Họ và tên	Kiều Công Hậu
MSSV	**18127259**

TOÁN ỨNG DỤNG VÀ THỐNG KÊ – 18CLC1

Đồ án 1: Color Compression

1. Ý tưởng thực hiện (mô tả chi tiết các hàm)

Mã nguồn đã được ghi chú khá chi tiết từng bước làm. Đọc mô tả dưới đây kết hợp với mã nguồn để hiểu rõ hơn. (Đọc mã nguồn theo thứ thứ tự các hàm được liệt kê dưới đây.)

- ***Hàm main***

- Nhập đường dẫn của hình ảnh cần nén.
- Mặc định nén với số lượng màu là: 3, 5 và 7; tương ứng với 2 kiểu khởi tạo các trung tâm: *random* và *in_pixels*.
- Như vậy, trong mã nguồn này, mặc định sẽ nén tổng cộng 6 hình. 6 hình này sau khi được nén (gọi hàm color_compression với các tham số đầu vào tương ứng) sẽ được lưu trong một danh sách là *img_list*.
- Sau khi nén, nội dung của mỗi hình được lưu dưới mảng 2 chiều với mỗi phần tử là 1 điểm ảnh, ta tiến hành đổi nó sang dạng ảnh RGB và lưu vào thư mục cùng cấp với nơi thực thi mã nguồn này.

- ***Hàm color_compression***

Hàm này được dùng để nén hình với k màu.

- Đầu vào:
  - *img_path* (str): đường dẫn của hình cần nén.
  - *k_clusters* (int): số lượng màu muốn nén.
  - *max_iter* (int): số vòng lặp tối đa cho thuật toán gom nhóm (kmeans).
  - *init_centroids* (str): cách khởi tạo các trung tâm (chỉ có 2 giá trị là *random* và *in_pixels*).
- Đầu ra:
  - *img* (np.ndarray): nội dung của bức hình sau khi được nén được lưu dưới dạng mảng của numpy với shape=(cao, rộng, 3).
  - Lưu ý: Hàm này trả về *None* nếu có lỗi trong quá trình nén.

- Kiểm tra thông số đầu vào hợp lệ (*k_clusters* và *max_iter* phải là số nguyên dương; *init_centroids* chỉ có 2 giá trị là *random* và *in_pixels*).
- Đọc nội dung của ảnh dựa vào đường dẫn, nếu không đọc được thì báo lỗi và trả về *None*.
- Chuyển đổi nội dung ảnh vừa đọc được sang ảnh RGB, rồi đổi sang lưu trữ dưới dạng mảng của numpy.
- Lúc này, nội dung của ảnh được lưu ở mảng có shape=(cao, rộng, 3), ta tiến hành đổi shape thành (cao * rộng, 3) để biến thành mảng 1 chiều cho dễ xử lý. Lúc này, mỗi phần tử của mảng 1 chiều này là 1 điểm ảnh, với điểm ảnh là mảng 3 phần tử có dạng [R, G, B].
- Truyền nội dung ảnh này cùng các thông số đầu vào khác vào hàm kmeans để gom nhóm các màu của ảnh lại. Sau khi nén ảnh, hàm kmeans này trả về 2 giá trị: *centroids* và *labels*:
  - *centroids*: danh sách k_clusters màu trung tâm.
  - *labels*: danh sách gồm cao*rộng phần tử, mỗi phần tử có giá trị thuộc { 0, 1, 2,… *k_clusters* – 1 } thể hiện trung tâm mà điểm ảnh tương ứng thuộc về.
- Cập nhật lại từng điểm ảnh dựa vào *centroids* và *labels*.
- Trả về nội dung của ảnh sau khi nén dưới dạng mảng của numpy với shape=(cao, rộng, 3).

- ***Hàm kmeans***

Hàm này được dùng để gom nhóm các điểm ảnh của bức ảnh về *k_clusters* cụm.

