

**University of Science**  
**Viet Nam National University - Ho Chi Minh city**

---



**TOÁN ỨNG DỤNG VÀ THỐNG KÊ  
CHO CÔNG NGHỆ THÔNG TIN**

**Đồ án 1: Color Compression**

**Lớp: 21CLC02  
Sinh viên thực hiện:  
Lê Hoàng Sang - 21127158**

# Mục lục

<b>1 Lời nói đầu</b>	<b>2</b>
<b>2 Lịch sử hình thành</b>	<b>2</b>
<b>3 Tham khảo</b>	<b>2</b>
<b>4 Mô tả mã nguồn, nhập xuất ảnh</b>	<b>3</b>
<b>5 Mô tả thuật toán</b>	<b>4</b>
5.1 Ý tưởng của thuật toán . . . . .	4
5.2 Các bước chi tiết . . . . .	4
5.2.1 Hàm kmeans() . . . . .	4
5.2.2 Hàm compress_handling() . . . . .	7
<b>6 Phân tích thời gian xử lí</b>	<b>8</b>
6.1 Thông kê thời gian . . . . .	8
6.2 Nhận xét . . . . .	9
<b>7 Hình minh họa</b>	<b>10</b>

## 1 Lời nói đầu

- Báo cáo này trình bày về việc nén ảnh bằng thuật toán K-means. Thuật toán K-means là một phương pháp phân cụm dữ liệu được sử dụng rộng rãi trong xử lý ảnh và có thể được áp dụng để giảm số lượng màu sắc trong ảnh.
- Việc nén ảnh giúp giảm kích thước tệp tin và tăng tốc độ tải ảnh, đồng thời tạo ra một phiên bản nén của ảnh với kích thước nhỏ hơn. Bài báo cáo này sẽ tìm hiểu về cách thuật toán K-means hoạt động và cách áp dụng nó để nén ảnh một cách hiệu quả.
- Thuật toán được triển khai bằng ngôn ngữ lập trình Python trong file Jupiter Notebook. Người dùng chỉ cần bấm Run All và nhập vào tên ảnh, định dạng ảnh muốn xuất ra.

## 2 Lịch sử hình thành

- Thuật toán K-means được giới thiệu bởi Stuart Lloyd vào năm 1957 và sau đó tái phát triển bởi James MacQueen và Edward W. Forgy vào những năm 1965 và 1967.
- Lịch sử phát triển của K-means đã chứng kiến sự đóng góp của nhiều nhà khoa học và nghiên cứu trong suốt nhiều thập kỷ. Các biến thể và cải tiến của thuật toán đã được đề xuất để cải thiện hiệu suất và khả năng áp dụng. Các ví dụ điển hình bao gồm thuật toán K-means++ giới thiệu bởi David Arthur và Sergei Vassilvitskii vào năm 2007, và thuật toán Mini-Batch K-means của David Sculley và cộng sự vào năm 2010.
- Hiện nay, K-means được xem là một trong những thuật toán phân cụm phổ biến và được sử dụng rộng rãi trong cả lĩnh vực học máy và xử lý ảnh. Đóng góp của K-means đã định hình và mở ra nhiều lĩnh vực nghiên cứu và ứng dụng liên quan đến phân cụm và khám phá dữ liệu.

## 3 Tham khảo

- Kiến thức môn Toán Ứng dụng và Thống kê - HCMUS
- Introduction to k-Means Clustering with scikit-learn in Python
- numpy.linalg.norm

- numpy.random.randint
- NumPy np.newaxis()
- NumPy argmin(): Get Index of the Min Value in Arrays
- Save Image with Pillow in Python
- numpy.allclose

