ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Khoa Công Nghệ Thông Tin

----------***----------

BÁO CÁO

Project 1 – Color Compression

Tác giả

Nguyễn Phúc Khang - 22127180

Lớp

22CLC08

Môn học

Toán ứng dụng và thống kê

cho Công Nghệ Thông Tin

Giảng viên hướng dẫn

Vũ Quốc Hoàng

Nguyễn Văn Quang Huy

Nguyễn Ngọc Toàn

Phan Thị Phương Uyên

Thành phố Hồ Chí Minh – 2024

Mục Lục

1. Ý tưởng thực hiện

Mục đích của báo cáo này là sử dụng thuật toán K-Means để phân đoạn màu sắc của một hình ảnh, sau đó biểu diễn kết quả với số lượng màu k khác nhau: k = 3, 5, và 7. Chúng tôi sẽ tiến hành thử nghiệm với các giá trị k khác để đánh giá sự thay đổi của kết quả.

2. Mô tả các hàm

2.1.read_img(img_path)

- **Chức năng:** đọc file ảnh từ đường dẫn được truyền vào.
- **Đầu vào:** tham số **img_path** là đường dẫn tới file ảnh.
- **Đầu ra:** trả về đối tượng thuộc lớp **Image** nếu thành công, ngược lại thất bại sẽ trả về **None**.
- **Mô tả chi tiết:** Sử dụng lớp Image từ thư viện **PIL** để mở file ảnh. Nếu thành công hàm sẽ trả về đối tượng ảnh thuộc lớp Image, nếu file không tồn tại hoặc có lỗi khi mở, hàm sẽ trả về **None** và in ra thông báo lỗi tương ứng.

2.2.show_img(img_2d)

- **Chức năng**: hiển thị hình ảnh 2D.
- **Đầu vào**: tham số **img_2d** là đối tượng ảnh 2D.
- **Đầu ra**: không có giá trị trả về, hàm sẽ hiển thị ảnh ra màn hình.
- **Mô tả chi tiết**: Sử dụng phương thức **imshow** và **show** của thư viện **matplotlib** để hiển thị ảnh. Phương thức **imshow** nhận đầu vào là 1 đối tượng ảnh **Image** hoặc **mảng 2D, 3D** để tạo ra hình ảnh nhưng không hiển thị ngay lập tức, ta cần sử dụng phương thức **show** để hiển thị hình ảnh ra màn hình.

2.3.save_img(img_2d, img_path)

- **Chức năng**: lưu ảnh 2D vào một file với đường dẫn được cung cấp.
- **Đầu vào**:
  - **img_2d:** đối tượng ảnh 2D dưới dạng mảng **Numpy**.
  - **img_path:** đường dẫn tới file ảnh và tên ảnh cần lưu.
- **Đầu ra**: không có giá trị trả về.
- **Mô tả chi tiết**: chuyển đổi mảng ảnh thành đối tượng **Image** và lưu nó bằng phương thức **save** của **PIL**, mỗi giá trị số nguyên trong mảng được lưu dưới dạng số nguyên 8 bits.

2.4.convert_img_to_1d(img_2d)

- **Chức năng**: chuyển đổi ảnh 2D thành mảng 1D.
- **Đầu vào**: tham số **img_2d** là đối tượng ảnh 2D.
- **Đầu ra**: trả về mảng 1D của ảnh.
- **Mô tả chi tiết**: chuyển đổi ảnh 2D thành mảng Numpy và sau đó **reshape** mảng này để mỗi pixel trở thành một dòng của mảng 1D**.**

2.5.kmeans(img_1d, k_clusters, max_iter, init_centroids='random')

- **Chức năng**: thực hiện thuật toán K-Means trên mảng ảnh 1D.
- **Đầu vào**:

  - **img_1d:** mảng ảnh 1D.
  - **k_clusters:** số lượng cluster muốn phân đoạn.
  - **max_iter:** số lần lặp tối đa cho thuật toán.
  - **init_centroids:** phương pháp khởi tạo centroids, có thể là **random** hoặc **in_pixels**.
- **Đầu ra**: Trả về mảng **centroids** và **labels** tương ứng với mảng 1D.
- **Mô tả chi tiết**:
  - **Khởi tạo centroids:** mảng centroids có kích thước **k_clusters** và phần tử của mảng là các màu RGB.
    - Nếu init_centroids là random thì các centroids được tạo ngẫu nhiên.
    - Nếu init_centroids là in_pixels thì các centroids được lấy từ các pixels của ảnh.
    - Đối với cả random và in_pixels, sau khi khởi tạo mảng xong ta sẽ dùng phương thức **unique** để tạo ra mảng các centroids phân biệt, nếu độ dài mảng unique bé hơn k_clusters – tức có phần tử trùng, ta sẽ chọn lại phần tử mới thay cho phần tử bị trùng lắp, lặp lại cho tới khi độ dài mảng unique = k_clusters.
  - **Thực thi thuật toán KMeans:** trong mỗi lần lặp (số lần lặp tối đa là max_iter) ta thực hiện các bước sau:
    - Tính khoảng cách từ mỗi pixel đến các centroids bằng phương thức **norm** của Numpy và lưu vào mảng **distance**. Như vậy distance sẽ là mảng 2 chiều, mỗi dòng là 1 pixel và giá trị mỗi cột là khoảng cách từ pixel đó đến 1 centroid.
    - Với mỗi pixel, ta chọn ra centroid gần nhất, số index của centroid trên dòng cũng chính là **label** của pixel đó. Ta sử dụng phương thức **armin** để tìm và trả về số index của giá trị nhỏ nhất, đối số truyền vào là mảng distance, kết quả được lưu lại trong mảng **labels**.
    - Cập nhật centroids mới bằng cách tính trung bình các pixel thuộc mỗi cluster (các pixel có cùng label) sau đó gán lại cho centroid tương ứng. Nếu cluster không có pixel nào thì giữ nguyên giá trị centroid.
    - Sau bước cập nhật centroids nếu không thay đổi, dừng thuật toán, trả về centroids và labels.

2.6.generate_2d_img(img_2d_shape, centroids, labels)

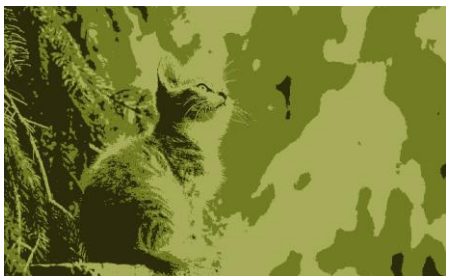
- **Chức năng**: tạo ảnh 2D từ các centroids và labels.
- **Đầu vào**:
  - **img_2d_shape:** kích thước của ảnh 2D (chiều cao, chiều rộng, số kênh màu).
  - **centroids:** mảng centroids.
  - **labels:** mảng labels.
- **Đầu ra**: Trả về ảnh 2D mới.
- **Mô tả chi tiết**: với mỗi giá trị trong mảng labels là vị trí index của pixel trong mảng centroids, cập nhật centroid tương ứng với mỗi labels và reshape lại thành ảnh 2D**.**

3. Kiểm thử và nhận xét

3.1.Kiểm thử

- Thực thi ảnh với kích thước như hình bên dưới, màu trung tâm được khởi tạo ngẫu nhiên (random) và chọn pixel sẵn có trong hình (in_pixels), số màu gom nhóm (cluster - k) lần lượt là 3, 5 và 7, thời gian thực thi được ghi liền kề.

random

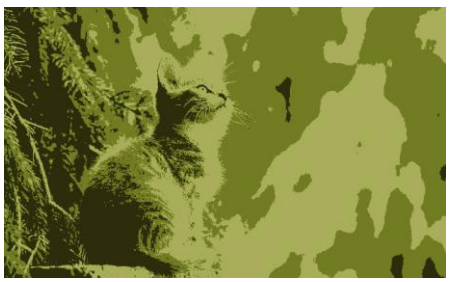


k=3 – 1.24s

k=7 – 3.53s

k=7 – 2.96s

in_pixelss



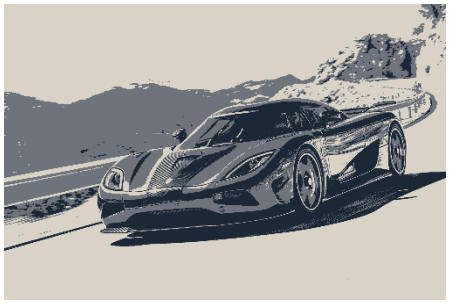
k=3 – 1.16s

k=5 – 1.82s

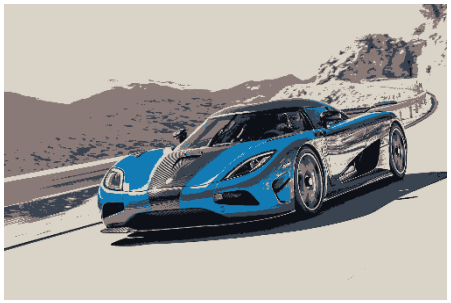
k=7 – 2.08s

- Ta thử chạy thuật toán với ảnh có kích thước lớn hơn.

