

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



**PROJECT 1**  
**COLOR COMPRESSION**

**Giảng viên lý thuyết** Vũ Quốc Hoàng

**Giảng viên thực hành** Phan Thị Phương Uyên

Nguyễn Văn Quang Huy

**Sinh viên** Vũ Đình Chương

21127236

# Mục lục

<b>1</b>	<b>Tổng quan lý thuyết</b>	<b>2</b>
1.1	Vấn đề đặt ra . . . . .	2
1.2	Sơ lược nguồn gốc K-Means . . . . .	2
1.3	Giới thiệu K-Means . . . . .	2
<b>2</b>	<b>Mô tả K-Means</b>	<b>2</b>
2.1	Ý tưởng K-Means . . . . .	2
2.2	Thuật toán K-Means . . . . .	3
2.2.1	Mô tả thuật toán . . . . .	3
2.2.2	Pseudo-code . . . . .	4
2.2.3	Kmeans trong python . . . . .	5
<b>3</b>	<b>Mô tả các hàm phụ trợ</b>	<b>8</b>
3.1	centroids_to_image() . . . . .	8
3.2	user_input() . . . . .	9
3.3	img_process() . . . . .	10
3.4	export_img() . . . . .	10
3.5	main() . . . . .	11
<b>4</b>	<b>Kết quả và nhận xét:</b>	<b>11</b>
4.1	Kết quả . . . . .	11
4.1.1	Với $K = 3$ . . . . .	12
4.1.2	Với $K = 5$ . . . . .	12
4.1.3	Với $K = 7$ . . . . .	14
4.2	Nhận xét: . . . . .	15
4.2.1	Nhận xét chung: . . . . .	15
4.2.2	So sánh giữa 'random' với 'in_pixels' . . . . .	15
4.2.3	So sánh giữa kmeans() với kmeans của thư viện sklearn . . . . .	16
<b>5</b>	<b>Kết luận</b>	<b>16</b>
<b>6</b>	<b>Tài liệu tham khảo</b>	<b>17</b>

# 1 Tổng quan lý thuyết

## 1.1 Vấn đề đặt ra

Cài đặt chương trình giảm số lượng màu cho ảnh sử dụng thuật toán K-Means.

## 1.2 Sơ lược nguồn gốc K-Means

Thuật ngữ K-Means được giới thiệu lần đầu bởi James MacQueen vào năm 1967, mặc dù ý tưởng đã được Hugo Stein đưa ra vào năm 1956. Thuật toán tiêu chuẩn ban đầu được đề xuất bởi Stuart Lloyd của Bell Labs năm 1957 như là một kỹ thuật mã hoá modul, dù cho nó không được công bố trong tạp chí chuyên ngành cho đến năm 1982. Vào năm 1965, Edward W. Forgy công bố một phương pháp tương tự, đó là lý do nó vẫn được gọi là thuật toán Lloyd-Forgy.

## 1.3 Giới thiệu K-Means

K-means là một thuật toán phân cụm mạnh mẽ và đa dụng trong lĩnh vực học máy. Công dụng của K-means rất đa dạng và có thể được áp dụng trong nhiều lĩnh vực khác nhau. Ví dụ, nó có thể được sử dụng để phân tích hình ảnh bằng cách phân đoạn các vùng tương tự trong hình ảnh, nhận dạng mẫu trong xử lý ngôn ngữ tự nhiên hoặc phân nhóm dữ liệu trong nghiên cứu thị trường.

