Trường Đại học Khoa học Tự nhiên ĐHQG-HCM

Khoa Công nghệ Thông tin

BÁO CÁO
Đồ án 1: COLOR COMPRESSION

Môn học

Toán Ứng dụng và thống kê cho CNTT (MTH00057)

Giảng viên bộ môn **Vũ Quốc Hoàng**
Nguyễn Văn Quang Huy
Lê Thanh Tùng
Phan Thị Phương Uyên

Họ và tên **Lê Thị Hoài Thư**
MSSV **21127176**
Lớp **21CLC02**

Ngày 18 tháng 7 năm 2023

1 Tổng quan

Nội dung đồ án

Bài toán đặt ra là giảm số lượng màu để biểu diễn ảnh sao cho nội dung ảnh được bảo toàn nhất có thể:

- Để thực hiện giảm số lượng màu, ta cần tìm ra các đại diện có thể thay thế cho một nhóm màu. Cụ thể ta cần thực hiện gom nhóm các pixel và chọn ra đại diện cho từng nhóm. Như vậy, bài toán trên trở thành gom nhóm các vec-tơ.
- Yêu cầu cài đặt chương trình giảm số lượng màu cho ảnh sử dụng thuật toán K-Means.

Nội dung báo cáo

Báo cáo được viết nhằm giới thiệu và trình bày những nội dung liên quan đến đồ án bao gồm những tiêu chí được giảng viên yêu cầu như:

- Ý tưởng thực hiện, mô tả các hàm.
- Hình ảnh kết quả với từng số lượng màu, ‘k = 3, 5, 7‘.
- Nhận xét về các kết quả trên

Mục lục

2 Ý tưởng thực hiện

- **Bước 1:** Khởi tạo KCentroids bằng 1 trong 2 phương thức sau
  - `"random"`: Khởi tạo centroids ngẫu nhiên với mỗi kênh màu là một giá trị bất kì trong khoảng `[0..255]`.
  - `"in_pixels"`: Khởi tạo centroids là một các điểm ảnh bất kỳ trong ảnh gốc.

  và đảm bảo rằng mỗi centroid trong tập được khởi tạo là duy nhất, không trùng.
- **Bước 2:** Với mỗi điểm ảnh trong ảnh gốc:
  - Tính khoảng cách từ vector màu của điểm đó đến từng vector màu trong tập centroids.
  - Chọn khoảng cách nhỏ nhất từ các khoảng cách vừa tính để lưu lại index của centroid tương ứng (gom nhóm).
- **Bước 3:** Cập nhật lại các centroids là mean của giá trị các vector màu thuộc nhóm của mỗi centroid đó (đã tính ở bước 2).
- **Bước 4:** Kiểm tra đã đạt 1 trong 2 điều kiện dừng sau hay chưa?
  - Thuật toán đã đến giới hạn tối đa lần duyệt (max_iter).
  - Centroids không còn thay đổi giá trị sau bước cập nhật (đã hội tụ).

  Nếu chưa, quay lại **Bước 2** để tiếp tục thuật toán.

3 Tổ chức và mô tả các hàm

• input_from_user()

Input: Không có dữ liệu đầu vào
Output: File ảnh cần giảm số lượng màu `[filename: str]`
Số clusters muốn gom nhóm `[k_clusters: int]`
Số lần duyệt tối đa `[max_iter: int]`
Phương thức khởi tạo centroids `[init_type: str]`
Định dạng ảnh đầu ra `[format: str]`

⇒ **Ý tưởng:** Sử dụng cấu trúc `try...except...` kết hợp với vòng lặp `while True` để ràng buộc dữ liệu đọc từ bàn phím vào (input không được để trống, sai định dạng,...) nhằm giảm thiểu lỗi nhập nhầm ký tự.

