Lab01: Color Compression

Courses `MTH00051`: Toán ứng dụng và thống kê
`18CLC6`, `FIT - HCMUS`.
`16/08/2020`

Đây là đồ án cá nhân, do một thành viên thực hiện:

- `18127231`: Đoàn Đình Toàn (GitHub: @t3bol90)

Về đồ án này:

Đề bài ở::
https://courses.ctda.hcmus.edu.vn/mod/resource/view.php?id=21162

Giới thiệu:

Một bức ảnh có thể lưu trữ dưới ma trận của các điểm ảnh. Có nhiều loại ảnh được sử dụng trong thực tế, ví dụ: ảnh xám, ảnh màu,...

Yêu cầu:

Trong đồ án này, bạn được yêu cầu cài đặt thuật toán K-Means để giảm số lượng màu cho ảnh. Các thư viện được phép sử dụng là: NumPy (tính toán ma trận), PIL (đọc, ghi ảnh), matplotlib (hiển thị ảnh).

Môi trường thực hiện

Đồ án mình đã sử dụng `python 3.7` và sử dụng `Google Colab` để chạy. Ban đầu mình sử dụng `Jupyter notebook` để chạy ở local nhưng `kernel dead` hoài (do máy cùi không đủ ram) nên mình quyết định chuyển lên Colab để hoàn thành đồ án này.

Ý tưởng thực hiện:

Đồ án đã cho một khuôn hàm có sẵn như sau:

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids='random'):
    '''
    K-Means algorithm

    Inputs:
        img_1d : np.ndarray with shape=(height * width, num_channels)
            Original image in 1d array

        k_clusters : int
            Number of clusters

        max_iter : int
            Max iterator
```

```
        init_cluster : str
            The way which use to init centroids
            'random' --> centroid has `c` channels, with `c` is initial random
in [0,255]
            'in_pixels' --> centroid is a random pixels of original image

