TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM
KHOA CÔNG NGHỆ THÔNG TIN

TOÁN ỨNG DỤNG VÀ THỐNG KÊ

ĐỒ ÁN 1

COLOR COMPRESSION

Giảng viên hướng dẫn:

Vũ Quốc Hoàng

Nguyễn Văn Quang Huy

Ngô Đình Hy

Phan Thị Phương Uyên

Sinh viên thực hiện:

21127232 – Nguyễn Thanh Bình

Thành phố Hồ Chí Minh, Tháng 07/2023

MỤC LỤC

I. Ý tưởng thực hiện và thuật toán K-means clustering 3

1. Ý tưởng thực hiện 3

2. Nội dung thuật toán K-means clustering 3

II. Mô tả các hàm 4

1. Hàm initialize_centroids 4

2. Hàm calc_dist_mat 4

3. Hàm update_centroids 6

4. Hàm kmeans 7

5. Hàm recolorize_image 8

III. Kết quả thử nghiệm 9

1. Lần 1: K = 3 9

a. Ảnh 1 9

b. Ảnh 2 10

2. Lần 2: K = 5 11

a. Ảnh 1 11

b. Ảnh 2 12

3. Lần 3: K = 7 14

a. Ảnh 1 14

b. Ảnh 2 15

IV. Nhận xét kết quả chương trình 16

1. Điểm tốt 16

2. Điểm hạn chế 16

V. Tài liệu tham khảo 17

I. Ý tưởng thực hiện và thuật toán K-means clustering

1. Ý tưởng thực hiện

Ngày nay, máy tính lưu trữ các bức ảnh dưới dạng các ma trận điểm ảnh, mỗi điểm ảnh sẽ là một vector chứa giá trị của các kênh màu. Có rất nhiều loại ảnh được sử dụng trong thực tế như ảnh xám, ảnh màu, ... Đối với các bức ảnh màu, số lượng kênh màu thường là 3 đại diện cho 3 kênh màu đỏ (R), xanh lá (G) và xanh lam (B). Mỗi giá trị của 1 kênh màu nằm trong đoạn [0; 255], vì thế khi tổ hợp 3 kênh màu chúng ta có thể tạo ra tối đa xấp xỉ 17 triệu màu.

Trong đồ án này, chúng ta cần sử dụng thuật toán **K-means clustering** là một loại của nhóm thuật toán Unsupervised Learning được sử dụng trong phân tích tính chất cụm của dữ liệu. Mục đích của thuật toán này là phân vùng dữ liệu thành k cụm (k clusters) khác nhau, giúp chúng ta xác định được dữ liệu của chúng ta thực sự thuộc về nhóm nào. Cụ thể hơn, đồ án yêu cầu chúng ta nén một bức ảnh có số lượng màu lớn thành một bức ảnh có số lượng màu nhỏ (từ 3 đến 7 màu), do đó chúng ta cần gom nhóm các màu của bức ảnh gốc lại với nhau bằng cách tạo ra k điểm màu trung tâm (k centroids) đại diện cho k cụm (k clusters) khác nhau. Sau đó ta thực hiện phân vùng các điểm ảnh về các cụm mà màu của điểm ảnh đó gần với màu trung tâm của cụm đó nhất.

2. Nội dung thuật toán K-means clustering

Thuật toán K-means gồm 4 bước cơ bản như sau:

1. Khởi tạo k điểm màu trung tâm cho k cụm bằng cách chọn ngẫu nhiên hoặc lấy ra bất kì từ bức ảnh gốc ban đầu.
2. Xét từng điểm ảnh trong bức ảnh gốc, tính khoảng cách từ điểm ảnh đó tới k điểm màu trung tâm. Xác định điểm màu trung tâm của điểm ảnh đó là 1 trong k điểm màu trung tâm gần nó nhất.
3. Sau khi phân vùng các điểm ảnh về k cụm, tính toán lại vị trí của các điểm màu trung tâm của từng cụm để đảm bảo điểm màu trung tâm luôn nằm chính giữa cụm.
4. Lặp lại bước 2 và 3 cho đến khi vị trí các điểm màu trung tâm gần như không thay đổi.

