VIET NAM NATIONAL UNIVERSITY,HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF: INFORMATIONTECHNOLOGY

Báo cáo đồ án 1

Toán ứng dụng và thống kê cho Công nghệ Thông tin

Họ và tên sinh viên: Lại Minh Thông
Mã số sinh viên: 20127635

Giáo viên hướng dẫn:

Thầy Nguyễn Văn Quang Huy

Thầy Vũ Quốc Hoàng

Cô Phan Thị Phương Uyên

Thầy Lê Thanh Tùng

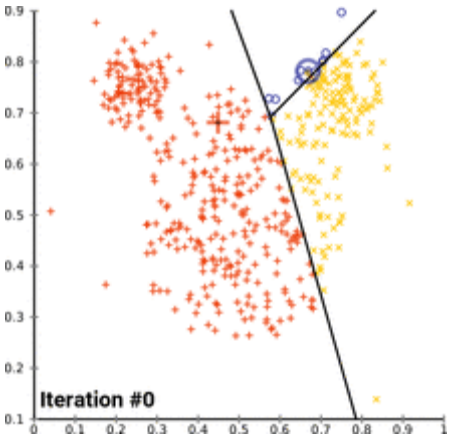
Thành phố Hồ Chí Minh – 2022

Đồ án 1 – Color Compression

I. Thuật toán K-means.

Giới thiệu.

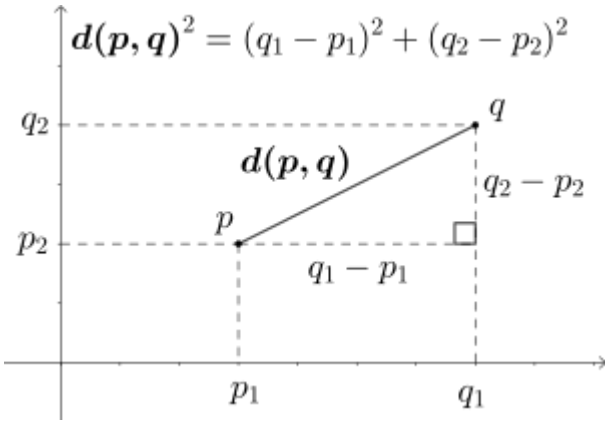
Thuật toán K-means(phân cụm K-means) là một phương thức để lượng tử hóa hay phân cụm các vector, ban đầu được sử dụng cho việc xử lý tín hiệu trong ngành kỹ thuật điện tử.



1 - K-means Visualization

Mục tiêu chính là để phân vùng cho n các cá thể quan sát thành k cụm khác nhau, và mỗi cụm sẽ được đại diện bởi 1 điểm cụm hoặc điểm trung tâm(centroid/center) thứ k. Trong đó mỗi cá thể quan sát sẽ thuộc về 1 cụm thứ k nếu cá thể đó có khoảng cách đến điểm cụm thứ k gần nhất. Vị trí của điểm trung tâm thứ k sẽ là điểm trung tâm của các cá thể quan sát trong cụm đó.

Với thuật toán k-means tiêu chuẩn thông thường, khoảng cách của từng cá thể đến 1 điểm trung tâm sẽ được tính theo khoảng cách Euclidean bình phương (Square Euclidean Distance).



2 - Square Euclidean Distance

Phân cụm k-means có nhiều ứng dụng, nhưng được sử dụng nhiều nhất trong Trí tuệ nhân tạo và Học máy (cụ thể là Học không có giám sát).

Thuật toán.

Đầu tiên chúng sẽ tạo ra các điểm trung tâm ngẫu nhiên. (Ở một số biến thể khác với k-means tiêu chuẩn, các điểm trung tâm ban đầu có thể được tạo ra dựa vào 1 hàm ước lượng heuristic khác nhau thay vì chọn ngẫu nhiên, như k-means++)

Gán mỗi điểm quan sát trong tập dữ liệu vào điểm trung tâm gần nó nhất (Sử dụng Square Euclidean Distance như đã nêu ở trên để tính khoảng cách). Sau đó cập nhật lại điểm trung tâm, vị trí điểm trung tâm mới sẽ bằng trung bình cộng của các điểm thuộc điểm trung tâm cần cập nhật và tiếp tục lặp lại các bước đã kể trên.