## 4 Mô tả mã nguồn, nhập xuất ảnh

- Hàm read\_image():
  - Dùng để mở file, trả về các thông số của ảnh từ tên ảnh người dùng nhập vào.
  - Ví dụ, mở ảnh HCMUS.jpg, người dùng nhập 'HCMUS.jpg' vào input (Lưu ý nhập cả đuôi ảnh).
- Hàm display\_image(image):
  - Dùng hiện thị ảnh bằng thư viện matplotlib.
- Hàm ask\_k():
  - Yêu cầu người dùng nhập số cụm màu, yêu cầu số nhập vào lớn hơn 0.
- Hàm save\_image(rgb\_array, image\_path):
  - Dùng để xuất file ra thư mục cùng cấp với tên file dạng: <tên>\_<số clusters>\_clusters\_compressed.<đuôi ảnh>.
  - Hàm sẽ gọi đến hàm ask\_image\_format() để yêu cầu người dùng nhập vào input định dạng ảnh (pdf/png), hoặc báo lỗi nếu nhập sai định dạng ảnh.
- Hàm get\_random\_in\_pixels(image):
  - Trong hàm này, chúng ta sử dụng hàm np.random.choice để lấy ngẫu nhiên k chỉ mục từ ảnh (trong giới hạn độ dài ảnh), sao cho các vị trí lấy được không được giống nhau (trường hợp các phần tử mảng centers trong thuật toán được sinh ra giống nhau) thông qua tham số replace=False.
- Cơ chế đếm thời gian:
  - Dùng hàm đếm thời gian của thư viện Time để tính khoảng thời gian từ khi bắt đầu và kết thúc thuật toán, không kể thời gian nhập xuất ảnh.
  - In thời gian đếm được ra Terminal.

## 5 Mô tả thuật toán

### 5.1 Ý tưởng của thuật toán

- Ý tưởng chính của thuật toán K-means là phân cụm dữ liệu thành các cụm (clusters) dựa trên sự tương đồng của các điểm dữ liệu (tương ứng với các màu trong đồ án). Điểm mấu chốt là tìm cách xác định các tâm cụm sao cho tổng bình phương khoảng cách giữa các điểm dữ liệu và tâm cụm tương ứng là nhỏ nhất.

### 5.2 Các bước chi tiết

#### 5.2.1 Hàm kmeans()

- Bước đầu tiên là người dùng số lượng cụm K mà ta muốn tạo (3, 5 hoặc 7). Và xét xem thuật toán đang chạy dưới trường hợp init\_centroids random hay in\_pixels.
- Trường hợp init\_centroids là random: chọn ngẫu nhiên K điểm từ dữ liệu ban đầu người dùng chọn làm tâm cụm ban đầu, bằng hàm randint của thư viện NumPy.
  - `randint(0, 256, size=(k, 3))`: dùng để tạo một ma trận có kích thước  $(k, 3)$  với các giá trị ngẫu nhiên nằm trong khoảng từ 0 đến 255 (chỉ số màu RGB).
  - `size=(k, 3)`: Đây là đối số (có thể có hoặc không) của hàm `np.random.randint()` để xác định kích thước của ma trận ngẫu nhiên. Kích thước của ma trận là  $(k, 3)$ , có nghĩa là ta có  $k$  hàng (số lượng cụm mà chúng ta muốn tạo trong thuật toán K-means) và 3 cột. Mỗi hàng của ma trận sẽ chứa 3 giá trị ngẫu nhiên, tương ứng với giá trị màu của các kênh Red, Green và Blue (RGB).
- Trường hợp init\_centroids là in\_pixels: trường hợp này được xử lý bằng cách tạo ra mảng random số tự nhiên trong khoảng `len(image)` (kích thước ảnh). Trường hợp này sẽ gọi đến hàm `get_random_in_pixels()` để lấy ra các màu ngẫu nhiên không trùng nhau (tham số `replace = False`).
- Nếu ở hàm Main gọi đến hàm `k_means()` nhưng truyền sai tham số `init_centroids` thì chương trình sẽ báo lỗi.
- Tiếp theo, với mỗi điểm màu ảnh, tính khoảng cách Euclidean từ điểm đó đến tất cả các tâm cụm.
  - Hàm `image[:, np.newaxis]` có tác dụng thêm một chiều mới vào ma trận `image`, biến nó từ một ma trận 2D có hình dạng (hàng, cột) thành một ma trận 3D có

hình dạng (hàng, cột, lớp). Khi đó, ta có thể thực hiện phép trừ giữa ma trận image và centroids.

