ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC
TOÁN ỨNG DỤNG VÀ THỐNG KÊ

Lab 01: Color Compression

Sinh viên thực hiện

Họ và tên	:	Võ Nguyễn Hoàng Kim
Mã số sinh viên	:	21127090
Lớp	:	21CLC07

Phụ lục

I. Thông tin sinh viên

Họ tên: Võ Nguyễn Hoàng Kim

Mã số sinh viên: 21127090

Lớp: 21CLC07

II. Ý tưởng thực hiện

Bài toán Color Compression được thực hiện bằng cách sử dụng thuật toán Kmeans (được xây dựng lại theo yêu cầu đề bài).

Bước 1: Lựa chọn k điểm bất kỳ để làm trung tâm (centroid)

Bước 2: Duyệt các điểm màu trên ảnh, xem xét vị trí của centroid nào gần nó nhất, phân điểm màu đó vào cụm màu tương ứng

Bước 3: Nếu việc gán điểm màu ở bước trên không có sự thay đổi so với lần gán trước đó, ta sẽ dừng thuật toán

Bước 4: Sau khi tất cả điểm màu được phân vào các cụm tương ứng, ta bắt đầu tính toán lại vị trí centroids của từng cụm

Bước 5: Lặp lại bước 2 đến khi hoàn thành

Thuật toán có thể rơi vào trạng thái lặp vô tận khi , do đó, ta chỉ thực hiện thuật toán trong số lần lặp hữu hạn (max_iter). Sau khi kết thúc số lần lặp, hàm sẽ trả về hình ảnh đã được xử lý.

III. Mô tả chi tiết các hàm

1. initCentroids

Hàm được sử dụng để nhận dạng phương thức khởi tạo các centroids ban đầu.

Với phương thức “random”: centroids sẽ được khởi tạo là các điểm màu được lựa chọn ngẫu nhiên trong khoảng [0,255]. Các centroids này sẽ có giá trị không trùng nhau

Với phương thức “in_pixels”: centroids sẽ được khởi tạo bằng các màu có trong hình gốc. Với trường hợp số cụm (k_cluster) nhập vào lớn hơn số lượng màu mà hình có (uniqPixels), chúng ta sẽ gán giá trị cho k_cluster bằng với uniqPixels. Từ đó chọn ra các số centroids ngẫu nhiên có trong ảnh.

```
def initCentroids(img_1d, k_clusters, init_centroids):
    if(init_centroids == 'random'):
        centroids = [np.random.choice(256, size = (img_1d.shape[1]), replace = False)
                     for _ in range(k_clusters)]
    elif(init_centroids == 'in_pixels'):
        uniqPixels = np.unique(img_1d, axis=0)
        if(len(uniqPixels) < k_clusters):
            k_clusters = len(uniqPixels)
```

```
        index = np.random.choice(len(uniqPixels), size = k_clusters,
                                 replace = False)
        centroids = [img_1d[i] for i in index]
    return centroids
```

2. labelPixels

Hàm được sử dụng để gán nhãn cho các điểm màu trong hình.

Để làm được điều này, trước hết, ta sẽ tính khoảnng cách của điểm màu thứ i của hình với các centroids. Điểm màu thứ i sẽ được phân vào cụm của centroids có khoảng cách gần nó nhất.

Đồng thời tại cụm màu đó, ta sẽ thêm điểm màu vừa xét vào.

```
def labelPixels(img_1d, k_clusters, centroids, labelArr, clusterList):
    for i in range(len(img_1d)):
        distance = np.linalg.norm(centroids - img_1d[i], axis=1)
        labelArr[i] = distance.argmin()
        clusterList[labelArr[i]].append(i)
    return labelArr, clusterList
```

3. updateCentroids

Hàm được thực hiện dựa trên phép tính trung tính trung bình cộng của các vị trí điểm màu có trong danh sách cụm (Sau khi tất cả điểm màu của hình đã được phân cụm). Kết quả thu được sẽ được lấy làm giá trị mới cho centroids của cụm (cluster) tương ứng.

```
def updateCentroids(img_1d, centroids, k_clusters, clusterList):
    for i in range(k_clusters):
        if(clusterList[i]):
            centroids[i] = np.mean([img_1d[j] for j in clusterList[i]], axis=0)
    return centroids
```

4. kmeans

Các tham số trong hàm gồm:

- **img_1d:** mảng lưu trữ hình ảnh cần xử lý.
- **k_cluster:** số cụm màu.
- **max_iter:** số lần lặp tối đa.
- **init_centroids:** phương thức để khởi tạo centroids ban đầu.

Như thuật toán gốc, trong hàm này, ta sẽ triển khai lần lượt từng bước:

- Đầu tiên, ta gọi hàm initCentroids để khởi tạo các centroids ban đầu dựa vào phương thức là random hay in_pixels.
- Ta khởi tạo một list chứa các nhãn (labelArr).
- Bước vào vòng lặp while với số lần lặp tối đa là max_iter.
- Sử dụng biến “preLabels” để đánh dấu lại kết quả của vòng lặp trước đó.