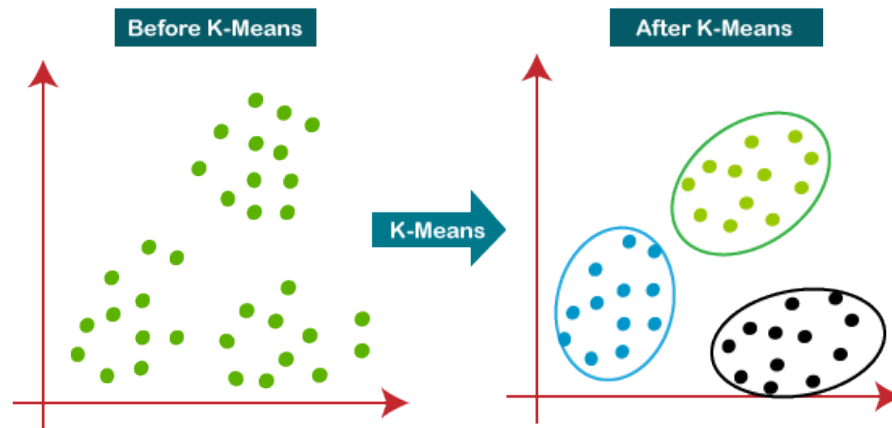
K-means cũng rất hữu ích trong việc phân tích dữ liệu và trực quan hóa. Nó có thể giúp phân nhóm khách hàng dựa trên hành vi mua hàng, phân loại dữ liệu trong lĩnh vực tài chính và kinh doanh, cũng như tạo ra các biểu đồ và biểu đồ trực quan để hiển thị sự phân cụm của dữ liệu. Ngoài ra, K-means còn được sử dụng trong nghiên cứu khoa học để phân tích dữ liệu sinh học, xử lý tín hiệu và nghiên cứu về môi trường.

# 2 Mô tả K-Means

## 2.1 Ý tưởng K-Means

K-Means là thuật toán dựa trên kỹ thuật centroid với mỗi cluster (cụm) là một cụm với trung tâm của cụm các pixel, và lưu vị trí của các màu vào labels (nhãn) là chỉ số của các centroid mà các pixel thuộc về thuật toán K-Means nhằm tới việc giảm số lượng màu, đặc biệt là các màu ít xuất hiện.

Ví dụ áp dụng kmeans:



## 2.2 Thuật toán K-Means

### 2.2.1 Mô tả thuật toán

input: img\_1d, img, k\_clusters, max\_iter, init\_centroids

- img\_1d: Ảnh gốc đã reshape thành mảng 1 chiều
- k\_cluster: Số cluster
- max\_iter: Số lần lặp lại tối đa cho K-Means
- init\_centroids: Quy định cách thức xác định centroids ban đầu: 'random' hoặc 'in\_pixels'

input: centroids, labels

- centroids: Mảng 2 chiều lưu trữ các màu
- labels: Mảng 1 chiều đánh dấu chỉ số của các centroid mà các pixel thuộc về

Ý tưởng thực hiện:

- Từ k cluster chọn ngẫu nhiên centroids có k phần tử
- Tìm khoảng cách của các pixels tới k centroid

$$d(x, y) = \sqrt{\sum_{i=0}^n (x_i - y_i)^2}$$

- Gán vào labels cho biết pixel nào thuộc vào centroid gần nhất

$$j = \operatorname{argmin} \sqrt{\|(x_i - y_i)^2\|} \Leftrightarrow \operatorname{argmin}(d(x_i, y_i))$$

- Tìm lại giá trị của centroids bằng giá trị trung bình giá trị của các pixels theo centroid đã được gán

$$C_i = \frac{1}{\|N_i\|} \sum x_i$$

- Lặp lại quá trình cho đến khi các phần tử trong centroids không thay đổi hoặc vượt quá max\_iter lần

## 2.2.2 Pseudo-code

Quy ước:

- biến[a]: mảng 1 chiều với a dòng
- biến[a][b]: Mảng 2 chiều với a dòng b cột

### Pseudo-code:

```
def kmeans(img_1d, img, k_cluster, max_iter, init_centroids):
```

```
    Nếu init_centroid == 'random':
```

```
        centroids[k_cluster] = ngẫu nhiên từ 0-255 với k_clusters centroids mỗi centroids có 3 giá trị
```

```
    Nếu init_centroids == 'in_pixels':
```

```
        centroids[k_cluster] = Lấy ngẫu nhiên k_clusters màu trong img_1d
```

```
    Chạy thuật toán K-Means với max_iter lần
```

```
        distances[k_cluster][Số pixel trong mảng img_1d] = tính khoảng cách của các pixels với mỗi centroid
```

```
        labels[Số pixel trong mảng img_1d] = gán vị trí cho các pixels thuộc về centroid nào gần nhất
```

```
        temp_centroids[k_cluster] = trung bình giá trị của các pixels theo centroid đã được gán
```

```
        Nếu temp_centroids gần bằng hoặc bằng centroids thì break vòng lặp
```

```
        Cập nhật centroids = temp_centroids
```

```
    return centroids, labels
```