- Phép trừ image[:, np.newaxis] - centroids: được thực hiện giữa ma trận image và ma trận centroids. Với mỗi điểm ảnh trong image, nó sẽ được trừ đi từng tâm cụm tương ứng trong centroids.
- Có rất nhiều cách để tính khoảng cách Euclidean ví dụ như hàm linalg.norm(), tuy nhiên, thay vì tính toán khoảng cách cho tất cả các cặp điểm bằng hàm norm(), ta sử dụng phép tính khoảng cách Euclidean cho từng cặp điểm một bằng cách sử dụng toán tử '-' và hàm np.sum() để tính tổng bình phương.
- Vì sao tổng  $** 2$  mà không lấy căn: Phép tính này cộng các bình phương từng phần tử trong mảng đã được tính toán từ phép trừ. Ta không lấy căn 2 lại sau khi tính tổng bình phương vì ta xét trên bình phương vẫn đúng bản chất min/max, nhưng đổi lại việc lấy căn sqrt() sẽ làm thuật toán chạy chậm đáng kể.
- Mở rộng số chiều của image bằng image[:, np.newaxis], kết quả là một ma trận có thêm một chiều. Ví dụ về mở rộng số chiều cho ma trận:

```

> import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
new_arr = arr[np.newaxis]
print(new_arr)
[90]   ✓ 0.0s
...
[[1 2 3]
 [4 5 6]]
[[[1 2 3]
 [4 5 6]]]

```

Hình 1: Ảnh minh họa về mở rộng số chiều cho ma trận

- Thực hiện phép trừ giữa ma trận mở rộng image[:, np.newaxis] và ma trận centroids, kết quả là một ma trận đại diện cho khoảng cách giữa từng điểm ảnh trong image và từng tâm cụm trong centroids.
- Vì sao axis=2, do ta tính toán theo chiều thứ 3 của ma trận kết quả. Vì axis=2, các giá trị sẽ được tính cho từng cặp điểm ảnh và tâm cụm trên chiều thứ 3. Kết quả sẽ là một ma trận có kích thước (số điểm ảnh, số tâm cụm), trong đó mỗi

phần tử đại diện cho khoảng cách giữa điểm ảnh và tâm cụm tương ứng. Minh họa ma trận distances:

```
[ [ 866 69354 39642 ... 5389 76609 6953]
  [ 866 69354 39642 ... 5389 76609 6953]
  [ 866 69354 39642 ... 5389 76609 6953]
  ...
  [55526   106  5654 ... 38017    149 33563]
  [55526   106  5654 ... 38017    149 33563]
  [56345   113  5897 ... 38694    110 34200] ]
```

Hình 2: Ảnh minh họa mảng distances khi chạy thuật toán

- Gán điểm dữ liệu vào cụm có tâm cụm gần nhất dựa trên khoảng cách nhỏ nhất.
  - Hàm argmin(): dùng để tìm chỉ số của giá trị nhỏ nhất trong một mảng hoặc trên một trực cụ thể. Trong trường hợp này, ta sử dụng argmin(distances, axis=1) để tìm chỉ số của giá trị nhỏ nhất trên mỗi cột của distances.
  - Vì sao axis=1: đối số này xác định trục theo đó ta muốn tìm chỉ số của giá trị nhỏ nhất. Kết quả sẽ là một mảng 1 chiều có kích thước (n), trong đó mỗi phần tử đại diện cho chỉ số của tâm cụm gần nhất cho từng điểm dữ liệu.
  - Ví dụ minh họa rõ hơn về hai trường hợp axis = 0 và axis = 1: với tham số axis = 0, hàm argmin() sẽ trả về vị trí phần tử nhỏ nhất theo dòng. Tương tự, axis = 1 hàm sẽ trả về vị trí min theo cột.
  - Như vậy, ta lấy chỉ số của giá trị nhỏ nhất trên mỗi cột của ma trận distances để xác định tâm cụm gần nhất cho từng điểm dữ liệu. Kết quả là một mảng labels có kích thước (n), trong đó mỗi phần tử đại diện cho tâm cụm tương ứng được gán cho từng điểm dữ liệu.
- Cập nhật tâm cụm:
  - Vòng lặp for duyệt qua tất cả các tâm cụm.
  - cluster\_pixels = image[labels == i]: tạo ra một mảng cluster\_pixels chứa các điểm ảnh thuộc về tâm cụm thứ i. Nó được thực hiện bằng cách sử dụng mảng labels để chọn các chỉ mục của các điểm ảnh có giá trị tương ứng với tâm cụm i. Labels == i sẽ trả về giá trị Boolean, cho biết vị trí của các điểm ảnh thuộc về tâm cụm thứ i.

```

import numpy as np

arr = np.array([[10, 7, 15], [11, 12, 6]])
print(arr)

min0_idx = np.argmin(arr, axis=0)
print(min0_idx)

min1_idx = np.argmin(arr, axis=1)
print(min1_idx)