- Khởi tạo một clusterList là danh sách các cụm màu.
- Ta gọi hàm labelPixels để gán nhãn cho từng điểm màu.
- Sau khi thực hiện gán nhãn cho tất cả điểm màu có trong ảnh, ta sẽ kiểm tra danh sách nhãn lúc này có sự khác biệt gì với danh sách ở bước trước (tức so sánh labelArr và preLabels). Nếu như không có sự khác biệt, tức là các điểm màu đã không còn phân cụm được nữa, ta sẽ thoát khỏi vòng lặp. Nhưng nếu có sự khác biệt, ta đến bước kế tiếp.
- Ta cập nhật lại giá trị centroids của các cụm màu.

Kết thúc vòng lặp, hàm sẽ trả về một mảng các centroids (centroids) và label (labelArr).

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids='random'):
    centroids = initCentroids(img_1d, k_clusters, init_centroids)
    labelArr = [None for _ in range(len(img_1d))]

    while(max_iter):
        preLabels = labelArr.copy()
        max_iter -=1
        clusterList = [[] for _ in range(k_clusters)]

        # gán nhãn cho các điểm màu trong ảnh
        labelArr, clusterList = labelPixels(img_1d, k_clusters, centroids, labelArr,
                                            clusterList)

        #  nếu không có sự thay đổi giữa các điểm ảnh
        if(labelArr == preLabels):
            break

        # tính lại giá trị trung tâm của các cụm
        centroids = updateCentroids(img_1d, centroids, k_clusters, clusterList)

    return np.array(centroids).astype(int), np.array(labelArr)
```

5. colorCompression

Các tham số truyền vào:

- **imgName:** Tên file ảnh cần được xử lý.
- **k_cluster:** Số cụm màu (cluster).
- **max_iter:** Số lần lặp tối đa.
- **init_centroids:** Cách thức để khởi tạo centroids.
- **extension:** Định dạng đầu ra cho ảnh (sau khi được xử lý).

Trong hàm này, ta sẽ xử lý các việc

- Mở ảnh (sử dụng tên ảnh), chuyển đổi sang kiểu “RGB” và đưa giá trị ảnh về thành mảng 1 chiều (làm phẳng).
- Chạy thuật toán Kmeans để thu về mảng các centroids và labels.
- Sau khi chạy thuật toán Kmeans, ta cập nhật lại từng điểm ảnh tương ứng với nhóm mà nó thuộc về.

- Trả về nội dung ảnh (sau khi đã thực hiện gom nhóm).
- Thực hiện lưu ảnh dưới dạng: **<tên ảnh>_k<số lượng cụm (k_clusters)>_<phương thức cho centroids (init_centroids)>.<định dạng đầu ra (extension)>**.
  - Ví dụ: tên ảnh là test1, k_clusters = 3, init_centroids = 'random', extension = 'png' → ảnh được lưu sẽ là **test1_k3_random.png**

```
def colorCompression(imgName, k_cluster, max_iter, init_centroids, extension):
    # Xử lý ảnh ban đầu
    img = Image.open(imgName, mode="r")
    img = img.convert('RGB')
    img = np.array(img)
    img_1d = img.reshape(img.shape[0]*img.shape[1], img.shape[2])

    # chạy thuật toán Kmeans
    centroids, labels = kmeans(img_1d, k_cluster, max_iter, init_centroids)

    # gán lại các điểm màu tương ứng sau khi chạy kmeans
    for i in range(len(img_1d)):
        img_1d[i] = centroids[labels[i]]
    img_1d = img_1d.reshape(img.shape).astype(np.uint8)

    fileName = f"{imgName.split('.')[0]}_k{k_cluster}_{init_centroids}.{extension}"
    Image.fromarray(img_1d, 'RGB').save(fileName)
```

6. main

Hàm main cho phép người dùng nhập tên ảnh. Do ảnh được nhận vào bằng tên ảnh, do đó, ảnh phải được nằm cùng thư mục chứa source code. Nếu không, chương trình sẽ báo lỗi và yêu cầu nhập lại.

Tiếp đến là lựa chọn của người dùng về định dạng đầu ra là png hay pdf. Nếu như chọn định dạng đầu ra khác với 2 loại trên, chương trình sẽ báo lỗi và yêu cầu nhập lại.

Sau khi đã nhận được tên file và định dạng đầu ra, chương trình sẽ gọi hàm colorCompression, truyền vào các tham số <Tên ảnh>, <Số cluster>, <Số lần lặp tối đa>, <Phương thức khởi tạo centroids>, <Định dạng đầu ra> để bắt đầu xử lý ảnh.

Lưu ý

- <Tên ảnh>, <Định dạng đầu ra>: được nhập từ bàn phím.
- <Số cluster>, <Số lần lặp tối đa>, <Phương thức khởi tạo centroids>: được truyền cố định trong source code. Nếu muốn thay đổi tham số, người dùng cần thay đổi trong source code.
- Với chương trình này, định dạng ảnh đầu vào chỉ có thể là .jpg và .png