• data_from_image(filename)

Input: Tên file ảnh cần đọc/mở `[filename: str]`
Output: Ma trận 1D chứa các điểm ảnh với số lượng kênh màu tùy ý
`[img_1d: np.ndarray, shape=(height×width,num_channels)]`
Chiều cao của ảnh `[len_img: int]`

⇒ **Ý tưởng:** Chương trình dùng hàm `PIL.Image.open()` để lấy dữ liệu từ ảnh sau đó truyền dữ liệu vừa lấy được vào `np.array()` để ma trận ảnh có kiểu np.ndarray. Đặc biệt rằng kiểu dữ liệu của các pixels được lấy ra mặc định là `np.uint8` nên ta phải truyền thêm phép gán `dtype='int'` để thuận tiện cho các thao tác tính toán về sau. (Ví dụ như khi tính khoảng cách, kiểu dữ liệu unsigned sẽ không cho giá trị âm mà sẽ lấy giá trị lớn nhất cộng vào giá trị âm đó, khiến cho tính toán bị sai) Sau đó gọi lệnh `reshape()` với tham số là -1 (Python cho phép để unknown một giá trị, hàm sẽ tự động tính toán trả về cho hợp lý) và số kênh màu của ảnh. Do vậy cần lưu lại chiều cao của ảnh trước khi reshape để sau khi hoàn thành thuật toán kmeans có thể reshape lại kích thước như cũ.

• init_k_centroids(img_1d, k_clusters, init_centroids='random')

Input: Ma trận 1D chứa các điểm ảnh `[img_1d]`
Số lượng clusters muốn khởi tạo `[k_clusters: int]`)
Phương thức khởi tạo centroids `[init_centroids: str]`
Output: Các centroids màu được khởi tạo thành công
`[k_centroids: np.ndarray, shape=(k_clusters,num_channels)]`

⇒ **Ý tưởng:** Với phương thức random, dùng hàm `np.random.choice()` với các tham số truyền vào lần lượt là cận trên 256, shape và tắt replace để tránh random trùng giá trị. Còn với phương thức in_pixels, dùng hàm như trên với tham số là kích thước mảng ảnh 1 chiều, số centroids cần và cũng tắt replace để tránh trùng lặp. Phương thức này cũng sẽ không random trùng do mảng ảnh truyền vào đã được chọn thông qua hàm `np.unique()` ở hàm kmeans trước khi được truyền vào đây.

• labelling(img_1d, k_centroids)

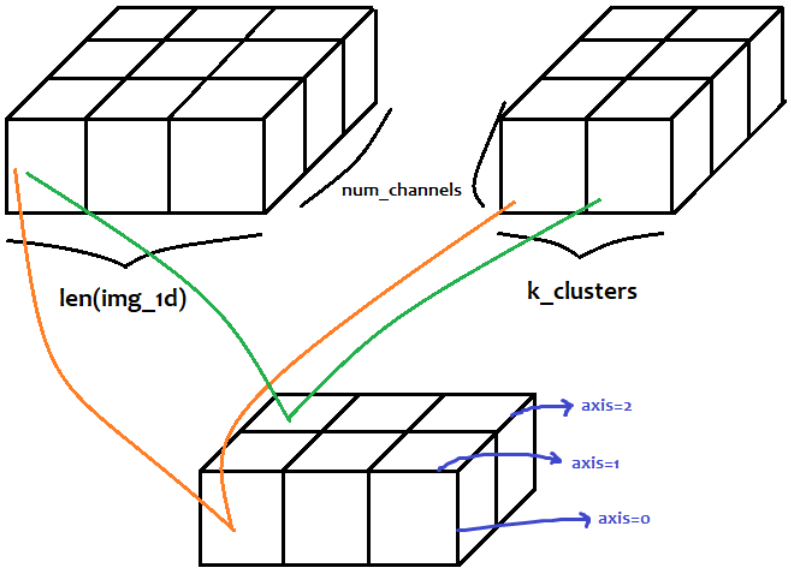
Input: Ma trận 1D chứa các điểm ảnh cần phân nhóm `[img_1d]`
Các centroids màu để phân nhóm `[k_centroids]`
Output: Lưu nhãn (index của centroid màu trong k_centroids) cho từng điểm ảnh
`[labels: np.ndarray, shape=(height×width,)]`

