VIETNAM NATIONAL UNIVERSITY HCMC
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

——————o0o——————

Project 1: Color Compression

Subject: Applied Mathematics and Statistics

Lecturer: **Mr. Vũ Quốc Hoàng**
Mr. Nguyễn Văn Quang Huy
Mr. Nguyễn Ngọc Toàn
Ms. Phan Thị Phương Uyên

Class: **22CLC05**

Student: **22127119 – Hồ Phước Hoàn**

HCMC, 6/2024

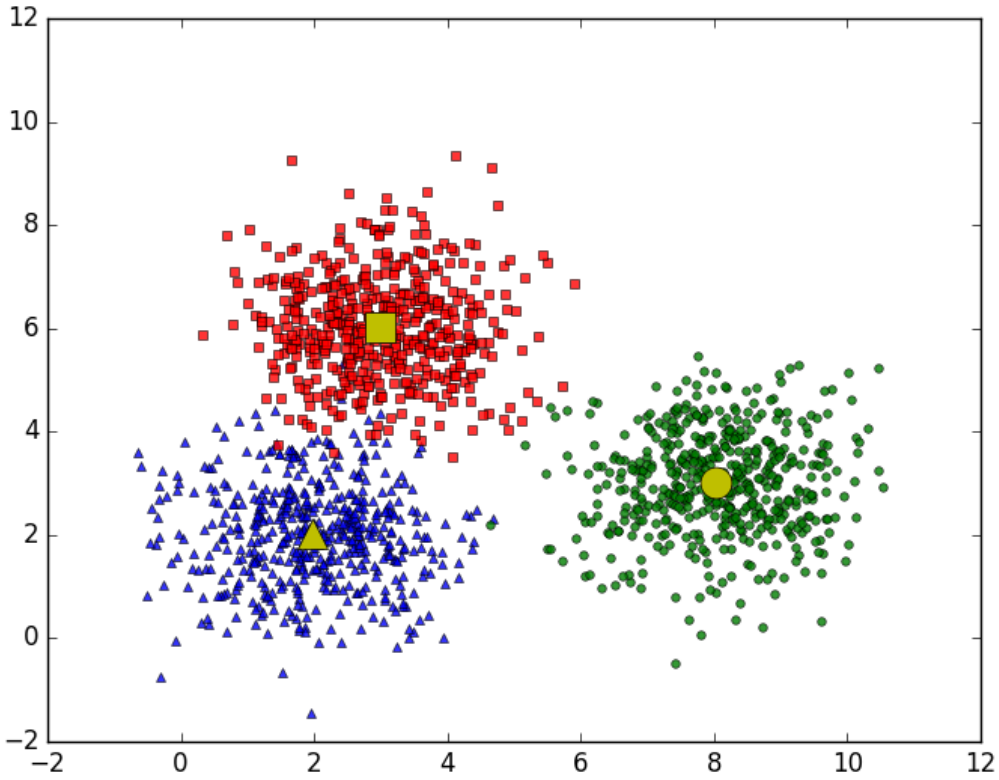
Table of Contents

I. K-Means

Phân cụm K-Means là 1 phương pháp lượng tử hóa vector dùng để phân các điểm dữ liệu cho trước vào các cụm khác nhau. Phân cụm K-Means có nhiều ứng dụng, nhưng được sử dụng nhiều nhất trong Trí tuệ nhân tạo và Học máy (cụ thể là Học không có giám sát). [1]

Trong thuật toán K-Means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thể nào để phân dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau. [2]

Mỗi cụm dữ liệu là tập hợp các điểm dữ liệu gần nhau trong 1 không gian (cluster). Giả sử mỗi cluster có một điểm trung tâm (centroid) màu vàng và những điểm xung quanh centroid thuộc vào cùng nhóm với centroid đó.



Bài toán 3 cluster

Nếu như ban đầu các điểm dữ liệu chưa được gắn nhãn thì bài toán đặt ra làm sao để ta gắn nhãn cho các điểm dữ liệu? Ta sẽ nghiên cứu qua thuật toán K-Means.