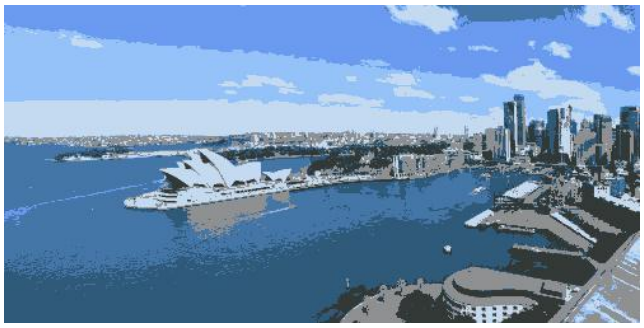
- Đầu vào:
  - *img_1d* (np.ndarray): nội dung của bức ảnh được lưu dưới dạng mảng của numpy với shape=(cao * rộng, 3).
  - *k_clusters* (int): số lượng màu muốn nén (= số lượng cụm muốn gom).
  - *max_iter* (int): số vòng lặp tối đa cho thuật toán gom nhóm.
  - *init_centroids* (str): cách khởi tạo các trung tâm (chỉ có 2 giá trị là *random* và *in_pixels*).
- Đầu ra:
  - *centroids* (np.ndarray): danh sách k_clusters màu trung tâm.
  - *labels* (np.ndarray): danh sách gồm cao*rộng phần tử, mỗi phần tử có giá trị thuộc { 0, 1, 2,… *k_clusters* – 1 } thể hiện trung tâm mà điểm ảnh tương ứng thuộc về.
- Khởi tạo 2 giá trị trả về của hàm này là *centroids* và *labels*.

- Nếu *init_centroids* là *random* thì chọn ngẫu nhiên *k_clusters* màu phân biệt bất kỳ làm trung tâm.
- Nếu *init_centroids* là *in_pixels* thì chọn ngẫu nhiên *k_clusters* màu phân biệt từ bức ảnh này là trung tâm. Có 1 trường hợp khá đặc biệt là số lượng màu có trong ảnh gốc ít hơn hẳn số lượng màu mà ta muốn nén, lúc này, ta sẽ cập nhật lại k_clusters = số lượng màu phân biệt có trong ảnh gốc. (Giả sử nén bức hình lá cờ Việt Nam chỉ có 2 màu là vàng và đỏ với *k_clusters* = 5 màu thì k_cluster sẽ được cập nhật lại thành 2.)

- Tìm bộ trung tâm mới cho đến khi nhãn của các điểm ảnh không có sự thay đổi nữa hoặc đạt mức giới hạn *max_iter*.
  - Copy bộ *labels* cũ sang bộ *pre_labels*.
  - Tạo danh sách *clusters*, trong đó *clusters[i]* là một danh sách lưu trữ các index của các điểm ảnh thuộc cùng cụm *i*, i thuộc { 0, 1, …, *k_clusters* - 1 }.
  - Duyệt từng điểm ảnh, tính khoảng cách từ điểm ảnh này đến tất cả các trung tâm, sau đó gán nhãn trung tâm gần điểm ảnh này nhất cho điểm ảnh đó, đồng thời thêm index của điểm ảnh này vào trung tâm gần nhất đó.
  - Nếu pre_labels giống y hệt labels, tức không có sự thay đổi nhãn giữa các điểm ảnh của hình, ta dừng vòng lặp.
  - Duyệt từng cụm, tính lại giá trị trung tâm của cụm đó bằng means (lấy trung bình giữa các điểm ảnh). Nếu cụm bất kỳ không có điểm ảnh nào thuộc về thì giữ lại màu của trung tâm cũ, tức không thay đổi giá trị của trung tâm.
  - Giảm giới hạn lặp xuống 1 đơn vị cho đến khi *max_iter* = 0 thì thoát vòng lặp.
- Trả về danh sách các màu trung tâm (*centroids*) và danh sách nhãn của từng điểm ảnh (*labels*).

2. Hình ảnh kết quả

Hình gốc

	init_centroids = 'random'	***init_centroids = 'in_pixels'***
k = 3		
k = 5		
k = 7		

3. Nhận xét

- Ảnh gốc có kích thước 500px * 250px, với tổng số màu phân biệt là 45872.
- Trong cả 6 hình, số lượng màu đã được giảm xuống còn 3 hoặc 5 hoặc 7 màu, ảnh không còn được chi tiết nhưng ta đều dễ dàng nhận biết được nội dung của bức ảnh, chứng tỏ việc cài đặt thuật toán đã có sự thành công nhất định.
- Với k = 3, cả 2 tấm ảnh đều chưa thể hiện tốt được các đám mây và cái bóng của tòa Sydney Opera House dưới dòng sông.

HẾT