### 2.2.3 Kmeans trong python

---

```

1 def kmeans(img_1d, k_clusters, max_iter = 300, init_centroids='
    random'):
2     height_width , channels = img_1d.shape
3
4     if init_centroids == 'random':
5         centroids = np.random.randint(0, 256, (k_clusters,
channels))
6     elif init_centroids == 'in_pixels':
7         rand_centroid = np.random.choice(range(height_width),
k_clusters, replace = False)
8         centroids = img_1d[rand_centroid]
9     else:
10        raise ValueError("Invalid option: 'random' or 'in_pixels' "
)

```

---

- Dòng 4-10 random centroids: chọn số ngẫu nhiên trong khoảng với số lượng k\_cluster và trong k cluster có channels phần tử
  - Với init\_centroids == 'random' sẽ random từ 0-255 cho từng centroid
  - với init\_centroids == 'in\_pixels' sẽ random các phần tử có trong ảnh cho từng centroid
  - Khi không thỏa sẽ thông báo lên màn hình "Invalid option: 'random' or 'in\_pixels'"

---

```

1     for _ in range(max_iter):
2         distances = np.linalg.norm(img_1d[:, np.newaxis] -
centroids, axis = 2)
3
4         labels = np.argmin(distances, axis = 1)
5
6         temp_centroids = np.array([np.mean(img_1d[labels == k] if
np.any(labels == k) else centroids, axis = 0) for k in range(
k_clusters)])

```

---

- Dòng 2 distance = np.linalg.norm(img\_1d[:, np.newaxis] - centroids, axis = 2) Tính khoảng cách giữa phần tử trong img\_1d với tất cả phần tử trong centroids.
  - np.newaxis chuyển phần tử mảng img\_1d từ 2 chiều thành 3 chiều cụ thể tăng 1 chiều từ, [[R,G,B],[R, G, B]] thành [[[R,G,B],[R, G, B]]]

- Sau khi chuyển trừ với centroids là trừ lần lượt [R] [G] [B] với từng phần tử trong centroids
- axis = 2 hàm norm tính theo  $\sqrt{R^2 + G^2 + B^2}$

Ví dụ cho np.newaxis:

```
x1 = [[1, 2, 4], [2, 3, 4], [3, 4, 5]]
x1 = np.array(x1)
x1 = np.array(x1[:, np.newaxis])
print(x1)
```

```
[[[1 2 4]]
 [[2 3 4]]
 [[3 4 5]]]
```

Ví dụ cho np.linalg.norm():

```
x1 = [[1, 2, 4], [2, 3, 4], [3, 4, 5]]
x2 = [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
x2 = np.array(x2)
x1 = np.array(x1)
x1 = np.array(x1[:, np.newaxis])
x = x1 - x2
x = np.linalg.norm(x1 - x2, axis = 2)
print(x)
```

```
[[2.44948974 1.         1.41421356]
 [3.46410162 1.73205081 0.         ]
 [5.19615242 3.46410162 1.73205081]]
```

- Dòng 4 labels = np.argmin(distances, axis = 1) Lọc ra k min trong distances
  - axis = 1 duyệt lần lượt từng phần tử theo cột để tìm min index trong cột ấy
  - Lưu index min vừa tìm được trong distances vào trong labels có nghĩa là màu pixel = centroids[index]

```

x1 = [[1, 2, 4], [2, 3, 4], [3, 4, 5]]
x2 = [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
x2 = np.array(x2)
x1 = np.array(x1)
x1 = np.array(x1[:, np.newaxis])
x = x1 - x2
dis = np.linalg.norm(x1 - x2, axis = 2)
print('distance:\n', dis)
labels = np.argmin(dis, axis = 1)
for i in labels:
    print('Min index của các cột lần lượt là', i )

distances:
[[2.44948974 1.         1.41421356]
 [3.46410162 1.73205081 0.         ]
 [5.19615242 3.46410162 1.73205081]]
Min index của các cột distances lần lượt là [1 2 2]

