

# Report Lab02 – Color Compression

---

AUGUST 2

---

Toán Ứng dụng và Thống Kê

Tác giả: Phạm Ngọc Thùy Trang – 18127022



---

# PROJECT 01 – LAB02

## THÔNG TIN CÁ NHÂN

Họ và tên: Phạm Ngọc Thùy Trang

MSSV: 18127022

Lớp: 18CLC1

## Ý TƯỞNG THỰC HIỆN VÀ MÔ TẢ CÁC HÀM

Như ta đã biết, phân mảnh hình ảnh là một quá trình phân đoạn một bức ảnh kỹ thuật số thành nhiều vùng riêng biệt mà mỗi pixel (một tập các pixel hay còn gọi là superpixels) của nó có cùng thuộc tính với nhau. Bằng cách này ta có thể xử lý chỉ với những thành phần quan trọng của bức ảnh thay vì phải xử lý toàn bộ bức ảnh đó. Mục đích của bài toán này chính là phân dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau. Trong đó cụm là tập hợp các điểm ở gần nhau trong một không gian nào đó. Từ đó ta sẽ xác định một con số cụ thể  $k$  bất kỳ cho clusters (con số này sẽ đề cập đến số lượng centroid ta cần trong tập dữ liệu).

Thuật toán này được sử dụng khi ta có dữ liệu chưa được đánh dấu (dữ liệu mà chưa được gom nhóm hay phân loại). Nó sẽ giúp ta tìm ra những nhóm dựa trên sự tương đồng của dữ liệu với số lượng các nhóm được biểu diễn bởi con số  $k\_cluster$ .

The diagram shows the objective function formula for K-means clustering:  $J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$ . Annotations include: 'number of clusters' pointing to  $k$ , 'number of cases' pointing to  $n$ , 'case  $i$ ' pointing to  $x_i^{(j)}$ , 'centroid for cluster  $j$ ' pointing to  $c_j$ , 'objective function' pointing to  $J$ , and 'Distance function' pointing to the norm expression  $\|x_i^{(j)} - c_j\|^2$ .

Pic Credit: [saedsayad.com](http://saedsayad.com)

Ý tưởng như sau:

1. Chọn số cluster  $k$ . Dữ liệu đầu vào là bức ảnh và số lượng cluster  $K$  cần tìm
2. Chọn  $K$  điểm bất kỳ làm các center (centroid) ban đầu
3. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất  $\rightarrow$  những điểm mà hình thành nên các  $K$  cluster
4. Nếu việc gán dữ liệu vào từng cluster ở bước trên không thay đổi gì so với vòng lặp trước nó thì ta dừng thuật toán. Dữ liệu đầu ra là bức ảnh sau khi đã được compress theo số lượng  $K$  cần tìm, các center  $M$  và label vector cho từng điểm dữ liệu  $Y$ , kích thước ảnh sau khi dùng thuật toán K-mean để compress (*không bắt buộc*)

5. Tính toán và cập nhật lại các center mới cho mỗi cluster bằng cách dùng trung bình cộng của tất cả các điểm dữ liệu đã được gán vào cluster đó ở bước 3
6. Quay lại bước 3

Theo như em tìm hiểu được thì bài toán này có liên quan đến hàm mất mát và bài toán tối ưu, cụ thể: Nếu ta coi center  $m_k$  là center (hoặc representative) của mỗi cluster và ước lượng tất cả các điểm được phân vào cluster này bởi  $m_k$ , thì một điểm dữ liệu  $x_i$  được phân vào cluster  $k$  sẽ bị sai số là  $(x_i - m_k)$ . Chúng ta mong muốn sai số này có trị tuyệt đối nhỏ nhất

### Mô tả các hàm:

Vì trong code đã được chú thích rất kĩ nên ta sẽ đi vào những hàm/công việc chính:

Tên hàm/ Tên công việc	Mô tả	Một phần code
	Import các thư viện cần thiết	<pre>#Import các thư viện cần thiết import matplotlib import numpy as np import numpy.matlib import random import PIL import os import warnings warnings.filterwarnings("ignore")</pre>
	Đọc và hiển thị bức ảnh đầu vào (dùng thư viện)	<pre>image = PIL.Image.open(path_image) matplotlib.pyplot.imshow(image) image = np.asarray(image)</pre>
	<p>Lấy các thông tin của bức ảnh như lấy thông số dòng và thông số cột cũng như chuyển mỗi giá trị pixel về dạng <math>[0,1]</math>. Điều này sẽ giúp thuật toán k-means xử lý hiệu quả hơn.</p> <p>Ta có thể dùng các method từ thư viện để có được những thông tin này và dùng lệnh print để kiểm tra</p>	<pre>image_height = image.shape[0] image_width = image.shape[1] image = image/255 img_id = image.reshape(image.shape[0]*image.shape[1],1) print(img_id.shape) print(img_id)</pre>

<pre>def closet_centroids(X,c)</pre>	<p>Hàm này được dùng để tìm các điểm giá trị pixel gần với mỗi giá trị centroids</p> <p>Để hiểu được công thức tìm các điểm giá trị pixel này, ta cần đọc trước và hiểu công thức hàm mất mát và bài toán tối ưu</p>	<pre>K = np.size(c,0) idx = np.zeros((np.size(X,0),1)) array = np.empty((np.size(X,0),1)) for i in range(0,K):     y = c[i]     temp = np.ones((np.size(X,0),1))*y     b = np.power(np.subtract(X,temp),2)     a = np.sum(b,axis = 1)     a = np.asarray(a)     a.resize((np.size(X,0),1))     array = np.append(array, a, axis=1) array = np.delete(array,0,axis=1) idx = np.argmin(array, axis=1) return idx</pre>
<pre>def compute_centroids(img_1d, idx, k_clusters)</pre>	<p>Bằng cách truyền vào các tham số như thông số vòng lặp chạy, số k cluster và ảnh X sau khi tiền xử lý để tính toán các giá trị các centroids (các center)</p>	<pre>n = np.size(img_1d,1) centroids = np.zeros((k_clusters,n)) for i in range(0,k_clusters):     ci = idx == i     ci = ci.astype(int)     total_number = sum(ci)     ci.resize((np.size(img_1d,0),1))     total_matrix = np.matlib.repmat(ci,1,n)     ci = np.transpose(ci)     total = np.multiply(img_1d,total_matrix)     centroids[i] = (1/total_number)*np.sum(total,axis=0) return centroids</pre>
<pre>def kmeans(img_1d, k_clusters, max_iter, init_centroids = 'random')</pre>	<p>Bằng cách truyền vào các tham số như thông số vòng lặp chạy, số k cluster, tổng số lần chạy của thuật toán kmeans trước khi kết thúc là max-iter, các giá trị center ban đầu được khởi tạo, ảnh X sau khi tiền xử lý để trả ra các centroid và các nhãn.</p>	<pre>initial_centroids = init_centroids_func(img_1d,k_clusters) m = np.size(img_1d,0) n = np.size(img_1d,1) K = np.size(initial_centroids,0) centroids = initial_centroids previous_centroids = centroids labels = np.zeros((m,1)) for i in range(1,max_iter):     labels = closest_centroids(img_1d,centroids)     centroids = compute_centroids(img_1d,labels ,k_clusters) return centroids, labels</pre>
	<p>Đoạn code này sẽ tìm các điểm gần với điểm center và đặt giá trị của center là các giá trị gần trong 1 nhóm đó.</p>	<pre>image_compressed = centroids[labels] print(image_compressed) image_compressed = np.reshape(image_compressed, (image_height, image_width, 3)) print("Kích thước ảnh sau khi reshape: ", np.shape(image_compressed))</pre>
	<p>Đoạn code này sẽ bắt đầu chạy thuật toán K-means trên bức ảnh đầu vào và hiển thị kích thước ví dụ số lượng center, ví dụ ta chọn k = 7 nên sẽ là một vector 7 chiều, và 1 giá trị là một mảng chứa giá trị 3 gam màu RGB</p>	<pre>centroids,labels = kmeans(img_1d,k_clusters,max_iter) # Hiện thị kích thước ví dụ số lượng center : ví minh c print(np.shape(centroids)) print(np.shape(labels)) print("Hiện thị label: ", labels) print("Hiện thị giá trị centroids :", centroids)</pre>