```

[122] ✓ 0.0s

... [[10 7 15]  
[11 12 6]]  
[0 0 1]  
[1 2]

Hình 3: Ảnh minh họa cho hàm argmin()

- Tiếp theo, kiểm tra xem cluster\_pixels có phần tử nào không. Nếu có, tiếp tục thực hiện cập nhật tâm cụm. Tính trung bình các giá trị của cluster\_pixels theo trục 0, tức là tính trung bình theo từng kênh màu. Kết quả là một vector có cùng kích thước với số kênh màu. Nó được gán cho tâm cụm thứ i trong ma trận centroids, để cập nhật tâm cụm đó.
- Lặp lại bước Gán điểm dữ liệu vào cụm có tâm cụm gần nhất và bước Cập nhật tâm cụm.
- Trong lúc lặp, nếu có trường hợp centroids gần như không đổi (xấp xỉ) sau khi cập nhật lại centroids (old\_centroids = centroids) thì ngưng lặp và trả về. Để kiểm tra xấp xỉ, ta dùng hàm allclose() với sai số 1 đơn vị, cách này sẽ làm cho tốc độ xử lí của thuật toán cải thiện đáng kể thay vì lây bằng chính xác.
- Cuối cùng, hàm trả về centroids (vị trí của các tâm cụm sau khi hoàn thành thuật toán) và labels (nhãn của các điểm ảnh tương ứng với các tâm cụm gần nhất).

### 5.2.2 Hàm compress\_handling()

- Hàm sẽ nhận các tham số tâm cụm (centroids), nhãn của các điểm ảnh (labels), kích thước của ảnh gốc (width và height) và trả về ảnh đã được nén trong hình dạng mới.

- Đầu tiên, ta tạo ra một mảng compressed\_pixels bằng cách chọn các tâm cụm tương ứng với nhãn của từng điểm ảnh trong mảng labels. Mảng compressed\_pixels chứa các giá trị màu của các tâm cụm được gán cho từng điểm ảnh.
- Tiếp theo, ta thay đổi hình dạng của mảng compressed\_pixels thành hình dạng (width, height, 3). Điều này tạo ra một ảnh với chiều rộng là width, chiều cao là height và 3 lớp kênh màu (R, G, B).

## 6 Phân tích thời gian xử lý

### 6.1 Thống kê thời gian

- Thời gian xử lí của thuật toán được chạy thử trên nhiều ảnh với các kích thước khác nhau, và không kể thời gian chương trình chờ người dùng nhập các thông tin như tên ảnh, định dạng.
- Việc chạy thuật toán và thống kê thời gian được thực hiện trên laptop Macbook Pro M1.