```
if __name__ == '__main__':
    # xử lý đầu vào
    while True:
        imgName = input('Name of image: ')
        if(not os.path.isfile(imgName)):
```

```
            print("Error! File not found.")
        else: break
    while True:
        extension = input('Save format (png/pdf): ')
        if(extension not in ['png', 'pdf']):
            print('Error! Invalid save format')
        else: break
    # Gọi hàm để thực hiện gom nhóm

    colorCompression(imgName, 7, 10, 'in_pixels', extension)
```

Ví dụ

Khi chạy chương trình với source code trên, với k_clusters = 7, max_iter = 10, init_centroids = 'in_pixel'

Input:

- Name of image: **test01.jpg**
- Save format (png/pdf): **png**

Output:

- Sau khi kết thúc chương trình, hình ảnh (đã được xử lý) sẽ được lưu trong thư mục (cùng với thư mục chứa source code) với tên: **test01_k7_in_pixels.png**

IV. Kết quả đạt được

Kích thước ban đầu của ảnh: 1000x662 – 148.20KB.

1. Init_centroids = 'random'

Ảnh ban đầu

k = 3

k = 5

k = 7

Số cụm (k_cluster)	Thời gian xử lý	Kích thước
K = 3	75.19 giây	61.95 KB
K = 5	91.03 giây	98.34 KB
K = 7	117.27 giây	116.86 KB

2. Init_centroids = ‘in_pixels’

Ảnh ban đầu

k = 3

k = 5

k = 7

Số cụm (k_cluster)	Thời gian xử lý	Kích thước
K = 3	87.25 giây	63.71 KB
K = 5	93.73 giây	93.20 KB
K = 7	111.92 giây	127.38 KB

V. Nhận xét

Em đã thử thêm 2 trường hợp với k =10, k=15 (với init_centroids = ‘random’)

k = 10

k = 15

Có thể thấy, k=10 hay k=15, kết quả ở hai hình không có gì khác biệt so với nhau. So với hình gốc, ảnh (đã xử lý) có thể giữ được gần như tất cả các màu cũng như các chi tiết, hình được phân màu rõ ràng, tuy nhiên vẫn không thể rõ được như ảnh gốc.

Nhưng khi đến với kết quả của k=3, k=5, k=7, ta có thể thấy, các sắc tố màu đã lệch đi rất nhiều so với hình gốc. Đặc biệt với k=3, các chi tiết trong hình không còn được nhìn thấy rõ ràng nữa, chỉ có thể nhìn được (ở mức tạm) đối với các chi tiết lớn, còn về các chi tiết nhỏ hầu như không thể quan sát được. Điều này dẫn đến nội dung ảnh không còn rõ ràng, không thể chấp nhận được.

Tuy nhiên, việc gom nhóm số lượng màu trong ảnh đã giúp cho kích thước của ảnh được giảm đi đáng kể.

Từ kết quả trên, em rút ra được một số nhận xét sau

Về kích thước: khi tiến hành gom nhóm các màu, lượng màu của một ảnh giảm đi cũng khiến cho kích thước của ảnh giảm. Tuy nhiên việc, với lượng kích thước giảm càng nhiều, ảnh sẽ càng bị sai lệch nhiều hơn so với ảnh gốc. Và đến một khoảng k rất nhỏ, ảnh sẽ chỉ còn thấy được các chi tiết lớn, những chi tiết nhỏ đều không thể nhìn thấy được. Dẫn đến nội dung của anh bị sai lệch, không thể chấp nhận.

Về thời gian xử lý: đối với k càng nhỏ, thời gian xử lý của chương trình càng ít. Điều này cũng đồng nghĩa, khi ta chọn một số k lớn, chương trình sẽ tốn nhiều thời gian để thực hiện xử lý hơn. Đồng thời, thời gian xử lý còn dựa vào độ phức tạp, kích thước của ảnh. Đối với ảnh có kích thước lớn, thời gian để gom nhóm các màu trong ảnh sẽ lâu hơn so với các ảnh đơn giản, các ảnh có kích thước nhỏ.

Tổng kết: Để có giảm kích thước của ảnh nhưng vẫn đảm bảo được nội dung của bức ảnh, ta cần tìm một con số k hợp lý để thực hiện quá trình gom nhóm. Đây là bước quan trọng để có thể vừa giảm đi kích thước của hình ảnh, cũng như có thể bảo đảm rằng nội dung của bức hình được giữ ở mức chấp nhận được.

VI. Nguồn tham khảo

https://machinelearningcoban.com/2017/01/01/kmeans/

https://connect.aisingapore.org/2023/02/image-compression-with-k-means/#:~:text=In%20image%20compression%2C%20we%20use,size%20of%20the%20image%20data

https://www.geeksforgeeks.org/image-compression-using-k-means-clustering/