Thuật toán

Đầu vào: dữ liệu X và số lượng cluster cần tìm K

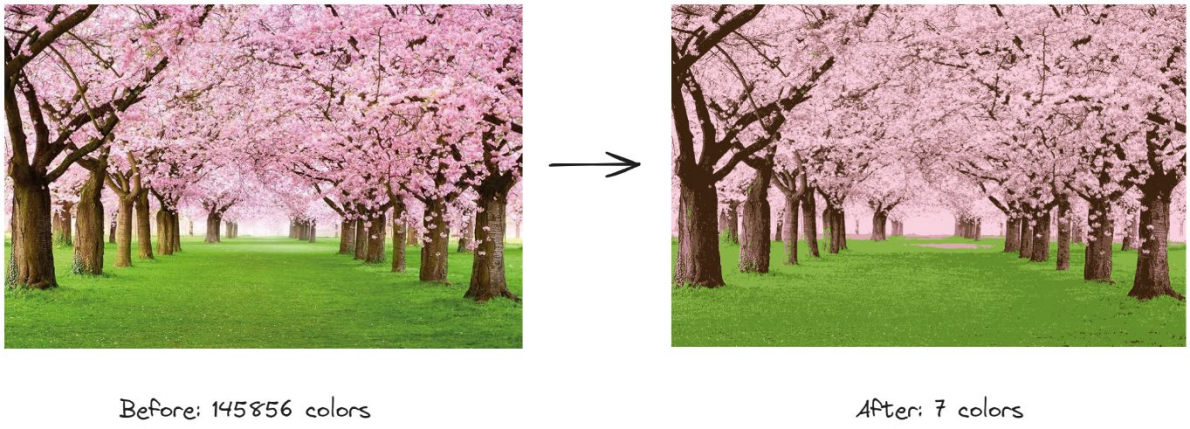
Đầu ra: Các centroid M và label vector cho từng điểm dữ liệu Y

1. Chọn K điểm bất kỳ làm các centroid ban đầu
2. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất

3. Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.
4. Cập nhật centroid cho từng cluster bằng các lấy trung bình cộng của tất cả các điểm dữ liệu đã được gán vào cluster đó sau bước 2
5. Quay lại bước 2

II. Color Compression

Với bài toán nén màu (color compression) thì cũng có những đặc điểm chung với các bài toán dùng K-Means, đó là đều tìm label để gán cho các điểm dữ liệu thông qua label của centroid gần nó nhất. Với mỗi điểm ảnh là một màu sắc khác nhau, mục tiêu là làm giảm số lượng màu sắc xuống bằng với số lượng màu mà ta mong muốn. Trong đó mỗi cluster sẽ là một màu đại diện trong đó.



Bức ảnh trước khi nén (145856 màu) và sau khi nén màu (7 màu) [3]

Thuật toán: Cũng giống như thuật toán K-Means, nhưng dữ liệu đầu vào là hình ảnh có cấu trúc 2D (heigth, width, channels). Ta cần phải có bước chuyển đổi sang 1D (height * width, channels) để thuật toán có thể khả thi trước khi bắt đầu thuật toán. Và sau cùng, ta chuyển đổi kích thước sang 2D trở lại để có thể hiển thị được hình ảnh.

1. Cài đặt chi tiết

Source: https://github.com/PhuocHoan/Color-Compression

Source code được viết bằng ngôn ngữ Python, thực hiện trên môi trường Jupyter Notebook. Trong Source code có sử dụng các thư viện bên ngoài như:

- Numpy: thư viện dùng để tính toán trong thuật toán
- Matplotlib: thư viện dùng để hiển thị ảnh và so sánh hình ảnh gốc và ảnh đã nén màu
- PIL: thư viện dùng để mở ảnh và lưu ảnh

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

Import các thư viện cần thiết

2. Ý tưởng thực hiện

Ta chia giải thuật thành các hàm để dễ triển khai đúng theo các bước thuật toán và dễ sửa lỗi.