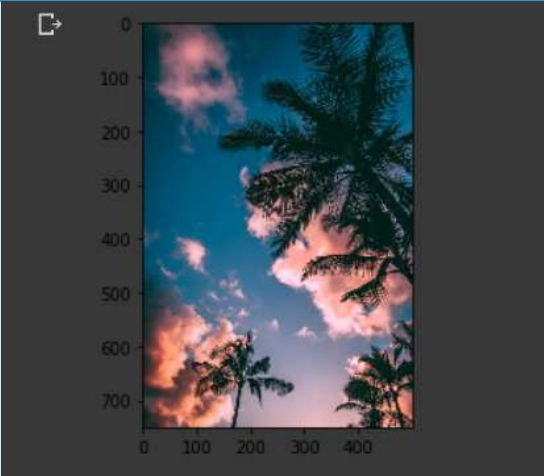
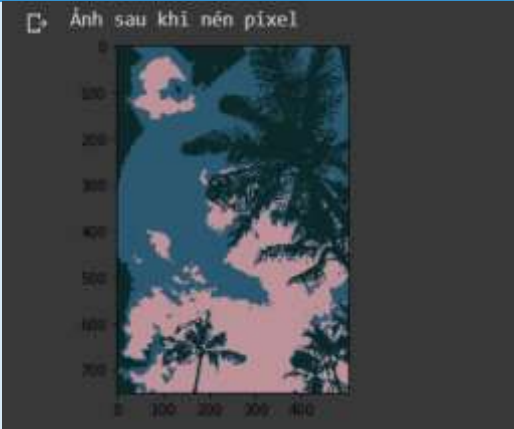
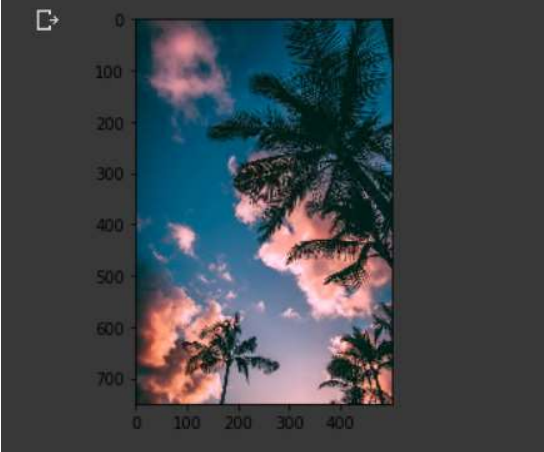
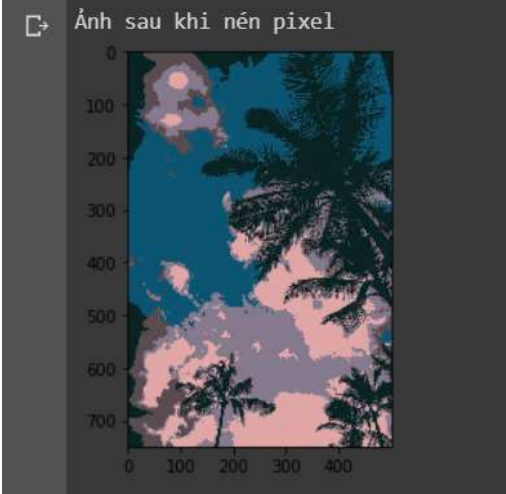
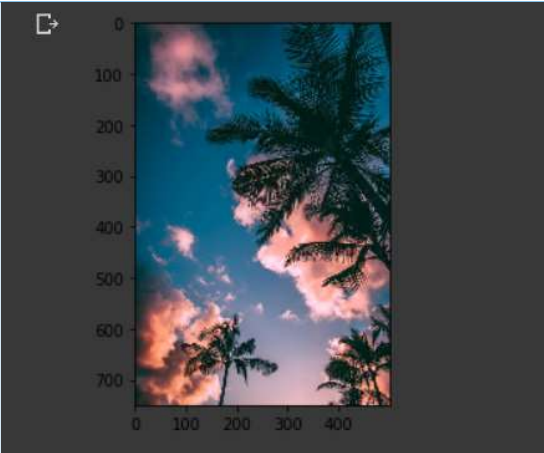