⇒ **Ý tưởng:** Tạo trục mới để với mỗi điểm ảnh và mỗi centroids ta tính hiệu các kênh màu của chúng và ra được kết quả trên chiều không gian mới. Sau đó dùng hàm `np.sum()` để tính tổng bình phương của giá trị hiệu chính là bình phương khoảng cách trên chiều không gian đó, tức trục `axis=-1` thì được ma trận mô phỏng như hình dưới. Dùng hàm `np.argmin()` trên trục `axis=1` để lấy được index của centroid mà điểm ảnh đó thuộc về (bình phương khoảng cách nhỏ nhất) và hoàn thành việc gom nhóm.

• update_k_centroids_and_labels(img_1d, k_centroids)

Input: `[img_1d: np.ndarray, shape=(height×width,num_channels)]`
`[k_centroids: np.ndarray, shape=(k_clusters,num_channels)]`
Output: Cập nhật centroids màu mới
`[new_k_centroids: np.ndarray, shape=(k_clusters,num_channels)]`
Nhãn tương ứng với centroids vừa được cập nhật
`[labels: np.ndarray, shape=(height×width,)]`

⇒ **Ý tưởng:** Dầu tiên lấy ma trận lưu nhãn đã phân lại bằng cách gọi hàm `labelling()` ở trên. Khởi tạo một numpy array `new_k_centroids` dựa theo kích thước của `k_centroids`.



Hình 1: Ví dụ về các thao tác trên hàm Labelling

Hàm sẽ cài đặt cho vòng lặp chạy và cập nhật vector màu của từng centroids mới bằng cách lấy trung bình của các vector màu trong ảnh gốc mà thuộc vào nhóm centroid đó (đã lưu nhãn ở trên):

- `[(labels==index).nonzero()[0]]` trả về ma trận 1 chiều vị trí những điểm ảnh có mang nhãn nhóm là index. Lồng giá trị trên vào ma trận ảnh 1D ta được nhóm các vector màu cùng nhóm với centroid.
- Từ đó cập nhật centroid mới trong vòng lặp bằng `mean()` trên trục `axis=0` nếu như nhóm đó không rỗng (size=0), nếu nhóm rỗng thì chọn random vector màu của một điểm ảnh bất kỳ trong ảnh gốc.

Để lọc trùng, cài đặt 2 vòng lặp lồng nhau: với mỗi centroid, nếu từ vị trí centroid đó đến cuối mảng có bất kì centroid nào khác trùng thì ta đổi nhãn của centroid trùng thành nhãn của centroid hiện tại và thực hiện gán cho centroid trùng vector màu của một điểm ảnh bất kỳ trong ảnh gốc.

• kmeans(`img_1d, k_clusters, max_iter, init_centroids='random'`)

Input: Ma trận 1D chứa các điểm ảnh `[img_1d]`
Số lượng clusters muốn khởi tạo `[k_clusters: int]`)
Số lần duyệt tối đa `[max_iter: int]`)
Phương thức khởi tạo centroids `[init_centroids: str]`

Output: Các centroids màu
`[k_centroids: np.ndarray, shape=(k_clusters,num_channels)]`
Lưu nhãn (index của centroid màu trong k_centroids) cho từng điểm ảnh
`[labels: np.ndarray, shape=(height×width,)]`

⇒ **Ý tưởng:** Đầu tiên gọi hàm `init_k_centroids()` để khởi tạo những centroids đầu tiên theo phương thức khởi tạo tương ứng, lưu ý phải truyền mảng ảnh thông qua hàm `np.unique()` với `axis=0` để lọc trùng theo bộ kênh màu. Sau đó cho chạy vòng for từ `0`→`max_iter`, trong mỗi vòng lặp cập nhật `new_k_centroids` và `labels` từ hàm `update_k_centroids_and_labels()`. So sánh nếu những centroids mới cập nhật không có thay đổi so với những centroids cũ thì trả về Output tương ứng. Ngược lại ta gán `k_centroids = new_k_centroids` để vòng lặp tiếp tục cập nhật và kiểm tra cho đến khi đạt điều kiện dừng.