Các hàm:

- *read_img*

```
def read_img(img_path: str) -> Image.Image:
    '''
    Read image from img_path

    Parameters
    ----------
    img_path : str
        Path of image

    Returns
    -------
        Image (2D)
    '''
    # YOUR CODE HERE
    try:
        return Image.open(img_path)
    except FileNotFoundError as e:
        print(f"File not found: {e.filename}")
        return None
```

- Tham số đầu vào:

  + 'img_path': đường dẫn file đầu vào (str)

- Kết quả trả về: data của ảnh vừa đọc có kiểu dữ liệu Image.Image trong PIL
- Mô tả:

  Hàm đọc ảnh từ đường dẫn file ảnh bên ngoài. Handle error nếu như đường dẫn file có lỗi

- *show_img*

```
def show_img(img_2d: np.ndarray) -> None:
    '''
    Show image

    Parameters
    ----------
    img_2d : <your type>
        Image (2D)
    '''
    # YOUR CODE HERE
```

```
    plt.title("Compressed image")
    plt.imshow(img_2d) # choose img_2d to show
    plt.axis('off') # clear axis of chart to completely display
image
    plt.show() # show image
```

- Tham số đầu vào:

  + 'img_2d': ảnh 2 chiều (np.ndarray)
- Kết quả trả về: không có
- Mô tả:

  Hàm hiển thị hình ảnh, có tiêu đề ảnh, không dùng các cột đồ thị trong ảnh.

- *save_img*

```
def save_img(img_2d: np.ndarray, img_path: str) -> None:
    '''
    Save image to img_path

    Parameters
    ----------
    img_2d : <your type>
        Image (2D)
    img_path : str
        Path of image
    '''
    # YOUR CODE HERE
    image = Image.fromarray(img_2d)
    image.save(img_path)
```

- Tham số đầu vào:

  + 'img_2d': ảnh 2 chiều (np.ndarray)

  + 'img_path': đường dẫn file ảnh muốn lưu (str)
- Kết quả trả về: không có
- Mô tả:

  + Hàm lưu ảnh theo đường dẫn file đưa vào xuống máy tính.

  + Hàm *Image.fromarray(img_2d)* dùng để chuyển đổi từ kiểu dữ liệu np.ndarray của Numpy sang kiểu dữ liệu Image.Image của PIL

- *convert_img_to_1d*

```
def convert_img_to_1d(img_2d: np.ndarray) -> np.ndarray:
    '''
    Convert 2D image to 1D image
```

```
    Parameters
    ----------
    img_2d : <your type>
        Image (2D)

    Returns
    -------
        Image (1D)
    '''

    # YOUR CODE HERE
    # 3: number of columns represent for color channels (this case:
rgb), -1: special number in numpy to automatically get number that
compatible to size of matrix, in this case is the row
    # before: shape = (row=x, col=y, channel=3), after: shape =
(x*y, 3)
    return img_2d.reshape(-1, 3)
```

- Tham số đầu vào:
  + 'img_2d': ảnh 2 chiều (np.ndarray)
- Kết quả trả về: ảnh 1 chiều (np.ndarray)
- Mô tả:
  + Hàm chuyển đổi từ ảnh 2 chiều sang ảnh 1 chiều
  + *reshape(-1, 3)* hàm chuyển đổi chiều của Numpy, trong đó:
    - **3**: nằm bên phải là số cột của ma trận, ở đây nghĩa là mỗi pixel có 3 kênh màu là red, green, blue. Mỗi kênh màu có giá trị từ 0-255 và cùng nhau tạo ra hàng triệu màu sắc. Mỗi pixel có 1 màu trong số đó.
    - Ngoài ra với Numpy, **-1**: nằm bên trái chỉ số dòng của ma trận nhưng ở đây Numpy sẽ tự tìm số dòng phù hợp với kích thước ma trận. Ví dụ: ma trận ban đầu kích thước (r, c, 3), ở đây ta đã định sẵn số cột là 3 và mong muốn Numpy tự tìm số dòng phù hợp nên số dòng sẽ là r * c thì phù hợp với kích thước. Do đó sẽ *reshape* từ (r, c, 3): 2 chiều –> (r * c, 3): 1 chiều
    - Thông thường (r * c, 3), ta nhìn sẽ là 2 chiều vì có dòng và cột nhưng ở đây ta sẽ hình dùng mỗi phần tử trong mảng sẽ là một pixel, một pixel thì luôn có 3 màu nên ta sẽ hình dùng rằng đây chỉ là mảng 1 chiều chứa các pixel là được. vd: [pixel1, pixel2, …]