Điều kiện dừng của thuật toán: Khi các trung tâm không thay đổi trong 2 vòng lặp kế tiếp nhau. Tuy nhiên, việc đạt được 1 kết quả hoàn hảo (tức các trung tâm không thay đổi trong 2 vòng lặp kế tiếp) là rất khó và rất tốn thời gian, vậy nên thường người ta sẽ cho dừng thuật toán khi đạt được 1 kết quả gần đúng và chấp nhận được bằng cách cho cụ thể số lần lặp giới hạn.

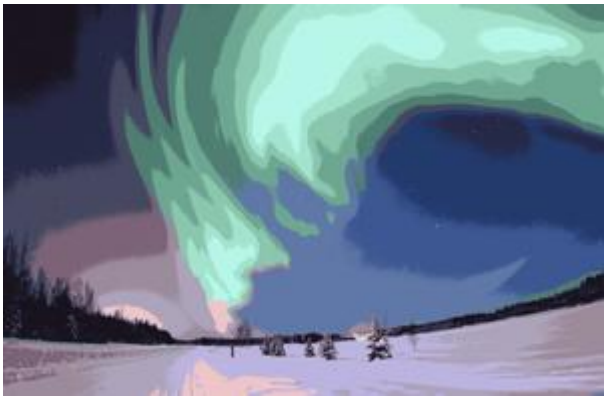
II. Bài toán nén màu ảnh - Color Compression

Bài toán

K-means ban đầu được dùng cho việc xử lý tín hiệu, và hiện nay vẫn còn được sử dụng để giải quyết các vấn đề trong lĩnh vực Khoa học Máy tính.

Như trong Đồ họa Máy tính, lượng tử hóa màu(color quantization) là công việc giảm số lượng bảng màu của 1 hình ảnh xuống một số lượng màu cố định k để đỡ tốn phí lưu trữ màu. Đây là một trong những bài toán kinh điển mà thuật toán K-means có thể được ứng dụng được và giải quyết dễ dàng, tạo ra các kết quả cạnh tranh khác nhau(Hình ảnh được qua xử lý mỗi lần sẽ khác nhau với cùng số lượng cụm k). Đây cũng là mục tiêu của đồ án này.

3 - Ảnh gốc



4 - Ảnh qua xử lý nén màu sử dụng k-means với k = 16

Một bức ảnh có thể lưu trữ dưới ma trận của các điểm ảnh(pixels). Ở đồ án này, loại ảnh được sử dụng để nén màu là ảnh RBG, tức mỗi pixel sẽ có 3 giá trị là R (red – đỏ), G(green - xanh), B(blue – xanh dương) với miền giá trị chạy từ 0 - 255. Vì thế ta sẽ biểu diễn mỗi điểm ảnh dưới dạng 1 vector được xác định bởi 3 giá trị tương ứng với R, G, B.

Để thực hiện giảm số lượng màu, ta cần tìm ra các đại diện có thể thay thế cho một nhóm màu. Cụ thể trong trường hợp ảnh RGB, ta cần thực hiện gom nhóm các pixel ($\mathbb{R}3$) và chọn ra đại diện cho từng nhóm. Như vậy, bài toán trên trở thành gom nhóm các vector pixels.

Thuật toán giải.

Ứng dụng thuật toán K-means, ta có kế hoạch để giải bài toán như sau:

1. Chọn k pixel trung tâm centroids với 3 giá trị R, B, G ngẫu nhiên trong khoảng 0 - 255.

2. Gán mỗi pixel trong hình với 1 cụm mà có khoảng cách tới centroid thứ k ngắn nhất trong k centroids.
3. Tính lại giá trị mới của các centroids bằng trung bình cộng các vector pixels trong cụm tương ứng.
4. Lặp lại bước 2 và 3 cho đến khi đạt được sự hội tụ(không có sự thay đổi các centroids trong 2 lần lặp liên tiếp hoặc số lần lặp đạt đến số lần lặp giới hạn được cho)

Ở đây khoảng cách giữa pixel tới 1 centroid sẽ được tính theo khoảng cách Euclidean bình phương.