```

- Dòng 6 `temp_centroids = np.array([np.mean(img_1d[labels == k] if np.any(labels == k) else centroids, axis = 0) for k in range(k_clusters)])`
  - Nếu k tồn tại trong labels thì `img_1d[labels == k]` thì lấy tất cả các phần tử trong labels mang giá trị k
  - Dùng `np.mean()` tính trung bình của các phần tử đã xác nhận trong `img_1d[labels == k]`
  - Nếu k không tồn tại sẽ giữ `centroids[k]` cũ

Ví dụ cho việc không tìm thấy k trong labels:

Như hình ví dụ cho việc tìm labels dùng `argmin` cho thấy index trong labels chỉ có [1 2 2] không có index 0 nên index 0 trong check ở ví dụ hay `temp_centroids` ở `kmeans` sẽ ra nan và sẽ bị lỗi không chạy tiếp được.



```

x1 = [[1, 2, 4], [2, 3, 4], [3, 4, 5]]
x2 = [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
x2 = np.array(x2)
x1 = np.array(x1)
x1 = np.array(x1[:, np.newaxis])
x = x1 - x2
dis = np.linalg.norm(x1 - x2, axis = 2)
print('distances:\n', dis)
labels = np.argmin(dis, axis = 1)
check = np.array([np.mean(x1[labels == k], axis = 0) for k in range(3)])
print (check)

```

```

[[[nan nan nan]]]

[[1.  2.  4. ]]

[[2.5 3.5 4.5]]]

```

```

1     if np.allclose(centroids, temp_centroids, atol = 1):
2         break
3
4     centroids = temp_centroids
5
6     return centroids, labels

```

- Dòng 11 np.allclose(centroids, temp\_centroids, atol = 1)
  - Dùng để kiểm tra xem centroids và temp\_centroids có gần bằng nhau chưa
  - giá trị tuyệt đối của (temp\_centroids - centroids) <= atol = 1 thì dừng
  - trả về centroids và labels cuối cùng tìm được

## 3 Mô tả các hàm phụ trợ

### 3.1 centroids\_to\_image()

def centroids\_to\_image(centroids, labels, img):

Input: centroids, labels, img

- centroids: mảng chứa k\_clusters centroid màu
- labels: mảng đánh dấu các pixels thuộc về centroids gần nhất

- `img`: Ảnh gốc chưa reshape

Output: `new_img`

- `new_img` ảnh sau khi nén màu

Chuyển đổi từ centroids và labels trở về thành tấm ảnh hoàn chỉnh

- `centroids[labels]` gán các giá trị màu vào lại theo các phần tử của labels
- reshape mảng vừa thu được thành ảnh theo shape ban đầu

### 3.2 `user_input()`

`def user_input()`: Output: `img_name`, `k_clusters`, `max_iter`, `init_centroids`, `file_extension`

- `img_name` đường dẫn tới ảnh
- `k_clusters` số màu còn lại của ảnh
- `max_iter` số lần lặp tối đa của K-Means
- `init_centroids` lựa chọn centroids ban đầu
- `file_extension`: Định dạng của file ảnh đầu ra

Dùng để nhập thông tin từ người dùng lần lượt là:.