- kmeans

```python
def kmeans(img_1d: np.ndarray, k_clusters: int, max_iter: int,
init_centroids='random'):
    '''
    K-Means algorithm

    Parameters
    ----------
    img_1d : np.ndarray with shape=(height * width, num_channels)
        Original (1D) image
    k_clusters : int
        Number of clusters
    max_iter : int
        Max iterator
    init_centroids : str, default='random'
        The method used to initialize the centroids for K-means
clustering
        'random' --> Centroids are initialized with random values
between 0 and 255 for each channel
        'in_pixels' --> A random pixel from the original image is
selected as a centroid for each cluster

    Returns
    -------
    centroids : np.ndarray with shape=(k_clusters, num_channels)
        Stores the color centroids for each cluster
    labels : np.ndarray with shape=(height * width, )
        Stores the cluster label for each pixel in the image
    '''

    # YOUR CODE HERE
    centroids = initalize_centroids(img_1d, k_clusters,
init_centroids)
    for _ in range(max_iter):
        labels = assign_cluster_labels(img_1d, centroids)
        new_centroids = update_centroids(img_1d, labels, k_clusters)
        # check after down round, if two array is equal
        if np.array_equal(new_centroids.astype(np.uint8),
centroids.astype(np.uint8)):
            break
        centroids = new_centroids

    return centroids, labels
```

- Tham số đầu vào:

  + 'img_1d': ảnh 1 chiều (np.ndarray)

  + 'k_clusters': số lượng cluster (màu sắc) còn lại của ảnh sau khi nén xuống (int)

  + 'max_iter': số vòng lặp tối đa để thực hiện thuật toán Kmeans cho tới khi tìm được các màu thích hợp cho mỗi pixel (int)

  + 'init_centroids' (str): "random" hoặc "in_pixels"

    - "random": các centroids được random màu sắc với mỗi kênh màu từ 0 – 255.
    - "in_pixels": random các centroids dựa vào các pixels bất kì có sẵn trong ảnh.

- Kết quả trả về: 'centroids', 'labels'.

  - 'centroids': các centroids sau khi đã chạy thuật toán xong (np.ndarray)
  - 'labels': các nhãn màu cho từng pixel trong ảnh tương ứng với các centroids (np.ndarray)

- Mô tả:

  + *initalize_centroids(img_1d, k_clusters, init_centroids)*: hàm khởi tạo các centroids trước khi vào vòng lặp lưu trong biến 'centroids'

  + *assign_cluster_labels(img_1d, centroids)*: hàm gán mỗi pixel 1 label ảnh phù hợp sau mỗi vòng lặp lưu trong biến 'labels'

  + *update_centroids(img_1d, labels, k_clusters)*: hàm update các centroids sau khi đã xác định label cho các pixel.

  + if np.array_equal(new_centroids.astype(np.uint8), centroids.astype(np.uint8)):
    break

    điều kiện dừng của vòng lặp, ở đây ta sẽ so sánh mảng 'new_centroids' sau khi casting sang kiểu dữ liệu uint8 và mảng 'centroids' sau khi casting sang kiểu dữ liệu uint8, nếu 2 mỗi phần tử của 2 mảng là bằng nhau thì thỏa điều kiện và vòng lặp kết thúc. Có nghĩa là các pixels đã được gán các labels tương ứng và không còn thay đổi gì nữa. Nếu không thỏa điều kiện thì gán *centroids = new_centroids*.

- *generate_2d_img*

```
def generate_2d_img(img_2d_shape: tuple, centroids: np.ndarray,
labels: np.ndarray) -> np.ndarray:
    '''
    Generate a 2D image based on K-means cluster centroids

    Parameters
    ----------
    img_2d_shape : tuple (height, width, 3)
        Shape of image
    centroids : np.ndarray with shape=(k_clusters, num_channels)
        Store color centroids
```

```
    labels : np.ndarray with shape=(height * width, )
        Store label for pixels (cluster's index on which the pixel
belongs)

    Returns
    -------
        New image (2D)
    '''

    # YOUR CODE HERE
    # assign new cluster label color for each pixels and casting
them to uint8 [0-255]
    compressed_image = centroids[labels].astype(np.uint8)
    return np.reshape(compressed_image, img_2d_shape)
```