    Outputs:
        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)
'''
```

Ý tưởng là dùng `PIL` để đọc ảnh, sau đó reshape ảnh từ một scalar ba chiều thành một mảng 2 chiều dạng `[height*width, n_channel]`, ở đây `n_channel = 3`.

Ứng với mỗi kiểu init centroids là random trong khoảng [0,255] hay random pick từ màu của ảnh, ta thực hiện init cho centroids là k điểm màu với k là số cluster (3,5 hoặc 7).

Sau đó ta chạy đúng `max_iter` lần, mỗi lần, đối với một điểm ảnh lấy ra ta tính lại `distance` (L2 distance) và lấy `mean` của các cụm ảnh lấy ra sau đó gán lại cho `label` mới đồng thời cập nhật lại `centroids`.

Trong cách cài đặt của hiện tại đều sử dụng tất cả các hàm của thư viện `NumPy` và hạn chế sử dụng các vòng lặp nếu không cần thiết. Từ đó có thể cải thiện tốc độ của thuận toán.

Mô tả các hàm

Ở đây mình sử dụng lại khuôn hàm duy nhất của đề bài yêu cầu đó là hàm `kmeans` như ở trên đã đề cập. Còn lại không có các hàm phụ, sau khi trả ra được labels và centroids từ `kmeans` thì ta chỉ cần gán lại cho ảnh bằng một đoạn for:

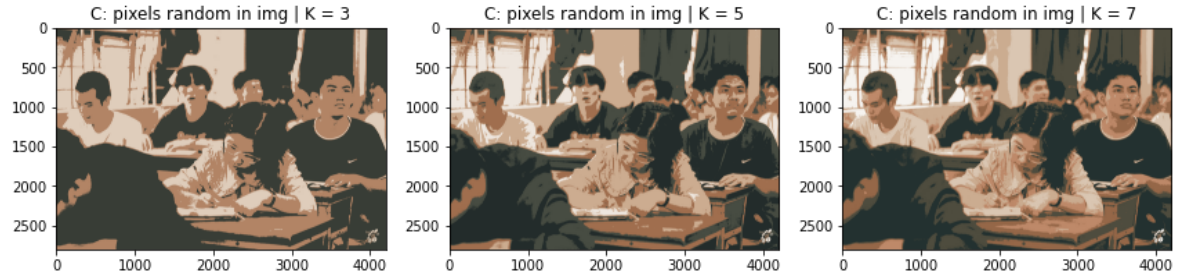
```
## Gan label lai cho anh
for k in range(centroids.shape[0]):
  img_ptest[labels == k] = centroids[k]
```