random



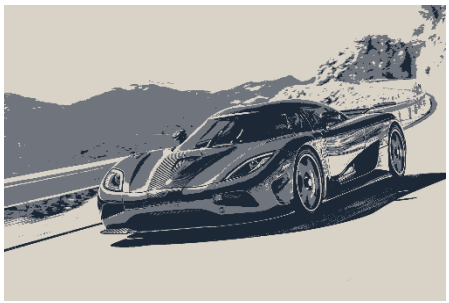
k=3 – 6.112.3



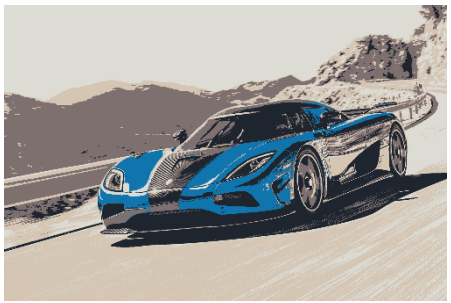
k=7 – 14.29s

k=7 – 16.25s

in_pixels



k=3 – 9.67s



k=5 – 14.90s

k=7 – 41.40s

3.2.Nhận xét

- Khi số lượng cluster tăng, thời gian thực thi cũng tăng vì thuật toán cần phân chia nhiều nhóm hơn, dẫn đến thời gian xử lý dài hơn. Điều này có thể thấy rõ qua thời gian thực thi với k lần lượt là 3, 5 và 7.
- **Ảnh kích thước nhỏ**:
    - Thời gian thực thi khá ngắn, sự chênh lệch thời gian thực thi khi thay đổi số lượng cluster không quá nhiều.
    - Khởi tạo ngẫu nhiên có thời gian thực thi dài hơn một chút so với chọn pixel sẵn có, nhưng sự khác biệt này cũng không quá lớn.

- **Ảnh kích thước lớn:**
    - Khi kích thước ảnh tăng, thời gian thực thi và sự chênh lệch giữa số lượng cluster được thấy rõ rệt hơn.
    - Ngược lại với ảnh có kích thước nhỏ, phương pháp chọn pixel sẵn có lại cho thấy thời gian thực thi lâu hơn so với khởi tạo ngẫu nhiên trong một số trường hợp. Điều này cho thấy việc chọn điểm bắt đầu ảnh hưởng lớn hơn đến thời gian hội tụ của thuật toán khi kích thước ảnh và số lượng cluster tăng.

- **Tổng kết:**
    - Ảnh kích thước lớn làm tăng thời gian thực thi đáng kể so với ảnh kích thước nhỏ. Tăng số lượng cluster dẫn đến thời gian thực thi của thuật toán có xu hướng tăng (số ít trường hợp giảm).
    - Phương pháp chọn in_pixel thường nhanh hơn khi ảnh nhỏ nhưng có thể chậm hơn với ảnh lớn và số lượng cluster cao.
    - Việc chọn điểm bắt đầu ảnh hưởng nhiều đến tốc độ thực thi của thuật toán.

4. Tài liệu tham khảo

- Wikipedia contributors, "K-means clustering," Wikipedia, The Free Encyclopedia, Link: https://en.wikipedia.org/wiki/K-means_clustering
- NgocTien0110, "Applied Mathematics and Statistics Repository," GitHub. Link: https://github.com/NgocTien0110/Applied-Mathematics-and-Statistics.
- "NumPy Documentation," NumPy. Link: https://numpy.org/doc/stable/reference/index.html. Accessed date: June 15, 2024.
- "Matplotlib Tutorials," Matplotlib. Link: https://matplotlib.org/stable/tutorials/index.html, Accessed date: June 15, 2024.