- Tham số đầu vào:
  + 'img_2d_shape': shape ban đầu của hình ảnh (height, width, 3) (tuple)
  + 'centroids': các centroids sau khi đã thực hiện K-Means xong (np.ndarray)
  + 'labels': mảng các label ứng với mỗi pixel trong ảnh
- Kết quả trả về: data của ảnh 2 chiều sau khi nén xong
- Mô tả:
  compressed_image = centroids[labels].astype(np.uint8)
  Với mỗi giá trị label trong labels tương ứng với mỗi pixels trong ảnh, ta sẽ lấy màu tương ứng đó trong centroids rồi lại casting kiểu dữ liệu từ float64 sang số nguyên không âm uint8 vì mỗi kênh màu phải mang giá trị nguyên không âm. Sau cùng gộp hết các phần tử pixels đó tạo ra một mảng có số lượng phần tử và shape giống như ảnh 1 chiều img_1d nhưng lại mang các màu sắc ứng với mỗi cluster nằm gần nhất với nó, gán ngược lại vào biến 'compressed_image'.

- *initalize_centroids*

```
def initalize_centroids(img_1d: np.ndarray, k_clusters: int,
init_centroids: str) -> np.ndarray:
    if init_centroids == 'random':
        return np.random.choice(256, (k_clusters, 3),
replace=False).astype(np.uint8)
    elif init_centroids == 'in_pixels':
        img_1d_unique = np.unique(img_1d, axis=0)
        return
img_1d_unique[np.random.choice(img_1d_unique.shape[0], k_clusters,
replace=False)]
```

- Tham số đầu vào:

  + 'img_1d': ảnh 1 chiều (np.ndarray)

  + 'k_clusters': số lượng cluster (màu sắc) còn lại của ảnh sau khi nén xuống (int)

  + 'init_centroids' (str): "random" hoặc "in_pixels"

- Mô tả:

  + Nếu *init_centroids == 'random'*:

  Trả về mảng các phần tử với shape (k_clusters, 3) với các kênh màu bất kì từ 0 – 255.

  + Nếu *init_centroids == 'in_pixels':*

  np.unique(img_1d, axis=0): hàm chuyển đổi mảng các phần tử thành mảng các phần tử không được trùng nhau (giống như tập set trong Python), axis = 0: chuyển đổi với chiều thứ nhất mà trong img_1d có số chiều thực tế là 2 nên chiều thứ nhất sẽ là các dòng. Vậy nên dòng nào trùng nhau thì bỏ, chỉ lấy những dòng không trùng nhau.

  Trả về mảng các phần tử với shape (k_clusters, 3) với mỗi phần tử là 1 phần tử là 1 phần tử bất kì trong mảng unique vừa tạo xong để các centroids không được phép trùng nhau.

- *assign_cluster_labels*

```
def assign_cluster_labels(img_1d: np.ndarray, centroids: np.ndarray)
-> np.ndarray:
    # assign the closest cluster label to each pixel in img_1d by
calculate all distances of each pixel with each centroids by
euclidean distance
    # and return the index of closest centroids for each pixels
    distances = ((img_1d[:, np.newaxis, :] - centroids[np.newaxis,
:, :])**2).sum(axis=2)
    return np.argmin(distances, axis=1)
```

- Tham số đầu vào:

  + 'img_1d': ảnh 1 chiều (np.ndarray)

  + 'centroids': các centroids tại 1 thời điểm trong vòng lặp (np.ndarrary)

- Kết quả trả về: những labels ứng với mỗi pixels trong ảnh tại một thời điểm trong vòng lặp (np.ndarray)

- Mô tả:

  + *distances = ((img_1d[:, np.newaxis, :] - centroids[np.newaxis, :, :])**2).sum(axis=2)*

  Ban đầu img_1d có shape (n, 3), centroids có shape (k, 3). Ta mong muốn tạo ra một matrix với shape (n, k) mang ý nghĩa: với mỗi pixel trong image sẽ có k distance ứng với k centroids. Sau đó sẽ dùng *np.argmin(distances, axis=1)* để lấy index của giá trị nhỏ nhất trong mảng các khoảng cách từ pixels tới mỗi centroids. Do đó sẽ tạo ra một mảng có shape (n, 1) với n pixels có 1 index mang label tới centroids gần nó nhất.

img_1d[:, np.newaxis, :], hoặc *centroids[np.newaxis, :, :]* đều là cách reshape trong numpy, mỗi dấu ':' tượng trưng cho 1 chiều trong biến dữ liệu, vd: ban đầu img_1d có shape (n, 3) tức là 2 chiều, ta muốn tạo ra img_1d có shape (n, 1, 3) có 3 chiều thì ta reshape theo kiểu Numpy *img_1d[:, np.newaxis, :], np.newaxis:* là attribute giúp tạo 1 chiều mới mang số 1 trong Numpy.