• to_img(centroids, labels, len_img)

Input: `[centroids: np.ndarray, shape=(k_clusters,num_channels)]`
`[labels: np.ndarray, shape=(height×width,)]`
Chiều cao của ảnh `[len_img: int]`
Output: Dữ liệu ảnh đã được hoàn tất gộp phân loại nhãn

⇒ **Ý tưởng:** Dùng mask `centroids[labels]` để lấy được ma trận các điểm ảnh. Chương trình tiến hành `reshape()` ma trận với tham số là `len_img` ta đã lưu khi trước, giá trị unknown -1 và số kênh màu (`centroids.shape[-1]`) sau đó `astype(np.uint8)` để chuyển ma trận ảnh về kiểu dữ liệu mặc định của nó.

• save_img(img, filename)

Input: Dữ liệu ảnh cần lưu vào file `[img: np.ndarray]`
Tên file ảnh cần lưu `[filename: str]`
Output: Không có dữ liệu đầu ra

⇒ **Ý tưởng:** Dùng hàm `matplotlib.pyplot.imsave()` để lưu ảnh.

• main()

Input: Không có dữ liệu đầu vào
Output: Không có dữ liệu đầu ra

⇒ **Ý tưởng:** Lần lượt gọi thực thi hàm theo thứ tự sau cùng các tham số phù hợp
`input_from_user()` ⟶ `data_from_image()` ⟶ `kmeans()` ⟶ `to_img()` ⟶ `save_img()`

4 Kết quả thử nghiệm

Thử nghiệm giảm số lượng màu với `max_iter = 500`.

Dùng thư viện `timeit` để đo runtime của hàm kmeans, độ chính xác về thời gian ghi trong báo cáo được làm tròn đến chữ số thập phân thứ 5.

Kích thước ảnh làm tròn đến KB.

Hình ảnh testcase

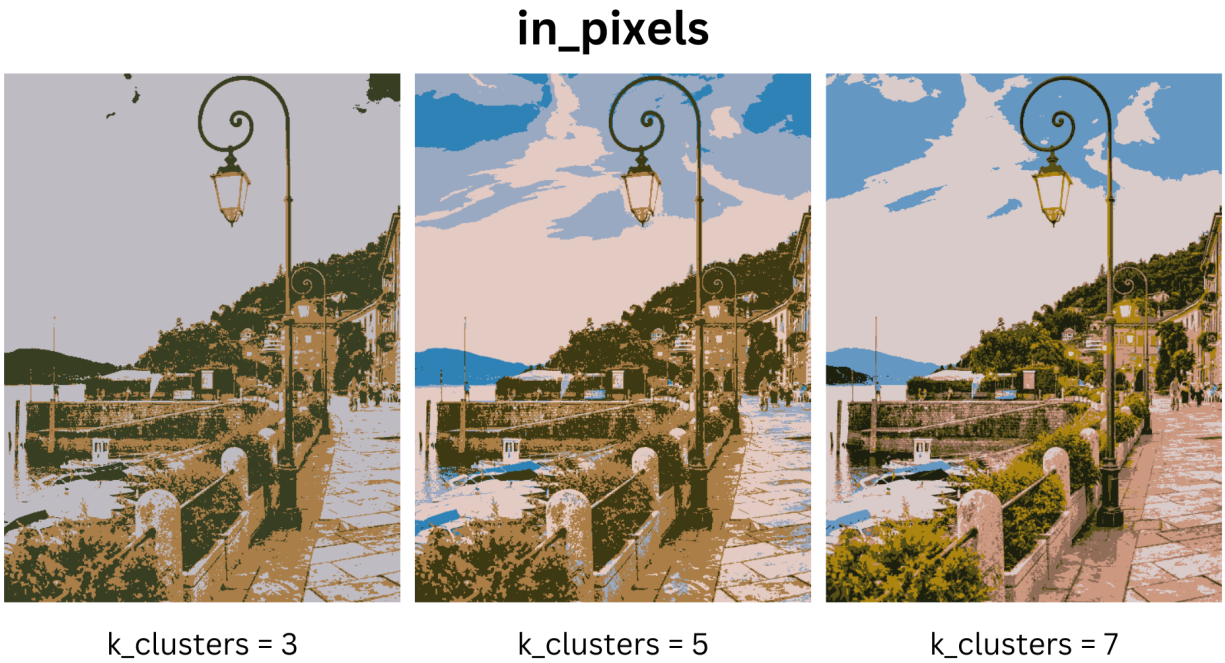
Hình 2: Ảnh dùng làm testcase. Nguồn: Vividsreen