Sau đó ta hiển thị ảnh lên lại, hoặc có thể lưu ảnh luôn bằng `plt.savefig(path)` để lưu ảnh (vì đồ án không cho sử dụng thư viện khác để đọc lưu ảnh tốt hơn như `opencv2` chẳng hạn 😔).

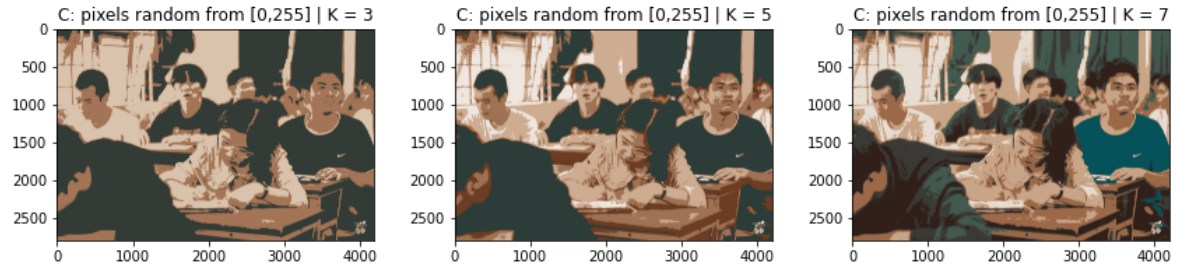
Kết quả:

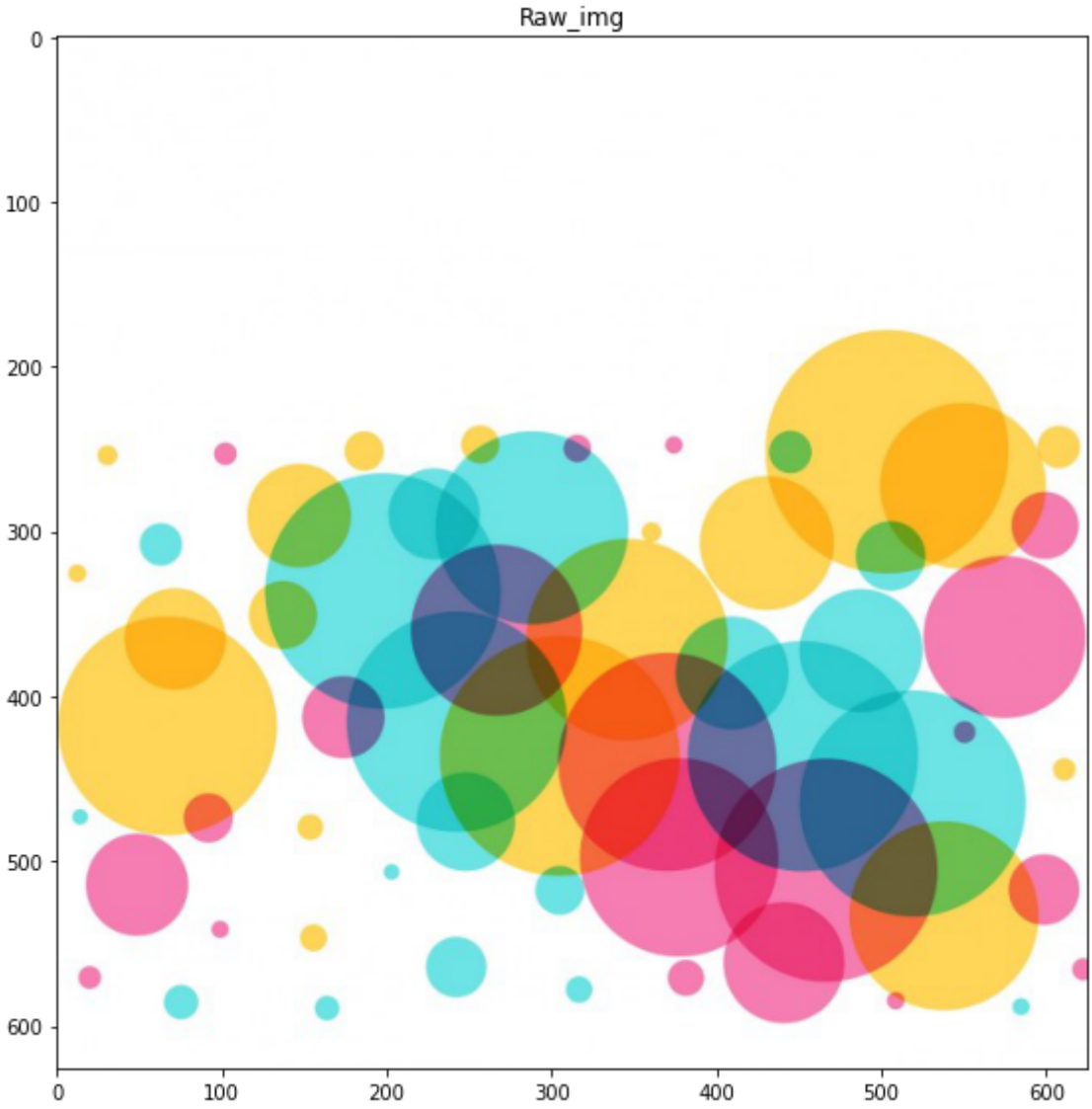
Ở trong file notebook, mình có viết hai section riêng biệt, một section để test nhanh tấm ảnh với input tùy chỉnh từ người dùng và một section test tất cả các trường hợp `['random', 'in_pixels'] x k = {3,5,7}.` Một số kết quả thu được như dưới đây:

Với 'in_pixels':

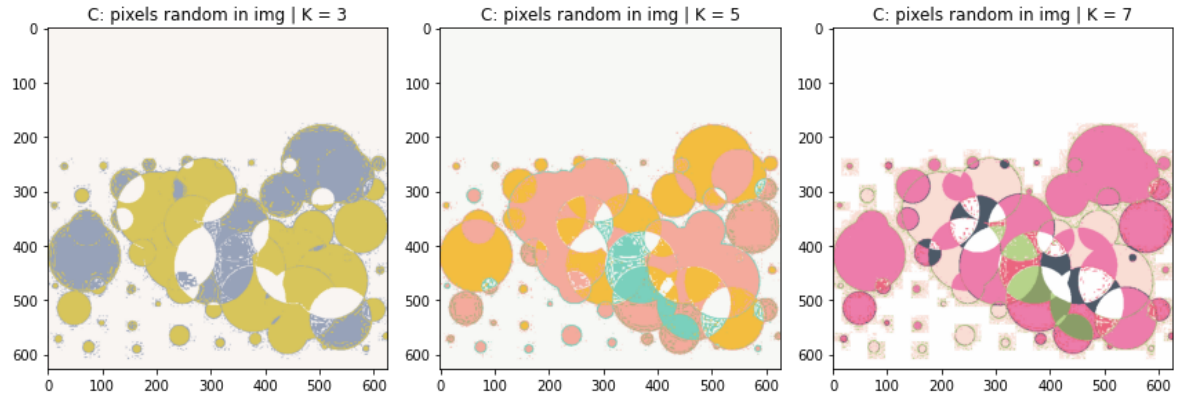


Với 'random':

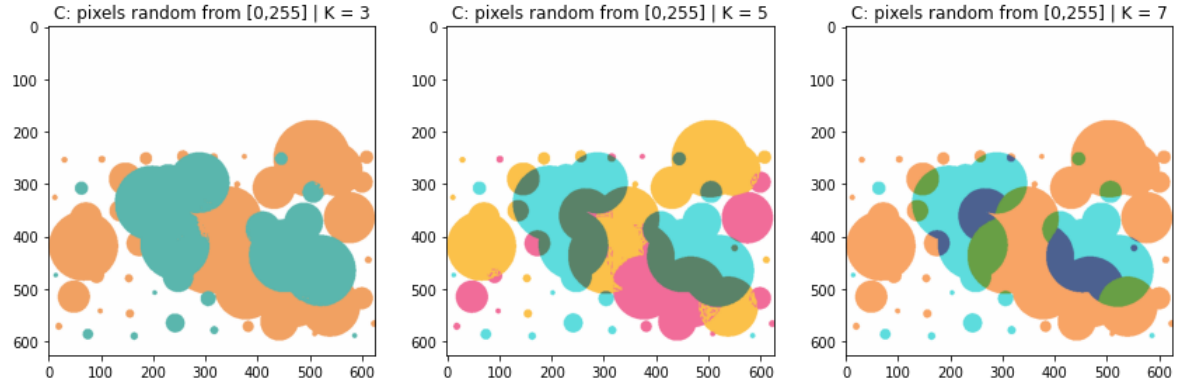




Với 'in_pixels':



Với 'random':



Kết luận:

Kết quả trên là chấp nhận được, nếu so sánh với Kmean của `scikit-learn` thì ta có hiệu quả cũng xấp xỉ tương đương. Tuy nhiên hiệu suất nén ảnh đối với các trường hợp ảnh nhỏ (ảnh đã được nén - giảm màu bằng phần mềm khác) hoặc ảnh đơn sắc thường ra kết quả không tốt với max_iter thấp.

Ngoài ra, nếu ta vẽ biểu đồ về sự hội tụ của Kmeans, khi ta init 'random' thì tốc độ hội tụ sẽ nhanh hơn là cách init lấy từ điểm trên ảnh trong một số trường hợp. Trường hợp ảnh có nhiều màu sắc thì lấy random cho ra hiệu suất tốt hơn.

References:

Have a Great Day