Lý giải tại sao lại phải tạo số chiều như vậy thì trong Numpy có cơ chế broadcasting là một cơ chế giúp các phương thức tính toán trên ma trận trở nên nhanh chóng hơn. Tuy nhiên để tính toán theo broadcasting ta cần phải kiểm soát số chiều của mỗi ma trận cần tính toán, nếu không sẽ không tính được và báo lỗi. Ví dụ cho trường hợp hợp lệ: ma trận với shape (n, 1, 3) với ma trận với shape (1, k, 3). Với broadcasting, số 1 sẽ luôn đi chung với mọi số được tức là n – 1, 1 – k, … Và 2 số giống nhau luôn đi chung với nhau được vd: 3 – 3.

Trường hợp không được là: n – 2, 3 – 4, … Không thỏa mãn các quy định trên. Ban đầu với shape (n, 3) và shape (k, 3) ta cũng không thể tính toán broadcasting vì không thỏa mãn, nên ta phải biến đổi chiều của nó. Ngoài ra với việc biến đổi chiều như vậy ta có thể tạo ra các ma trận theo ý mà ta mong muốn từ những ma trận ban đầu và còn tính toán nhanh hơn cũng như là ngắn gọn hơn trong Python.

Trong code trên, ta cần tính toán khoảng cách của các pixel tới mỗi centroids nên ta sẽ trừ 2 ma trận đã reshape rồi bình phương. Sau khi xong thì shape của ma trận vẫn là (n, k, 3), sau đó ta cộng lại với *sum*(*axis = 2)* giúp shape thu lại còn (n, k) vì *axis = 2* chiều thứ 3 nên khi cộng các cột chiều thứ 3 sẽ thu về còn 1 giá trị thì sẽ bỏ luôn chiều của 3 giá trị trước đó. Ta không dùng căn vì nếu có tính căn cũng không quan trọng bởi ta chỉ lấy giá trị nhỏ nhất trong ma trận đó mà thôi, và ta không dùng lại distances này sau khi đã tìm giá trị nhỏ nhất xong nên nếu tính căn sẽ làm chậm đi thuật toán.

- *update_centroids*

```
def update_centroids(img_1d: np.ndarray, labels: np.ndarray,
k_clusters: int) -> np.ndarray:
    # update centroids by calculating mean of color channel of all
pixels that have same label
    return np.array([img_1d[labels == i].mean(axis=0) if
np.any(labels == i) else np.random.choice(256, 3, replace=False) for
i in range(k_clusters)])
```

- Tham số đầu vào:
  + 'img_1d': ảnh 1 chiều (np.ndarray)
  + 'labels': mảng các label ứng với mỗi pixel trong ảnh
  + 'k_clusters': số lượng cluster (màu sắc) còn lại của ảnh sau khi nén xuống (int)
- Kết quả trả về: các centroids mới (np.ndarray)

- Mô tả:

  + Với mỗi giá trị i trong ‘k_clusters’ sẽ tương ứng với mỗi nhãn trong labels, *img_1d[labels == i]* có nghĩa là lấy giá trị của các phần tử pixels trong ‘img_1d’ sao cho giá trị nhãn của nó ứng với i thì sẽ tạo ra mảng 2 chiều (x, 3) chứa các giá trị của kênh màu ban đầu sau đó *.mean(axis=0)*, *axis = 0*: nghĩa là ta sẽ lấy mỗi kênh màu của mỗi pixels sau đó trung bình cộng lại cuối cùng sẽ ra 1 màu với 3 kênh màu, tức mảng 1 chiều. Tuy nhiên trước khi vào vòng lặp đó, ta check *if np.any(labels == i)* có nhãn nào có giá trị i không, nếu không thì ta sẽ random ra 1 centroid bất kì để không tính mean vì nếu tính mean mà không có giá trị thì sẽ chia cho 0 mà chia cho 0 là bị lỗi. Cuối cùng với các mảng 1 chiều centroids có được thì ta gom lại trong hàm *np.array()* để ghép lại thành mảng 2 chiều chứa các centroids.