II. Mô tả các hàm

1. Hàm initialize_centroids

```
1  def initialize_centroids(img_1d, k_clusters, init_centroids):
2      if init_centroids == 'random':
3          return np.random.choice(256, size=(k_clusters, img_1d.
           shape[1]), replace=False)
4      elif init_centroids == 'in_pixels':
5          return img_1d[np.random.choice(img_1d.shape[0],
           k_clusters, replace=False)]
```

- Hàm nhận vào các tham số gồm:
  - *img_1d* là ma trận 1 chiều các điểm ảnh.
  - *k_cluster*s là số lượng cụm cần phân vùng.
  - *init_centroids* là kiểu chọn centroid, có 2 lựa chọn là ‘random’ hoặc là ‘in_pixels’.
- Hàm này trả về ma trận 1 chiều các centroid.
- Nếu *init_centroids* = ‘random’ thì lấy ngẫu nhiên k điểm ảnh bất kì. Mỗi điểm ảnh có kích cỡ bằng số kênh màu, mỗi kênh nằm trong đoạn [0;255] bằng cách dùng hàm *np.random.choice().*
- Nếu *init_centroids* = ‘in_pixels’ thì lấy ra bất kì k điểm ảnh thuộc *img_1d* bằng cách dùng hàm *np.random.choice()* để chọn ra ngẫu nhiên *k_clusters* chỉ số nằm trong nửa khoảng từ 0 đến số điểm ảnh trong *img_1*d. Sau đó dùng *img_1d[]* để trả về mảng các điểm ảnh tương ứng.
- Có thể truyền thêm tham số *replace=False* vào trong hàm *np.random.choice()* để không bị lấy trùng giá trị centroid.

2. Hàm calc_dist_mat

```
1  def calc_dist_mat(img_1d, centroids):
2      # Lấy ý tưởng dựa trên broadcasting, bằng cách reshape hai ma trận img_1d và
       centroids, để có thể trừ cùng lúc tất cả phần tử thuộc img_1d cho tất cả phần tử
       thuộc centroids
3      a = img_1d.reshape(img_1d.shape[0], 1, img_1d.shape[1])
4      b = centroids.reshape(1, centroids.shape[0], img_1d.shape[1])
5      return np.sqrt(np.square(a - b).sum(axis=2))
```

- Hàm nhận vào các tham số gồm:
  - *img_1d* là ma trận 1 chiều các điểm ảnh.

- *centroids* là ma trận 1 chiều các centroid.

- Hàm này trả về ma trận 2 chiều, trong đó mỗi dòng thứ i trong ma trận là khoảng cách từ điểm ảnh thứ i của *img_1d* đến tất cả các centroid của *centroids*.

Ý tưởng dùng broadcasting để tính toán nhanh khoảng cách:

- Ban đầu *img_1d* là ma trận có shape = (số pixel, số channel) và *centroids* là ma trận có shape = (số centroid, số channel).
- Để đơn giản ta lấy 1 ví dụ cụ thể có số pixel = 4, số centroid = 2 và số channel = 3 như sau:

```
1  img_1d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
2  centroids = np.array([[2, 4, 6], [3, 5, 7]])
```

Trước hết, để tìm được khoảng cách, ta cần tìm được hiệu giữa từng vector trong *img_1d* với tất cả vector trong *centroids*, nhưng không được dùng vòng lặp vì sẽ tốn rất nhiều thời gian chạy.

Nếu ta lấy img_1d trừ trực tiếp cho centroids, tức là lấy ma trận có cỡ 4x3 trừ cho ma trận có cỡ 2x3 thì chương trình sẽ báo lỗi vì hai cỡ này không tương thích.

```
   1  img_1d - centroids
⊗ 0.0s

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[210], line 1
----> 1 img_1d - centroids

ValueError: operands could not be broadcast together with shapes (4,3) (2,3)
```

Do đó ta cần phải chèn thêm 1 chiều có cỡ bằng 1 nữa vào cho các ma trận này để có thể sử dụng được broadcasting ép hai ma trận này về cùng 1 kích cỡ tương thích nhau và có thể trừ được.