Thuật toán trên đảm bảo sẽ hội tụ sau 1 thời gian. Chất lượng của lời giải với thuật toán trên phụ thuộc vào giá trị khởi tạo đầu tiên của k centroids và số lượng các cụm(tức giá trị k). Do việc khởi tạo giá trị một cách ngẫu nhiên của các centroids nên kết quả chất lượng hình ảnh sau khi nén màu cũng sẽ ngẫu nhiên và có thể không phải là kết quả tốt nhất.

Có thể xem xét sử dụng thuật toán Mean Shift, thuật toán này có được sự thuận lợi là không phải khởi tạo giá trị ban đầu ngẫu nhiên, điều này tạo ra lời giải và kết quả nhìn chung tốt hơn cho các trường hợp đa dạng. Tuy nhiên yêu cầu đồ án lần này sử dụng thuật toán chuẩn K-means.

III. Cài đặt chi tiết

Ngôn ngữ lập trình được sử dụng để giải quyết bài toán của đồ án là Python và trên môi trường Jupyter Notebook.

Mã nguồn: https://github.com/ThongLai/Kmeans-ColorCompression

Phần lớn mã nguồn được kham khảo từ "Ander fernández" - *How to program the kmeans algorithm in Python from scratch*. Tuy nhiên mã nguồn này sử dụng khá nhiều vòng lặp và thư viện pandas thay vì các phép toán ma trận của numpy, vì vậy một vài phép toán ma trận khác được tham khảo tại Stackoverflow và trang web document của thư viện numpy. Các đường dẫn đến website, tài liệu tham khảo được đặt ở phần *Tài liệu tham khảo*

Các thư viện cần thiết:

+ numpy – Tính toán các phép toán ma trận.

+ PIL – Các thủ thuật trên hình ảnh như đọc, lưu.

+ matplotlib.pyplot – Hiển thị hình ảnh.

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from numpy import linalg
```

5 - Khai báo thư viện

Các hàm:

1) *main.*

Hàm main chạy chính của chương trình, dùng để đọc hình ảnh người dùng, gọi thuật toán k-means để xử lý ảnh của người dùng và xuất ra kết quả.

- Cho phép người dùng nhập tên ảnh để đọc và lưu lại dưới dạng ma trận theo numpy.
- Đổi không gian ma trận thành dạng mảng các vector pixels (dài*rông, 3).
- Xử lý việc nén màu ảnh bằng hàm *kmeans*, với số cụm là *k*. Giá trị nhận được là mảng k các vector pixels *centroids* cùng mảng chứa giá trị phân cụm từng pixels ứng với 1 centroid - *labels*
- Thực hiện việc sắp xếp lại không gian các vector pixel thành không gian ma trận lúc đầu để hiển thị hình ảnh kết quả.
- Đồng thời cho phép người dùng đặt tên và lưu ảnh qua xử lý với định dạng theo lựa chọn.

```
def main():
    imName = input("Enter picture name: ")
    image = Image.open(imName, 'r')

    # Convert into np.array
    image_np = np.array(image)
    dim = image_np.shape

    #flatten image into 1d and reshape into an array of pixels
elements
    image_np = image_np.flatten()
    image_np = image_np.reshape(int(image_np.shape[0]/3), 3)

    #K-means algorithm
    k = 5
    centroids, labels = kmeans(image_np, k, 100)

    #Converted image
    kmeans_image = centroids[labels]

    #Reshape back into image's dimension
    kmeans_image = kmeans_image.reshape(dim)

    print("Image's size: ", kmeans_image.shape)
    plt.imshow(kmeans_image)

    #Save image
    imName = input("Enter picture name: ")

    fmat = ""
    while fmat.upper() not in ["PNG" ,"JPG" ,"PDF"]:
        fmat = input("Choose the format to save: PNG, JPG, PDF")

    generate_image(fmat, imName, kmeans_image)
```

2) *initialize_centroids.*

Đầu tiên cần phải khởi tạo các centroids với giá trị ngẫu nhiên từ 0 - 255.