- `img_name`: Tên ảnh
- `k_clusters`: Số màu còn lại của ảnh
- `max_iter`: Số lần lặp tối đa trong kmeans
- `init_centroids`: Cách thức lựa chọn centroids ban đầu
- `file_extension`: Định dạng của file ảnh đầu ra
- Cuối cùng trả về cho tất cả các thông tin vừa đi vào

### 3.3 `img_process()`

`def img_process(img_name):` Input: `img_name`

- `img_name` đường dẫn tới ảnh

Output: `img_array.reshape(-1, 3), img`

- `img_array.reshape(-1, 3)` trả về ảnh đã được reshape thành mảng một chiều
- `img` trả về ảnh gốc

Dùng để xử lý ảnh chuyển ảnh sang mảng một chiều và in ảnh gốc ra màn hình trước khi đưa vào hàm `kmeans`

- Mở ảnh bằng hàm `open` trong `PIL.Image`
- Chuyển ảnh sang mảng của `numpy` bằng `np.array`
- trả về ảnh đã reshape về một chiều bằng `np.reshape(-1, 3)` và trả về ảnh gốc khi mới mở ảnh

### 3.4 `export_img()`

`def export_img(new_img, file_extension)` Input: `new_img`, `file_extension`

- `new_img` ảnh sau khi giảm màu
- `file_extension` phần mở rộng của ảnh sẽ tạo ra

Output: Ảnh ở đã giảm màu ở folder

Dùng để in ra ảnh sau khi đã chỉnh màu và lưu định dạng ảnh ra folder

- Chuyển đổi ảnh từ mảng `numpy` trở về mảng ban đầu khi vừa mở ảnh bằng `PIL.Image.fromarray` ta được `Image.fromarray(new_img.astype(np.uint8))`
- Nếu `file_extension` là 'pdf' thì sẽ lưu định dạng ảnh là pdf ra folder
- Nếu `file_extension` là 'png' thì sẽ lưu định dạng ảnh là png ra folder

### 3.5 main()

Output: Ảnh đã qua xử lý giảm màu Hàm main() lần lượt lấy input từ người dùng, reshape ảnh và chạy hàm giảm màu kmeans() sau đó tính thời gian và lưu định dạng ảnh ra folder cũng như cho người dùng sau khi xong hàm kmeans() bằng hàm export\_img() và hàm centroids\_to\_image()

- Lấy img\_name, k\_clusters, max\_iter, init\_centroids, file\_extension từ hàm user\_input()
- Đọc ảnh và reshape ảnh bằng hàm img\_process()
- Lấy centroids và labels đã được giảm màu từ hàm kmeans(), đồng thời tính thời gian xử lý bằng hàm time() trong thư viện time
- Xử lý ngược lại để trả về ảnh sau khi giảm màu bằng hàm centroids\_to\_image()
- Lưu định dạng ảnh ra folder
- Cuối cùng trình ảnh ra màn hình sau khi đã giảm màu

## 4 Kết quả và nhận xét:

### 4.1 Kết quả

Sau khi chạy thử kmean với k lần lượt là 3, 5, 7 và init\_centroids là 'random' và 'in\_pixels':