Dựa trên ý tưởng đó, ta có thể chèn thêm 1 chiều một cách phù hợp như sau:

```
1  img_1d = img_1d.reshape(4, 1, 3)
2  centroids = centroids.reshape(1, 2, 3)
```

Khi đó, việc trừ hai ma trận này sẽ được broadcasting ép cả hai về cùng 1 shape là (4, 2, 3), do đó có thể trừ hai ma trận trực tiếp cho nhau thu được ma trận hiệu như mong muốn.

```
array([[[-1, -2, -3],
        [-2, -3, -4]],

       [[ 2,  1,  0],
        [ 1,  0, -1]],

       [[ 5,  4,  3],
        [ 4,  3,  2]],

       [[ 8,  7,  6],
        [ 7,  6,  5]]])
```

Từ đó ta có thể dùng hàm *np.square()* để bình phương tất cả các phần tử trong ma trận hiệu thu được và dùng hàm *sum(axis=2)* để tổng tất cả lại theo chiều ngang của ma trận 3 chiều và thu được ma trận 2 chiều. Cuối cùng dùng hàm *np.sqrt()* để tính căn bậc hai của tất cả các phần tử thu được ma trận khoảng cách 2 chiều như mong muốn.

```
1  np.sqrt(np.square(img_1d - centroids).sum(axis=2))
```
✓ 0.0s

```
array([[ 3.74165739,  5.38516481],
       [ 2.23606798,  1.41421356],
       [ 7.07106781,  5.38516481],
       [12.20655562, 10.48808848]])
```

- Từ ý tưởng của một ví dụ cụ thể đó, ta tổng quát hóa và dùng cho bài toán.

3. Hàm update_centroids

```python
def update_centroids(img_1d, centroids, labels):
    for k in range(centroids.shape[0]):
        # Lấy ra các điểm màu thuộc về cluster thứ k
        cluster_points = img_1d[labels == k]
        # Update lại cluster thứ k nếu có điểm màu
        if len(cluster_points) > 0:
            centroids[k] = np.mean(cluster_points, axis=0)
```

- Hàm nhận vào các tham số gồm:
  - *img_1d* là ma trận 1 chiều các điểm ảnh.
  - *centroids* là ma trận 1 chiều các centroid.
  - *labels* là mảng 1 chiều cho biết điểm ảnh thứ i thuộc về cluster k nào bằng cách dán nhãn labels[i] = k.
- Hàm này dùng để cập nhật lại các centroid mới cho các cluster, với mỗi cluster k ta lấy ra các điểm ảnh được gán label = k trong ma trận ảnh, sau đó tính trung bình các điểm ảnh đó để thu được 1 giá trị centroid mới cho cluster k. Trong trường hợp cluster k đó không có điểm ảnh nào thuộc về thì không cần cập nhật lại centroid.