```
def initialize_centroids(k, img_1d):
    n_dims = k, img_1d.shape[1]

    centroids = np.random.uniform(255, 0, n_dims)

    return centroids
```

6 - initialize_centroids

Tham số đầu vào: giá trị k, mảng một chiều chứa các vector pixels của ảnh.

- Khởi tạo 3 giá trị pixel cho mỗi centroid bằng hàm *random.uniform()* của numpy với giá trị ngẫu nhiên chạy từ 0 – 255.

Kết quả trả về: mảng chứa các centroids vừa khởi tạo.

3) *assign_centroid_errors.*

Phân cụm mỗi pixel tới cụm có centroid gần nhất và đồng thời lưu lại các khoảng cách của từng pixel đến centroid gần nhất.

Ban đầu việc thiết lặp hàm này được tìm hiểu và tham khảo theo trang Ander fernández - *How to program the kmeans algorithm in Python from scratch* với cách cài đặt duyệt theo vòng lặp như hàm *calculate_errors* sau:

```
def calculate_errors(centroids, img_1d):
    centroid_assign = np.array([])
    centroid_errors = np.array([])

    for pixel in img_1d:

        # Calculate the errror
        errors = np.array([])
        for centroid in centroids:
            error = np.sqrt(np.sum((centroid - pixel)**2))
            errors = np.append(errors, error)

        # Calculate closest centroid & error
        closest_centroid = np.argmin(errors)
        centroid_error = np.amin(errors)

        # Assign values to lists
        centroid_assign = np.append(centroid_assign, closest_centroid)
        centroid_errors = np.append(centroid_errors, centroid_error)
        print("loading...")

    print(centroid_assign)
    return (centroid_assign,centroid_errors)
```

7 - calulate_errors

Tham số đầu vào: mảng các centroids, mảng một chiều chứa các vector pixels của ảnh.

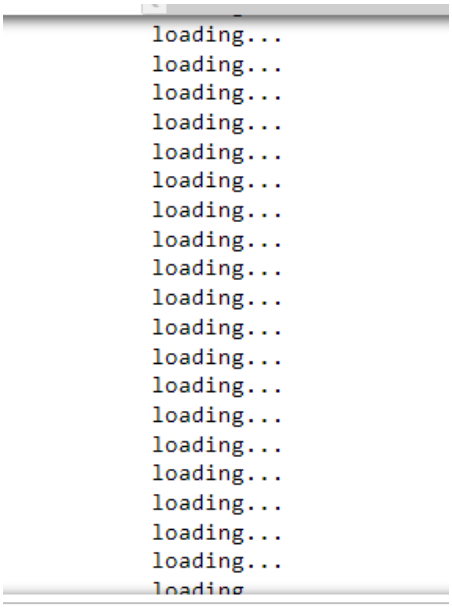
- Thực hiện vòng lặp duyệt từng pixel trong ảnh.
- Duyệt từng centroid và tính khoảng cách với pixel hiện tại theo cách tính Euclidean distance lưu vào mảng *errors*. Tức sau khi duyệt tất cả centroid, *errors* có không gian là kx1(ứng với mỗi pixel sẽ có k giá trị khoảng cách đến k centroids).
- Dùng hàm *argmin* của numpy để tìm ra centroid có khoảng cách tới pixel hiện tại nhỏ nhất trong mảng *errors*.
- Dùng hàm *amin* của numpy lấy khoảng cách nhỏ nhất trong

mảng *errors*.

- Thêm hai giá trị vừa tìm được vào 2 mảng *centroid_assign* và *centroid_errors*. Tiếp tục duyệt đến pixel thứ 2 và lặp lại các phép tính.

Kết quả trả về: mảng chứa giá trị phân cụm từng pixels - *centroid_assign*, khoảng cách sai số giữa từng pixel và centroid được phân cụm – *centroid_errors*.

Tuy nhiên sau khi cho hàm trên xử lý thử 1 hình ảnh có tổng cộng 701440 pixels với k = 5. Thì việc chờ đợi hình ảnh được xử lý rất lâu.



8- Chạy hàm calculate_error

Nhận thấy điều này không khả thi khi dùng vòng lặp duyệt từng phần tử để tính khoảng cách như bài viết của trang Ander fernández.Vì thế việc cố gắng giảm bớt vòng lặp thay bằng các phép toán ma trận sẽ giảm được chi phí tính toán. Ở đây có 1 vòng lặp con bên trong, vì vậy tìm cách bỏ đi vòng lặp con đó trước.