- *compressed_image_color*

```
def compressed_image_color(original_image: np.ndarray) ->
np.ndarray:
    original_shape = original_image.shape
    img_1d = convert_img_to_1d(original_image)
    # compress image
    centroids, labels = kmeans(img_1d, k_clusters=7, max_iter=100,
init_centroids='random')
    return generate_2d_img(original_shape, centroids, labels)
```

- Tham số đầu vào:

  + ‘original_image’: ảnh 2 chiều ban đầu (np.ndarray)

- Kết quả trả về: ảnh 2 chiều đã nén (np.ndarray)

- Mô tả:

  + Đầu tiên ta lấy shape ban đầu của ảnh ban đầu để về sau reshape lại cho ảnh đã nén. Sau đó convert ảnh 2 chiều ban đầu sang ảnh 1 chiều rồi chạy qua thuật toán kmeans để lấy centroids, labels. Sau trả về hàm *generate_2d_img()* để trả về ảnh 2 chiều đã nén. Trong ví dụ trên ta thấy *k_clusters=7, max_iter=100, init_centroids='random'* là các tham số có thể thay đổi giá trị tùy ý khi đưa vào hàm

- *main()*

```
def main() -> None:
    img_path = input("Input file path or file name of image, must
have extension of image file") # image path to input
    out_img_path = input("type file path or file name of image to
save, must have extension of image file") # image path to save

    image = read_img(img_path)
    if image is None:
```

```
        print("Please provide a valid image path.")
    else:
        original_image = np.array(image, dtype=np.uint8)

        compressed_image = compressed_image_color(original_image)

        show_img(compressed_image) # show compressed image
        try:
            save_img(compressed_image, out_img_path) # save file
        except ValueError:
            print("Unknown file extension when save image")
```

- Mô tả:

  + Hàm main, chạy chương trình chính. Đầu tiên sẽ nhận địa chỉ file image gốc để nén ảnh, và nhận địa chỉ file image cần lưu vào máy tính. Sau đó đọc ảnh, nếu *if image is None,* có nghĩa là đường dẫn file đó bị lỗi và sẽ có error handling kết thúc chương trình. Nếu không lỗi thì sẽ bắt đầu chuẩn bị nén ảnh. *original_image = np.array(image, dtype=np.uint8)* ta chuyển đổi ảnh từ kiểu dữ liệu Image.Image của PIL sang np.ndarray của Numpy với kiểu dữ liệu phần tử bên trong là uint8. Sau đó bắt đầu nén ảnh, *show_img(compressed_image),* hàm hiển thị ảnh sau khi nén. Cuối cùng là lưu ảnh, *save_img(compressed_image, out_img_path),* nếu đường dẫn file không có tên đuôi (extension) thì sẽ không lưu được ảnh và handle exception. Các tên đuôi có thể là (jpeg, jpg, png, pdf, …) tùy ý.