4. Hàm kmeans

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids='random'):
    centroids = initialize_centroids(img_1d, k_clusters, init_centroids).astype(float)
    labels = np.zeros(img_1d.shape[0])
    while max_iter > 0:
        # Lưu trữ centroids cũ trước khi update
        pre_centroids = np.copy(centroids)
        # Tính labels
        labels = calc_dist_mat(img_1d, centroids).argmin(axis=1)
        update_centroids(img_1d, centroids, labels)
        # Nếu mức độ thay đổi của centroids mới so centroids cũ không
        # quá hằng số Epsilon cho trước thì dừng
        if(np.allclose(pre_centroids, centroids, atol=1)):
            break
        max_iter -= 1
    return centroids, labels
```

- Hàm cài đặt thuật toán K-means theo các bước của thuật toán.
- Hàm nhận vào các tham số:
  - *img_1d* là ma trận 1 chiều các điểm ảnh.
  - *k_clusters* là số lượng cụm.
  - *max_iter* là số lần lặp lại tối đa, tránh trường hợp các centroid không hội tụ được thì chương trình luôn được dừng.
  - *init_centroids* là kiểu chọn centroid, có 2 lựa chọn là ‘random’ hoặc là ‘in_pixels’.
- Hàm trả về ma trận *centroids* và *labels*.
- Bước 1 của thuật toán được thực hiện bằng cách dùng hàm *initialize_centroids()*.

- Bước 2 được thực hiện bằng cách dùng hàm *calc_dist_mat()* để thu được ma trận khoảng cách và hàm *argmin(axis=1)* để lấy ra các chỉ số của các cột có giá trị khoảng cách nhỏ nhất thuộc từng dòng trong ma trận 2 chiều khoảng cách. Từ đó lưu vào ma trận *labels*.
- Bước 3 được thực hiện bằng cách dùng hàm *update_centroids()* để cập nhật centroids mới.
- Bước 4 được thực hiện bằng cách dùng vòng lặp while với số lần *max_iter* cho trước, ma trận *pre_centroids* để lưu trữ ma trận *centroids* cũ và hàm *np.allclose()* để so sánh mức độ thay đổi của ma trận *centroids* mới so với ma trận *pre_centroids* với hằng số Epsilon cho trước (trong trường hợp này là xấp xỉ 1).

5. Hàm recolorize_image

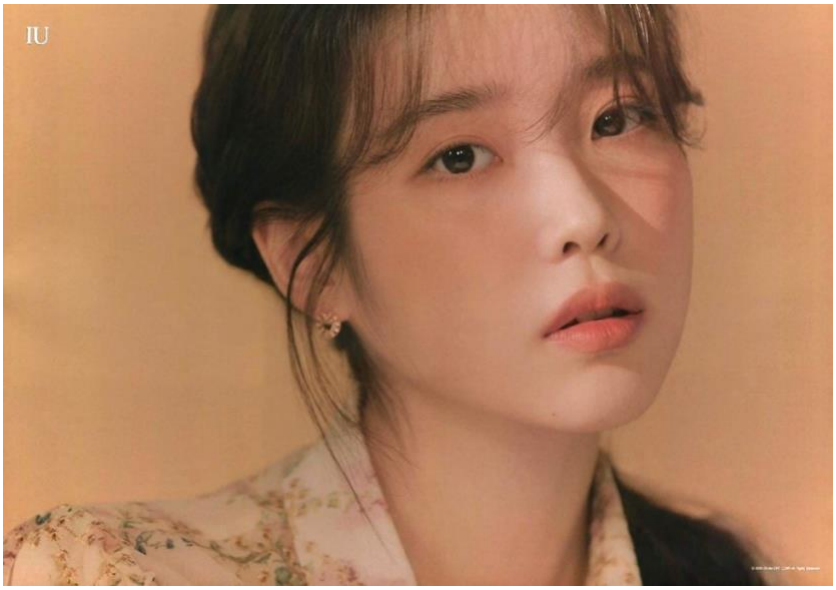
```
1 def recolorize_image(img_1d, k_clusters, centroids, labels):
2   # Tạo ma trận 0 có kích thước bằng img_1d
3   recolorized_img = np.zeros((img_1d.shape[0], img_1d.shape[1]))
4   for k in range(k_clusters):
5     # Cộng các phần tử của ma trận 0 (thỏa điều kiện có labels tại đó = k) cho
      centroid k, tương đương với phép gán centroid k cho các phần tử có label tương ứng
6     recolorized_img[labels == k] += centroids[k]
7   return recolorized_img
```

- Hàm nhận vào các tham số gồm:
  - *img_1d* là ma trận 1 chiều các điểm ảnh.
  - *k_clusters* là số lượng cụm.
  - *centroids* là ma trận 1 chiều các centroid.
  - *labels* là mảng 1 chiều cho biết điểm ảnh thứ i thuộc về cluster k nào bằng cách dán nhãn labels[i] = k.
- Hàm trả về bức ảnh sau khi đã gán màu của các centroid k lên các điểm ảnh được dán nhãn label = k tương ứng.

III. Kết quả thử nghiệm

Chạy thử nghiệm với hai bức ảnh sau:

Ảnh 1: img.jpeg



Ảnh 2: img-1.jpeg

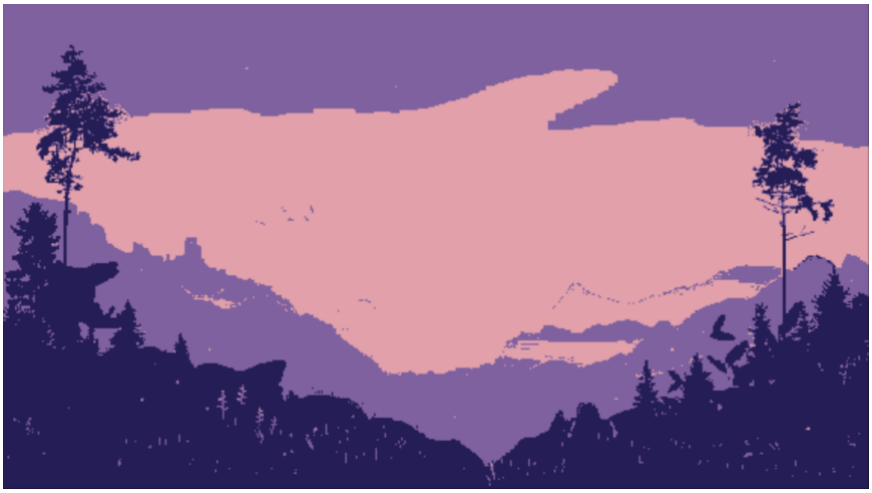
1. **Lần 1: K = 3**
   a. **Ảnh 1**

      Ma trận centroids của chương trình:

```
[[ 39.65850177  29.52183627  88.02659453]
[237.04770266 156.14018145 169.02595382]
[132.8642718   95.85869793 163.79692838]]
```

Ma trận centroids của Sklearn:

```
[[ 39.70378062  29.56070798  88.07674902]
 [133.31289518  95.99141314 163.81627257]
 [237.3557491  156.45150533 169.06887593]]
```



b. Ảnh 2

Ma trận centroids của chương trình:

```
[[ 52.42593796  42.48456538  31.67346768]
[149.21644919 106.77986745  81.4328097 ]
[191.25182198 146.87946294 112.20755125]]
```

Ma trận centroids của Sklearn:

```
[[147.63677607 105.52453583  80.40343483]
 [190.98070613 146.58786106 111.99314095]
 [ 51.33672356  41.82037172  31.21180065]]
```

2. Lần 2: K = 5

a. Ảnh 1

Ma trận centroids của chương trình:

```
[[ 99.77672533  90.0717216  161.88573329]
[249.15853228 196.88246172 180.32360947]
[232.35928575 133.49386889 162.24244092]
[ 38.94019101  28.99110236  87.36033864]
[163.00799359 100.95983832 164.53245864]]
```

Ma trận centroids của Sklearn:

```
[[234.85044048 135.77739163 162.51192112]
 [ 38.98860231  29.03160651  87.41451208]
 [102.27490544  90.32002885 162.01757252]
 [249.13720294 198.63872522 180.89276092]
 [167.18125187 102.2233268  164.50082411]]
```

b. Ảnh 2

Ma trận centroids của chương trình:

```
[[155.92959205 111.63655672  85.63712765]
[202.98571127 157.63951522 120.73152883]
[183.51318654 139.76221577 106.58846408]
[ 40.62462937  35.60476109  27.07862247]
[106.94081482  76.00137003  55.53117668]]
```

Ma trận centroids của Sklearn:

```
[[ 40.00590003  35.27508751  26.87007557]
 [182.6977067  138.91320714 105.97921001]
 [104.44948928  74.33108722  54.17307521]
 [202.22270789 156.94990756 120.12933269]
 [154.39654634 110.3770511   84.62641038]]
```

3. Lần 3: K = 7

a. Ảnh 1

Ma trận centroids của chương trình:

```
[[ 55.65091864  38.91341106 104.34867757]
[232.91218108 133.7857095  162.21959419]
[107.4127082   79.98597947 154.43822556]
[ 84.22434117 135.9509723  196.57692019]
[ 28.26515646  22.76132217  76.38194799]
[249.15789474 196.69882008 180.25485091]
[165.68512219 101.55694556 164.52192299]]
```

Ma trận centroids của Sklearn:

```
[[ 98.86621583  75.49369341 150.13396214]
 [242.50967076 143.54793768 163.36201272]
 [ 38.01890375  28.4780751   86.57578766]
 [197.8704823  112.52995239 162.54183399]
 [148.75647625  95.94049396 164.30346574]
 [ 83.49692875 136.27994472 196.86640049]
 [249.12552646 201.70119088 181.97112359]]
```

b. Ảnh 2

Ma trận centroids của chương trình:

```
[[196.11006379 150.96743364 114.47990382]
[ 93.63964676  66.962666    48.42018249]
[137.02038714  96.91813575  72.80126277]
[211.11053828 166.19353934 131.14245218]
[ 38.12831556  34.24951416  26.21182835]
[162.90785183 117.45675069  90.58240293]
[181.50772156 138.13498342 105.11495141]]
```

Ma trận centroids của Sklearn:

```
[[ 36.83106796  33.51369552  25.7327274 ]
 [195.66069817 150.55351543 114.26773349]
 [159.727897   114.69162854  88.31493657]
 [ 87.62614974  63.01181881  45.44963406]
 [180.62271942 137.16770605 104.34591158]
 [130.49130254  92.19183689  68.81358245]
 [211.00735982 166.07572947 130.99340683]]
```

IV. Nhận xét kết quả chương trình

1. Điểm tốt

Nhìn chung, kết quả chạy của chương trình khá tốt và sát với lại kết quả chạy hàm *Kmeans()* của thư viện Sklearn. Các ma trận centroids kết quả của chương trình sau khi thực hiện một vài phép biến đổi sơ cấp trên dòng thì gần như xấp xỉ với các ma trận centroids khi sử dụng thư viện Sklearn. Khi so sánh màu sắc các bức hình trong từng trường hợp của k, bằng mắt thường, ta có thể thấy chúng gần như giống nhau, không có quá nhiều điểm khác biệt.

2. Điểm hạn chế

Ta có thể thấy khi k càng lớn và số lượng *max_iter* = 20 không thay đổi, kết quả chương trình của bức hình phong cảnh (img.jpeg) càng trở

nên khác so với thư viện Sklearn. Cụ thể ở k = 3 gần như giống nhau hoàn toàn, nhưng ở k = 5 có sự khác nhau một chút, và ở k = 7 thì sự khác nhau rõ hơn.

Ngược lại, ở bức hình chân dung (img-1.jpeg), kết quả cho ra ở cả 3 k gần như đều tốt và giống nhau hoàn toàn, tuy có sai số một chút ở ma trận khi k càng lớn nhưng hình ảnh không có quá nhiều sự khác biệt.

Lý do có sự khác nhau về kết quả chạy giữa hai bức hình này là bởi vì bức hình phong cảnh có số màu gốc là 54549 màu, gấp xấp xỉ 1.6 lần số màu gốc của bức hình chân dung với chỉ 32305 màu. Vì thế sai số của bức hình chân dung khi chạy thuật toán sẽ thấp hơn so với bức hình phong cảnh. Một nguyên nhân nữa khiến cho việc sai số đó khá lớn là vì số lần lặp *max_iter* còn tương đối nhỏ (20 lần), nên để khắc phục được sai số khi k càng lớn, ta có thể tăng số lần *max_iter* lên.

V. Tài liệu tham khảo

[1] Lab 02 + Project 01

[2] geeksforgeeks.org

[3] machinelearningcoban.com

[4] stackoverflow.com

[5] blog.luyencode.net

[6] numpy.org

[7] en.wikipedia.org

[8] iq.opengenus.org