```
def calculate_errors(centroids, img_1d):
    ...

    for pixel in img_pixels:

        # Calculate the erroor
        errors = np.array([])
        for centroid in centroids:
            error = np.sqrt(np.sum((centroid - pixel)**2))
            errors = np.append(errors, error)

        # Calculate closest centroid & error
        ...
        ...
```

9 - Vòng lặp con duyệt từng centroid

Cụ thể ở đây, có thể thay vòng lặp con duyệt từng centroids để tính khoảng cách bằng phép tính ma trận như sau:

```
def calculate_errors(centroids, img_1d):
    ...

    for pixel in img_pixels:

        # Calculate the erroor
        errors = np.sqrt(np.sum((centroids - pixel)**2, axis=1))

        # Calculate closest centroid & error
        ...
        ...
```

Phép tính này tương đương với vòng lặp ở trên. Tuy nhiên thì sau khi thử nghiệm lại với hình ảnh và giá trị k trước đó, kết quả cũng không được khả quan. Điều này chứng tỏ việc dùng vòng lặp duyệt từng pixel làm cho hàm xử lý quá phức tạp. Vì vậy sau khi tham khảo bài viết của trang Stackoverflow tìm được cách tính khác hiệu quả hơn:

```
def assign_centroid_errors(centroids, img_1d):
    #Calculate square Euclidean distances for each centroid
    distance = np.sum((img_1d[:, None] - centroids)**2, axis = 2)

    labels = np.argmin(distance, axis = 1)
    centroid_errors = np.amin(distance, axis = 1)

    return (labels, centroid_errors)
```

Tham số đầu vào: mảng các centroids, mảng một chiều chứa các vector pixels của ảnh.

- Dùng phép toán ma trận theo numpy tính khoảng cách các pixels tới từng centroids theo Square Euclidean Distance lưu vào mảng *dístance*. *distance* có không gian là nxk (ứng với mỗi pixel sẽ có k giá trị khoảng cách đến k centroids) với n là số lượng pixels.
- Dùng hàm *argmin* của numpy để tìm ra centroid có khoảng cách tới pixel nhỏ nhất ứng với từng pixel trong mảng *distance*. Gán vào mảng *labels* với không gian là nx1.
- Dùng hàm *amin* của numpy lấy khoảng cách tới pixel nhỏ nhất ứng với từng pixel trong mảng *distance*. Gán vào mảng *centroids_errors* với không gian là nx1.

Kết quả trả về: mảng chứa giá trị phân cụm từng pixels ứng với 1 centroid - *labels*, khoảng cách sai số giữa từng pixel và centroid được phân cụm – *centroid_errors*.

4) *update_centroids*.

Cập nhật lại giá trị mới cho các centroids bằng giá trị trung bình cộng các vector pixel trong cụm.

```
def update_centroids(img_1d, labels, K):
    centroids = np.zeros((K, img_1d.shape[1]))
    for k in range(K):
        # collect all points assigned to the k-th cluster
        img_1d_k = img_1d[labels == k]

        # take average
        centroids[k] = np.mean(img_1d_k, axis = 0)
    return centroids
```

Tham số đầu vào: mảng một chiều chứa các vector pixels của ảnh, mảng đánh dấu phân cụm cho từng pixel, giá trị k.

- Duyệt từng centroid.
- Lấy ra những vector pixel được phân vào cụm của centroid hiện tại. Gán vào mảng *img_1d_k*.
- Tính giá trị trung bình cộng của *img_1d_k* và cho bằng giá trị mới của centroid hiện tại.

Kết quả trả về: mảng chứa các giá trị mới của centroids.

5) *kmeans.*

Hàm chạy thuật toán chính của K-means.

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

    centroids = initialize_centroids(k_clusters, img_1d)
    errors = np.array([0])
    labels = np.array([])
    compr = True
    i = 0

    #main loop
    while(compr and i < max_iter):
        # Obtain centroids and error
        labels, centroid_errors = assign_centroid_errors(centroids, img_1d)
        np.append(errors, sum(centroid_errors))

        # Recalculate centroids
        centroids = update_centroids(img_1d, labels, k_clusters)

        # Check if the error has decreased
        if(round(errors[i],3) !=  round(errors[i-1],3)):
            compr = True
        else:
            compr = False

        i = i + 1

    centroids = centroids.astype(np.uint8)
    return (centroids, labels)
```