Ảnh gốc



**4.1.1 Với  $K = 3$** **Random****In\_pixels****4.1.2 Với  $K = 5$** **Random**

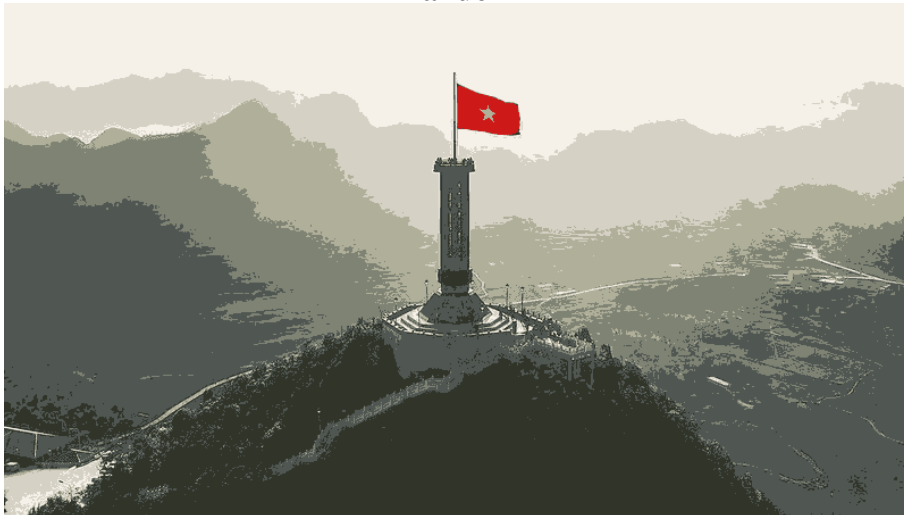


In\_pixels



#### 4.1.3 Với $K = 7$

Random



In\_pixels



## 4.2 Nhận xét:

### 4.2.1 Nhận xét chung:

Độ phức tạp của kmeans là:  $O(\text{height\_width} * K\_cluster * \text{max\_iter} * \text{channels})$ .

Tuy nhiên, thuật toán vẫn chưa tối ưu, ở xử lý centroids dẫn đến chương trình lặp lại nhiều vòng hơn dẫn tới thời gian xử lý lâu hơn.

### 4.2.2 So sánh giữa 'random' với 'in\_pixels'

- Về hình ảnh: Cả random và in\_pixels tương đối giống nhau về chất lượng hình ảnh. Cấu trúc của ảnh vẫn được giữ nguyên qua các số k\_clusters khác nhau.
- Về thời gian xử lý: Cả 2 random và in\_pixels đều không quá ổn định về thời gian vì thuật toán dựa trên random centroids. Sẽ có sự trên lệch về thời gian dựa trên điều kiện dừng tại np.allclose() atol càng lớn dừng càng nhanh và sự ổn định của ảnh đầu ra càng không ổn định.

Bảng đo thời gian giữa random và in\_pixels lần 1 với max\_iter = 50:

K	random	in_pixels
3	0.44	0.49
5	1.1	1.6
7	1.75	1.15

Bảng đo thời gian giữa random và in\_pixels lần 2 với max\_iter = 50:



K	random	in_pixels
3	0.96	0.68
5	1.19	1.84
7	1.89	2.4

### 4.2.3 So sánh giữa kmeans() với kmeans của thư viện sklearn

- Về hình ảnh: Cả random và in\_pixels trong cả 2 cách đều tương đối giống nhau về chất lượng hình ảnh. Cấu trúc của ảnh vẫn được giữ nguyên qua các số k\_clusters khác nhau tuy nhiên ngay cả random hay in\_pixels kmeans() trong sklearn cho ra kết quả đồng nhất và ổn định hơn.
- Về thời gian xử lý: kmeans() trong sklearn có tốc độ nhanh hơn nhiều so với kmeans().

Bảng đo thời gian giữa random và in\_pixels và kmeans() trong sklearn với max\_iter = 50:

K	random	in_pixels	sklearn.KMeans()
3	0.96	0.68	0.22
5	1.19	1.84	0.45
7	1.89	2.4	0.79

## 5 Kết luận

Thuật toán Kmeans tuy nhanh nhưng vẫn còn nhiều hạn chế:

- Cần biết số lượng cluster cần clustering
- Nghiệm cuối cùng phụ thuộc vào các centers được khởi tạo ban đầu
- Các cluster cần có số lượng điểm gần bằng nhau nhưng kết quả cuối cùng là không chính xác
- K-Means chỉ phát hiện được các cụm có dạng hình cầu do dựa trên tính chất về khoảng cách.

Mặc dù có những hạn chế, K-means clustering vẫn cực kỳ quan trọng trong Machine Learning và là nền tảng cho nhiều thuật toán phức tạp khác sau này.

## 6 Tài liệu tham khảo

### Tài liệu

- [1] Ý tưởng kmeans
- [2] Ý tưởng kmeans
- [3] Ý tưởng - thuật toán kmeans
- [4] Khái niệm lý thuyết - Ý tưởng kmeans
- [5] Hàm numpy
- [6] Broadcasting cho kmeans
- [7] Thuật toán kmeans
- [8] Kmeans trong sklearn
- [9] Hướng dẫn Kmeans trong sklearn