hiện phép trừ với **centers** theo cách phát sóng (broadcasting). Kết quả của phép trừ này là một mảng mới, trong đó mỗi điểm dữ liệu trong **X** được trừ đi mỗi tâm cụm trong **centers**, tạo ra một mảng các vector độ lệch.

- **np.linalg.norm(..., axis=-1):** Sau đó, chúng ta tính **norm (chuẩn)** của mỗi vector độ lệch này, dùng Euclidean.
- **np.argmax(..., axis=-1):** Cuối cùng, chúng ta tìm chỉ số của tâm cụm gần nhất với mỗi điểm dữ liệu bằng cách sử dụng np.argmax trên trục của các tâm cụm. np.argmax trả về chỉ số của giá trị nhỏ nhất trong mỗi hàng.

### **Bước 3: Cập nhật tâm cụm.**

**Def kmeans\_update\_centers(X, labels, k)**

Sau khi tất cả các điểm đã được gán vào cụm, cập nhật lại vị trí của tâm cụm bằng cách lấy trung bình cộng của tất cả các điểm trong cụm đó.

- **Khởi tạo:** Hàm nhận vào ba tham số: X, labels, và k. X là một mảng numpy chứa dữ liệu, labels là mảng chứa nhãn của cụm cho mỗi điểm dữ liệu, và k là số lượng cụm.
- **Khởi tạo mảng centers:** Một mảng zeros với kích thước (k, X.shape[1]) được tạo ra để lưu trữ tâm mới của các cụm. X.shape[1] đại diện cho số lượng thuộc tính của dữ liệu, do đó mỗi tâm cụm sẽ có cùng số lượng thuộc tính với dữ liệu.
- **Duyệt qua từng cụm:** Vòng lặp for chạy từ 0 đến k-1 để xử lý từng cụm một.
  - **Tìm điểm thuộc cụm:** Sử dụng **labels == i** để lọc ra những điểm dữ liệu thuộc về cụm thứ i. Kết quả là một mảng con của X chứa tất cả các điểm thuộc cụm i.
  - **Kiểm tra và cập nhật tâm cụm:**

Nếu cụm i có ít nhất một điểm và số lượng điểm trong cụm lớn hơn hoặc bằng số lượng thuộc tính, tâm của cụm i sẽ được cập nhật bằng cách tính trung bình cộng (**mean**) của tất cả các điểm trong cụm đó.
- **Trả về mảng centers:** Cuối cùng, hàm trả về mảng centers chứa tâm mới của tất cả các cụm.

### **Bước 4: Kiểm tra điều kiện dừng.**

**Def kmeans\_has\_converged(centers, new\_centers)**

Lặp lại bước 2 và 3 cho đến khi đạt đến số lần lặp tối đa (**max\_iter**) hoặc khi vị trí của các tâm cụm không thay đổi nữa, tức là thuật toán đã hội tụ.

- **centers** và **new\_centers** là hai danh sách chứa các trung tâm của các cụm từ lần lặp hiện tại và lần lặp trước đó.
- **[tuple(a) for a in centers]** chuyển danh sách các trung tâm từ dạng list của list sang list của tuple, vì tuple có thể được sử dụng làm key trong set (tập hợp), trong khi list thì không.

- **set([tuple(a) for a in centers])** tạo một set từ list của tuple, giúp loại bỏ các trung tâm trùng lặp và cho phép so sánh dễ dàng giữa hai tập hợp.
- Cuối cùng, **return (set([tuple(a) for a in centers]) == set([tuple(a) for a in new\_centers]))** trả về True nếu hai tập hợp trung tâm giống nhau (tức là không có sự thay đổi nào giữa hai lần lặp), và False nếu chúng khác nhau.

### **Bước 5: Kết quả.**

Kết hợp các bước. Trả về các tâm cụm cuối cùng và nhãn cụm cho mỗi điểm dữ liệu.

#### **# Main**

```
centers = kmeans_init_centers(img_1d, k_clusters)

labels = np.zeros(img_1d.shape[0])

iters = 0

while iters < max_iter:

    new_centers = centers

    labels = kmeans_assign_labels(img_1d, centers)

    centers = kmeans_update_centers(img_1d, labels, k_clusters)

    if kmeans_has_converged(new_centers, centers):

        break

    iters += 1

return centers, labels
```

### **4. Tạo ảnh mới từ tâm cụm và nhãn**

**Def generate\_2d\_img(img\_2d\_shape, centroids, labels)**

Hàm tạo ra một hình ảnh 2D từ các trung tâm cụm (centroids) và nhãn (labels) đã cho.

- **Khởi tạo một mảng ảnh trống:** Sử dụng np.zeros để tạo một mảng ảnh với kích thước img\_2d\_shape và kiểu dữ liệu np.uint8.
- **Chuyển đổi nhãn để phù hợp với kích thước ảnh:** Nhãn được cung cấp dưới dạng một mảng một chiều và cần được chuyển đổi (reshape) để phù hợp với kích thước 2D của ảnh.
- **Gán màu cho mỗi pixel dựa trên nhãn của nó:** Với mỗi pixel, tìm màu tương ứng của trung tâm cụm (centroid) dựa trên nhãn của pixel đó. Nhãn được sử dụng như một chỉ số để truy cập vào mảng centroids và lấy màu tương ứng.
- **Gán màu cho pixel:** Màu tìm được từ bước trên được gán cho pixel hiện tại trong ảnh.

- Trả về ảnh đã tạo.

## 5. Kết quả thử nghiệm

- a. Trường hợp 1 : nhiều màu sắc chạy, `init_centroids='random'` với `k_cluster = [3,5,7]`



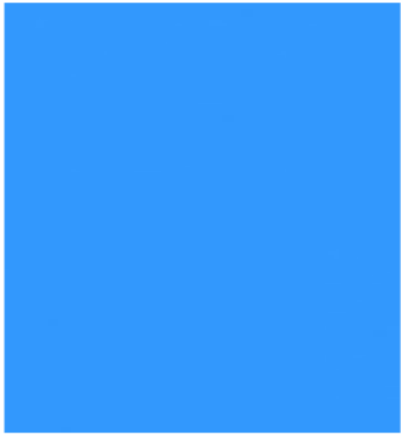
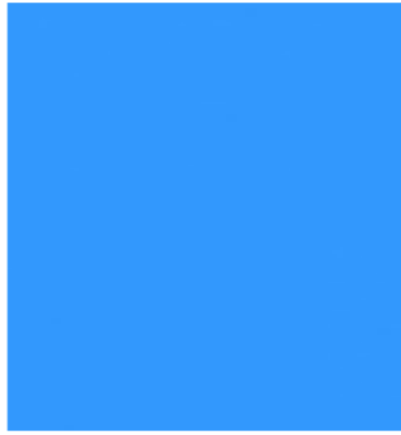
- Tổng thời gian chạy: 26.7 s

- b. Trường hợp 2 : nhiều màu sắc chạy, `init_centroids='in_pixels'` với `k_cluster = [3,5,7]`



- Tổng thời gian chạy: 34.4 s.

**c. Trường hợp 2 : một màu sắc chạy với k\_cluster = [3,5,7]**

Ảnh gốc	K = 3,5,7
	

- Tổng thời gian chạy: 0.2 s

**6. Tài liệu tham khảo:**

<https://machinelearningcoban.com/2017/01/01/kmeans/>

**-HẾT-**