Tham số đầu vào: mảng một chiều chứa các vector pixels của ảnh, giá trị k, số lượng vòng lặp giới hạn, chế độ khởi tạo centroids(mặc định là ‘random’)

- Sử dụng hàm *initialize_centroids* để khởi tạo các giá trị centroids.
- Vòng lặp while thực hiện từng bước của thuật toán đã mô tả và lặp lại cho đến khi hội tụ.

- Sử dụng hàm *assign_centroid_error* để phân cụm các pixels trong ảnh lưu lại ở biến *labels*, đồng thời lưu lại các giá trị khoảng cách nhỏ nhất giữa từng centroid đến các pixels và lưu ở *centroid_errors*.
- Tính tổng khoảng cách nhỏ nhất của mảng *centroid_errors* và lưu vào *errors*.
- Sử dụng hàm *update_centroids* để cài đặt lại giá trị các centroids mới sau khi phân cụm.
- Kiểm tra giá trị centroids có thay đổi trong 2 lần lặp kế tiếp bằng cách kiểm tra khoảng cách các centroids tới pixels nhỏ nhất của lần lặp 1 so với lần lặp 2 ở mảng *errors*.
- Khi số lần lặp vượt quá số lần lặp giới hạn hoặc không có sự thay đổi trong 2 lần lặp kế tiếp thì thuật toán đạt sự hội tụ và dừng xử lý.

Kết quả trả về: mảng chứa các giá trị cuối cùng của centroids - *centroids*, mảng chứa giá trị phân cụm từng pixels ứng với 1 centroid - *labels*.

IV. Đánh giá đồ án.

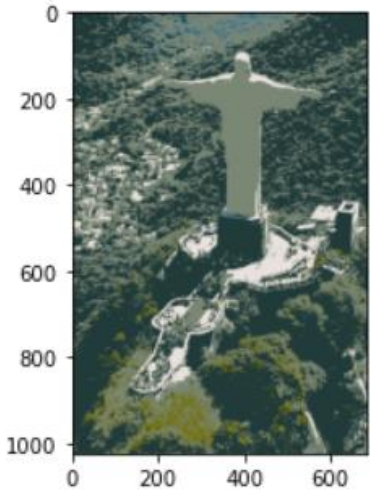
Thử nghiệm xử lý hình ảnh để đánh giá mức độ hoàn thiện của thuật toán như sau:

10 - Ảnh gốc 1024x685

Với k = 5:

Kết quả lần 1:

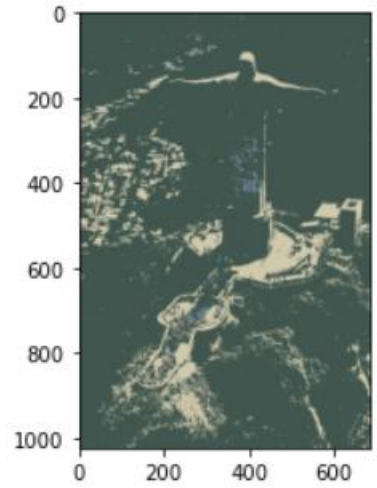
```
Enter picture name: test2.jpg
Number of k-clusters: 5
Image's size:  (1024, 685, 3)
Number of k-clusters:  5
Enter picture name to save: test
Choose the format to save (PNG, JPG, PDF): png
```



11 - Ảnh 1024x685 sau xử lý lần 1, k = 5

Kết quả lần 2:

```
Enter picture name: test2.jpg
Number of k-clusters: 5
Image's size:  (1024, 685, 3)
Number of k-clusters:  5
Enter picture name to save: test
Choose the format to save (PNG, JPG, PDF): png
```