<code>def init_centroids_func(img_1d, k_clusters):</code>	Hàm này được dùng để khởi tạo ngẫu nhiên giá trị centroid với tham số đầu vào là ảnh X và số lớp clusters K	<pre>def init_centroids_func(img_1d, k_clusters):     c = random.sample(list(img_1d), k_clusters)     return c</pre>
	Đoạn code này được dùng lưu ảnh xuống thành file image sau khi nén	<pre># Lưu ảnh xuống thành file image sau khi nén from PIL import Image image_compressed = Image.fromarray((image_compressed * 255).astype(numpy.uint8)) image_compressed.save('hg_compression.jpg')</pre>
	Đoạn code này được dùng để hiển thị kết quả kích thước ảnh đầu vào và ảnh sau khi nén.	<pre>import matplotlib print("Ảnh sau khi nén pixel") matplotlib.pyplot.imshow(image_compressed)  # Hiển thị kết quả kích thước ảnh đầu vào và ảnh sau khi nén info = os.stat(path_image) print("Kích thước ảnh gốc ban đầu: ", info.st_size/1024, "KB") info = os.stat('hg_compression.jpg') print("Kích thước ảnh sau khi sử dụng K-mean: ", info.st_size/1024, "KB") print("Hiển thị labels: ", labels) print("Hiển thị giá trị centroids: ", centroids)</pre>

## HÌNH ẢNH KẾT QUẢ VỚI TỪNG SỐ LƯỢNG MÀU, $K = \{3, 5, 7\}$

Với đầu vào là một tấm ảnh bên dưới



Tương ứng với từng chỉ số  $k\_clusters$  ta có kết quả tương ứng như sau:

$k\_clusters$	Hình ảnh đầu vào	Hình ảnh kết quả
3		
5		
7		

---

## NHẬN XÉT KẾT QUẢ TRÊN

Mục đích của bài toán này chính là phân dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau. Trong đó cụm là tập hợp các điểm ở gần nhau trong một không gian nào đó. Từ đó ta sẽ xác định một con số cụ thể  $k$  bất kì cho clusters (con số này sẽ đề cập đến số lượng centroid ta cần trong tập dữ liệu và mỗi một điểm ảnh sau khi xử lý sẽ được biểu diễn bởi 1 số tương ứng với 1 cluster). Hay nói cách khác là ta phân mảnh dữ liệu đầu vào thành  $K$ -cluster hoặc những phân khác nhau dựa trên các  $K$ -centroids

Đó là lý do nếu ta có số  $k\_clusters$  càng nhiều thì ta thấy màu sắc càng gần với màu sắc của bức ảnh ban đầu hơn do thuật toán Kmeans đã phân loại nhiều cluster màu hơn, còn nếu càng ít thì màu sắc của nó càng xa với màu sắc thực tế hơn. Và sau khi compress thì ta thấy *chất lượng ảnh đã giảm đi nhiều*.

Chẳng hạn:

- **Với  $k = 3$**  thì thuật toán đã phân loại bức ảnh gốc của chúng ta thành **3 miền màu khác nhau**. Bức tranh lúc này do chỉ có 3 màu nên khá mờ và một số vật thể vẫn chưa rõ hình thù như bức tranh gốc, đặc biệt là vùng trời, ta có thể thấy hình thù của mây và cây, nhưng với mây thì nó chỉ có một sắc thái duy nhất.
- **Với  $k = 5$** , thuật toán đã phân loại bức ảnh gốc của chúng ta thành **5 miền màu khác nhau**. Bức tranh lúc này đã rõ hơn lúc trước và ta cũng có thể thấy thêm nhiều sắc thái khác nhau của bầu trời và mây.
- **Với  $k = 7$** , thuật toán đã phân loại bức ảnh gốc của chúng ta thành **7 miền màu khác nhau**. Bức tranh lúc này đã rõ hơn lúc  $k = 5$  và  $k = 3$  khá nhiều, sắc thái của mây, trời cũng thay đổi đáng kể.



---

## REFERENCES

[1]: <https://towardsdatascience.com/introduction-to-image-segmentation-with-k-means-clustering-83fd0a9e2fc3>

[2]: <https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1>