3. Kết quả

*ảnh gốc (1280*853)*

ảnh nén với k = 3

1ảnh nén với k = 5

ảnh nén với k = 7

4. Nhận xét:

- Ở đây ta thấy với ảnh gốc, số lượng màu đầy đủ, ảnh cho ra rất đẹp và nịnh mắt.
- Với ảnh có k = 3, tức 3 màu, thì ảnh chỉ có những màu chủ đạo nhất trong ảnh gốc ví dụ màu (đen, xanh nhạt, xanh đậm), tuy nhiên chỉ với 3 màu vẫn đủ để hiển thị đủ nội dung của bức ảnh gốc tức là một con lizard và vẫn thấy đầy đủ mắt mũi miệng của nó nhưng lại không được rõ rệt và không phân biệt các chi tiết với nhau.
- Với ảnh có k = 5, dường như mọi chi tiết được rõ ràng hơn, con mắt được chi tiết và sắc sảo hơn, ta có thể thấy được phần tai của con lizard, những vùng màu được tách biệt rõ ràng hơn.
- Với ảnh có k = 7, những màu chiếm nhiều nhất trong bức ảnh lại tiếp tục được thể hiện và ta thấy lizard trở nên chi tiết và sinh động hơn, tuy nhiên phần màu vàng nâu vẫn quá chung chung mà chưa thể phân cụm cũng như tách biệt để biết rõ màu sắc trên các vùng bộ phận trên lizard.
- Với *init_centroids='random' hay 'in_pixels'* thì tốc độ ảnh cho ra có khác biệt nhau, tuy nhiên tốc độ cho ra ảnh quan trọng nhất là ở thuật toán có xác định được những centroids nào quan trọng và chiếm nhiều nhất trong bức ảnh không. Vì thuật toán phụ thuộc rất nhiều vào việc chạy đi chạy lại nhiều lần để lấy được các labels và centroids cuối cùng hợp lý nhất nên nếu đầu vào là những centroids tốt sẵn thì thuật toán chạy sẽ nhanh hơn khoảng 50% - 70%. Các thuật toán hỗ trợ giúp tìm các centroids tốt từ ban đầu ví dụ như: k-means++, …
- Ngoài ra tốc độ thuật toán còn phụ thuộc vào mức độ vector hóa tính toán trong Numpy. Căn bản là trong Python, mọi thứ đều rất chậm nên nếu biết tận dụng, giảm được chỗ nào không cần thiết, điều kiện kết thúc sớm hoặc broadcasting trong Numpy tốt thì thuật toán sẽ tối ưu hơn.
- Ngoài ra, việc chọn số lượng centroids cũng rất quan trọng trong nén ảnh để đạt được chất lượng ảnh tốt nhất mà số lượng màu vẫn thấp nhất. Có thể kể đến một số thuật toán tìm ra số lượng centroids phù hợp như là: Elbow method (clustering), Silhouette (clustering), …
- Với các ảnh Lizard như trên thì tốc độ thuật toán rất nhanh tầm 5 – 10s với các k = 3, 5, 7.

Tuy nhiên thử với ảnh có resolution lớn hơn thì tốc độ nén ảnh sẽ hoàn toàn khác.

*ảnh gốc (2705 * 3381)*

ảnh nén với k = 3

ảnh nén với k = 5

ảnh nén với k = 7

Ta thấy với ảnh có độ phân giải (resolution) cao hơn thì tốc độ nén ảnh sẽ lâu hơn từ 20s – 1 phút Vì có kích thước lớn hơn nên ma trận cũng lớn hơn do đó các phép toán trên ma trận cũng lâu hơn đáng kể.

References

Tài liệu tham khảo

[1] "k-means clustering," [Online]. Available: https://en.wikipedia.org/wiki/K-means_clustering. [Accessed 18 6 2024].

[2] V. H. Tiệp, "Bài 4: K-means Clustering," [Online]. Available: https://machinelearningcoban.com/2017/01/01/kmeans/. [Accessed 18 6 2024].

[3] P. T. P. Uyên, Ma trận trong Python - Đồ án 1.

[4] H. t. t. OLM, Director, *Giải thuật phân cụm K-means.* [Film].

[5] Numpy, "NumPy reference," [Online]. Available: https://numpy.org/doc/stable/reference/. [Accessed 18 6 2024].

[6] "PILLOW Reference," [Online]. Available: https://pillow.readthedocs.io/en/stable/reference/. [Accessed 18 6 2024].

[7] "How can the Euclidean distance be calculated with NumPy?," Stackoverflow, [Online]. Available: https://stackoverflow.com/questions/1401712/how-can-the-euclidean-distance-be-calculated-with-numpy. [Accessed 18 6 2024].

[8] "Numpy Axes, Explained," [Online]. Available: https://www.sharpsightlabs.com/blog/numpy-axes-explained/#:~:text=NumPy%20axes%20are%20the%20directions,along%20the%20rows%20and%20columns.. [Accessed 18 6 2024].

[9] "What does -1 mean in numpy reshape?," Stackoverflow, [Online]. Available: https://stackoverflow.com/questions/18691084/what-does-1-mean-in-numpy-reshape. [Accessed 18 6 2024].

[10] "Trying to understand what is happening in this Python Function," Stackoverflow, [Online]. Available: https://stackoverflow.com/questions/47544099/trying-to-understand-what-is-happening-in-this-python-function. [Accessed 18 6 2024].

[11] "Using Matplotlib," [Online]. Available: https://matplotlib.org/stable/users/index.html. [Accessed 18 6 2024].

Hình ảnh

https://www.pexels.com/photo/work-coffee-25935100/

https://pixabay.com/photos/lizard-iguana-fijian-iguana-8787888/