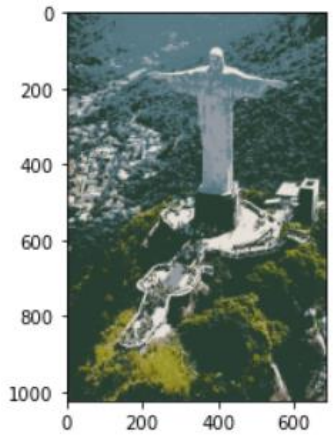


12 - Ảnh 1024x685 sau xử lý lần 2, k = 5

Nhận xét: Kết quả lần 1 ảnh vẫn có thể thấy rõ nét. Tuy nhiên kết quả ảnh lần 2 phân chia cụm chưa tốt dẫn đến ảnh khó nhìn hơn. Điều này xảy ra là do việc khởi tạo ngẫu nhiên các điểm trung tâm ban đầu có thể tạo ra trường hợp không cho ra lời giải tốt.

Với k = 10:

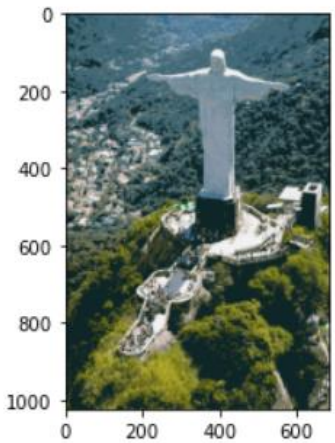
```
Enter picture name: test2.jpg
Number of k-clusters: 10
Image's size:  (1024, 685, 3)
Number of k-clusters:  10
Enter picture name to save: test
Choose the format to save (PNG, JPG, PDF): png
```



13- Ảnh 1024x685 sau xử lý, k = 10

Với k = 50:

```
Enter picture name: test2.jpg
Number of k-clusters: 50
Image's size:  (1024, 685, 3)
Number of k-clusters:  50
Enter picture name to save: test
Choose the format to save (PNG, JPG, PDF): png
```



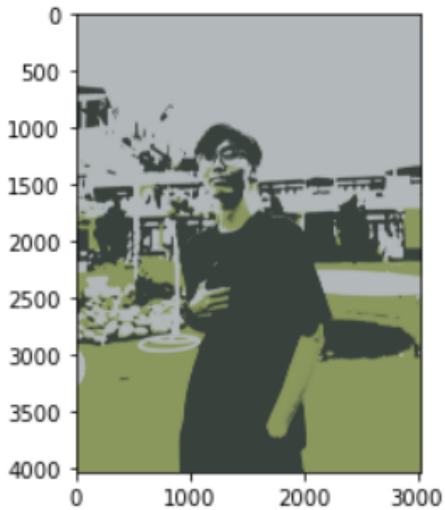
14- Ảnh 1024x685 sau xử lý, k = 50

Nhận xét: Với k = 10 chưa có sự khác biệt rõ về mức độ phủ màu của hình ảnh sau xử lý so vơi k = 5. Tuy nhiên khi số cụm chia thành 50, hình ảnh đã trở nên có màu sắc gần giống với ảnh gốc và đồng thời thời gian chạy cũng lâu hơn.

Hình ảnh kích thước khác:

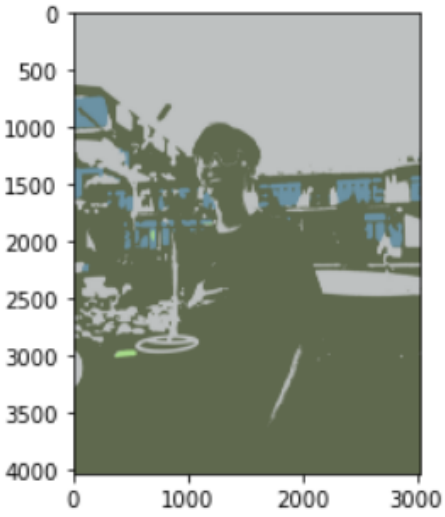
15- Ảnh gốc 4032x3024

```
Enter picture name: test.jpg
Number of k-clusters: 3
Image's size:  (4032, 3024, 3)
Number of k-clusters:  3
Enter picture name to save: test
Choose the format to save (PNG, JPG, PDF): PNG
```