Bảng 1: Bảng thống kê thời gian chạy của ảnh JPG 70 KB kích thước  $800 \times 454$

STT	Số cụm màu	Loại init_pixels	Thời gian (giây)	Ghi chú
1	3	Random	0,04s	
2	3	In_pixels	0,04s	
3	5	Random	0,08s	
4	5	In_pixels	0,06s	
5	7	Random	0,1s	
6	7	In_pixels	0,08s	

Bảng 2: Bảng thống kê thời gian chạy của ảnh JPG 442 KB kích thước  $1920 \times 1228$  được tải từ Unsplash.com

STT	Số cụm màu	Loại init_pixels	Thời gian (giây)	Ghi chú
1	3	Random	0,5s	
2	3	In_pixels	0,5s	
3	5	Random	0,7s	
4	5	In_pixels	0,6s	
5	7	Random	0,96s	
6	7	In_pixels	0,5s	

Bảng 3: Bảng thống kê thời gian chạy của ảnh JPG 4,9 MB kích thước  $6000 \times 3111$  được tải từ Unsplash.com

STT	Số cụm màu	Loại init_pixels	Thời gian (giây)	Ghi chú
1	3	Random	4,1s	
2	3	In_pixels	2,5s	
3	5	Random	5,0s	
4	5	In_pixels	4,0s	
5	7	Random	9,4s	
6	7	In_pixels	5,2s	

## 6.2 Nhận xét

- Nhận xét 1: Các trường hợp khởi tạo tâm cụm in\_pixels cho thời gian chạy ổn định hơn, vì các giá điểm ảnh không bị khởi tạo bên ngoài các giá trị màu của bức ảnh.
- Nhận xét 2: Hình ảnh càng lớn, tốc độ chạy có phần chậm hơn rõ rệt. Tuy nhiên, do ở phần thuật toán ta lấy xấp xỉ 1 đơn vị màu, nên vẫn đảm bảo tốc độ cho các ảnh lớn.
- Nhận xét 3: So sánh trên bình phương khoảng cách và so sánh xấp xỉ để tạm dừng thuật toán là 2 điểm mấu chốt cải thiện tốc độ của thuật toán này. Nếu muốn thuật toán chạy nhanh hơn nữa, ta chỉ cần chỉnh lại độ sai số của hàm allclose() lớn hơn nữa.
- Nhận xét 4: Việc cài đặt sai số là 1 đơn vị, trên lí thuyết thì mắt người khó có thể nhận ra sắc độ màu khi nó bị sai lệch 1 đơn vị RGB.
- Nhận xét 5: Tuy nhiên trên thực tế, ở một số hình có số lượng màu quá ít thì việc để sai số bằng 1 rất dễ gây ra trường hợp mất màu, các màu có sắc độ gần nhau bị trộn

lẫn nhau. Do đó, việc để sai số như vậy ta phải đánh đổi sự ổn định, độ chính xác của hình ảnh đều ra đặc biệt là các hình ít màu, các màu có sắc độ giống nhau.

- Nhận xét 6: Việc để sai số atol và rtol của hàm allclose() là 1 đôi khi hình ảnh xuất ra sẽ bị vỡ màu, trộn màu do số lần lặp iterations bị giảm xuống đáng kể. Dẫn đến quá trình phân cụm sẽ không thể diễn ra hoàn hảo như trường hợp sai số bằng 0.

## 7 Hình minh họa



Hình 4: Ảnh  $800 \times 454$ , 70KB trước khi chạy thuật toán



Hình 5: Ảnh sau khi chạy thuật toán với  $K = 3$



Hình 6: Ảnh sau khi chạy thuật toán với  $K = 5$



Hình 7: Ảnh sau khi chạy thuật toán với  $K = 7$



Hình 8: Ảnh  $1920 \times 1228$ , 442 KB trước khi chạy thuật toán



Hình 9: Ảnh sau khi chạy thuật toán với  $K = 3$



Hình 10: Ảnh sau khi chạy thuật toán với  $K = 5$



Hình 11: Ảnh sau khi chạy thuật toán với  $K = 7$



Hình 12: Ảnh 4,9 MB kích thước  $6000 \times 3111$  trước khi chạy thuật toán



Hình 13: Ảnh sau khi chạy thuật toán với  $K = 3$



Hình 14: Ảnh sau khi chạy thuật toán với  $K = 5$



Hình 15: Ảnh sau khi chạy thuật toán với  $K = 7$