Thông số ảnh dùng làm testcases:

Chiều rộng:	480
Chiều cao:	640
Số lượng màu:	135274
Dimension:	(480×640) pixels
Size:	223 KB

Hình ảnh kết quả

Hình 3: Kết quả thử nghiệm với kiểu khởi tạo random



Hình 4: Kết quả thử nghiệm với kiểu khởi tạo in_pixels

Bảng thông số ảnh và kết quả chạy		
	random	**in_pixels**
`k_clusters = 3`		
Runtime **Size**	0.56682s 37.4 KB	0.77191s 39.4 KB
`k_clusters = 5`		
Runtime **Size**	1.47187s 51.7 KB	2.86247s 51.7 KB
`k_clusters = 7`		
Runtime **Size**	2.14903s 74.3 KB	2.65578s 78.9 KB

Nhận xét

Nhìn tổng quát ta thấy được k_clusters càng lớn thì ảnh kết quả càng rõ nét và có màu sắc giống với ảnh gốc.

- Ở k=3 thì kiểu khởi tạo random không lấy được ánh lóa của đèn đường, trong khi kiểu khởi tạo in_pixels vẫn lấy được mặc dù nhìn chung cả 2 ảnh đều bị kéo về gần giống ảnh trắng đen.

- Ở k=5 và k=7 thì nhìn chung kiểu khởi tạo in_pixels vẫn cho ánh xanh của hàng cây và ánh lóa của đèn đường rõ hơn kiểu còn lại.

Có thể thấy thời gian chạy thuật toán có xu hướng tăng dần tỉ lệ thuận theo tham số k_clusters, do khi yêu cầu nhiều điểm hơn thì thời gian tốn cũng nhiều hơn để kéo số centroids đó về điểm hội tụ. Ngoại lệ xảy ra là do sự không chắc chắn và ổn định của tính chất chọn ngẫu nhiên (random).

Kích thước của ảnh đã giảm số lượng màu thì cũng giảm theo. Số lượng k_clusters càng nhỏ thì kích thước ảnh càng nhỏ và nhẹ hơn so với ảnh gốc.

⇒ **Kết luận:** Ảnh giảm số lượng màu nhìn chung chỉ thay đổi về màu sắc và độ chi tiết. Những sự vật của ảnh ban đầu tuy có nhỏ hay mờ hơn nhưng vẫn còn được biểu diễn chứ không hề mất đi.

5 Nguồn tham khảo

- Tài liệu hướng dẫn Lab và thực hành **Đồ án 1** của môn Toán Ứng dụng & Thống kê.
- Tài liệu thư viện *Numpy*
- Tài liệu thư viện *Pillow*.
- Dùng Numpy để *tính khoảng cách* hiệu quả hơn.
- Giới hạn *input* cho người dùng.
- Lấy *index* nhanh trong hàm gán nhãn.
- *Thay thế các giá trị* dựa trên điều kiện mà không cần lặp.