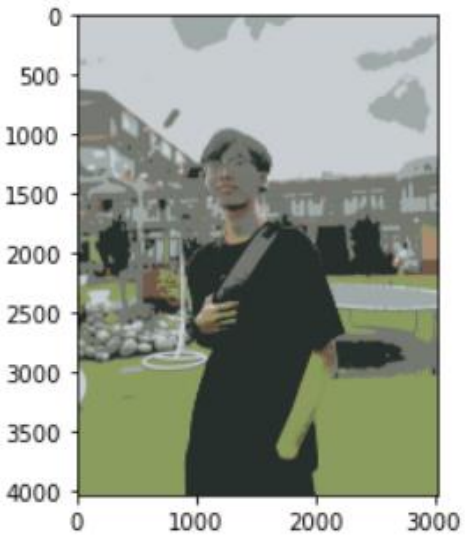
16 - Ảnh 4032x3024 sau xử lý, k = 3

```
Enter picture name: test.jpg
Number of k-clusters: 5
Image's size:  (4032, 3024, 3)
Number of k-clusters:  5
Enter picture name to save: test.png
Choose the format to save (PNG, JPG, PDF): png
```



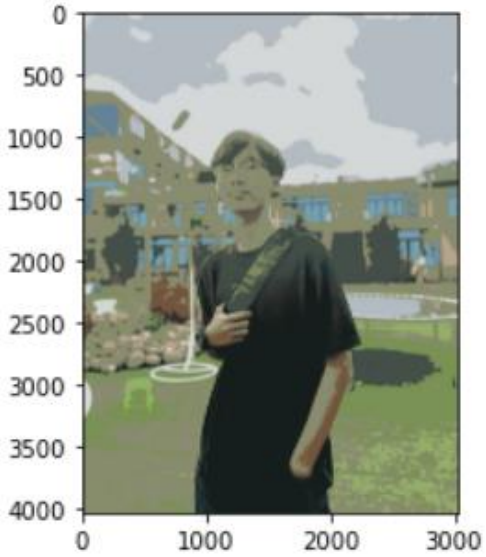
17 - Ảnh 4032x3024 sau xử lý, k = 5

```
Enter picture name: test.jpg
Number of k-clusters: 7
Image's size:  (4032, 3024, 3)
Number of k-clusters:  7
Enter picture name to save: test
Choose the format to save (PNG, JPG, PDF): png
```



18 - Ảnh 4032x3024 sau xử lý,, k = 7

```
Image's size:  (4032, 3024, 3)
Number of k-clusters:  30
Enter picture name to save: test
Choose the format to save (PNG, JPG, PDF): png
```



19- Ảnh 4032x3024 sau xử lý, k = 30

Mức độ hoàn thiện: 100%

V. Tài liệu tham khảo

Đồ án 1.

https://github.com/ThongLai/Kmeans-ColorCompression

Mã nguồn tham khảo.

https://anderfernandez.com/en/blog/kmeans-algorithm-python/

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

Tài liệu, kiến thức.

https://en.wikipedia.org/wiki/K-means_clustering

https://en.wikipedia.org/wiki/Image_segmentation

https://en.wikipedia.org/wiki/Mean_shift

Hàm và phép tính ma trận trong numpy.

https://stackoverflow.com/questions/33677183/subtracting-numpy-arrays-of-different-shape-efficiently

https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html

https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html

https://stackoverflow.com/questions/6430091/efficient-distance-calculation-between-n-points-and-a-reference-in-numpy-scipy

https://stackoverflow.com/questions/54269647/is-there-a-head-and-tail-method-for-numpy-array

https://stackoverflow.com/questions/1401712/how-can-the-euclidean-distance-be-calculated-with-numpy

https://stackoverflow.com/questions/69905398/substitute-row-in-numpy-if-a-condition-is-met

https://stackoverflow.com/questions/25823608/find-matching-rows-in-2-dimensional-numpy-array

Thao tác hình ảnh.

https://stackoverflow.com/questions/11137120/how-to-convert-an-image-from-one-format-to-another-with-python

https://www.delftstack.com/howto/numpy/save-numpy-array-as-image/