ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC KHOA HỌC TỰ NHIÊN THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN – CHẤT LƯỢNG CAO

ĐỒ ÁN THỰC HÀNH SỐ 1
TOÁN ỨNG DỤNG VÀ THỐNG KÊ – MTH00051

ĐỒ ÁN 01: COLOR COMPRESSION

THẦY (CÔ) HƯỚNG DẪN:

CÔ. PHAN THỊ PHƯƠNG UYÊN
THẦY. NGUYỄN NGỌC TOÀN

—o0o—

NGƯỜI THỰC HIỆN

HỌ VÀ TÊN: LÊ PHƯỚC PHÁT
MSSV: 22127322
LỚP: 22CLC10

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 06 NĂM 2024

MỤC LỤC

NHẬN XÉT CỦA GIÁO VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Ngày … tháng .., năm 2024
Giáo viên nhận xét

LỜI CAM KẾT

Tôi cam đoan rằng nghiên cứu này là do tôi thực hiện, dưới sự giám sát và hướng dẫn của các thầy cô bộ môn Toán Ứng Dụng và Thống Kê: cô **Phan Thị Phương Uyên** và thầy Nguyễn Ngọc Toàn. Kết quả của nghiên cứu này là hợp pháp và chưa được công bố ở bất kỳ hình thức nào trước đây. Tất cả các tài liệu được sử dụng trong nghiên cứu này được tôi thu thập bằng cách tự mình và từ các nguồn khác nhau, và được liệt kê đầy đủ trong phần tài liệu tham khảo. Ngoài ra, chúng tôi cũng sử dụng kết quả của một số tác giả và tổ chức khác. Tất cả đều được trích dẫn đúng đắn. Trong trường hợp có phạm bản quyền, chúng tôi chịu trách nhiệm cho hành động đó. Do đó, **Trường Đại học Khoa học Tự nhiên TP.HCM** không chịu trách nhiệm về bất kỳ vi phạm bản quyền nào được thực hiện trong nghiên cứu của tôi.

NỘI DUNG BÁO CÁO ĐỒ ÁN NGHIÊN CỨU

I. Ý tưởng thực hiện

a) Giới thiệu chung về yêu cầu đồ án nghiên cứu

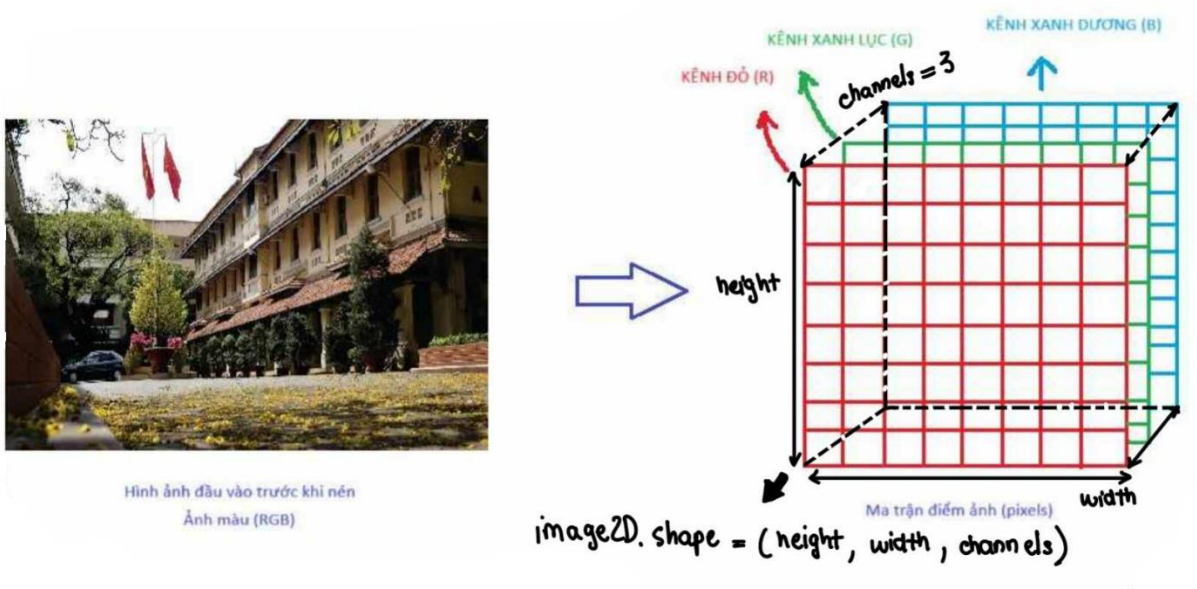
Trong **đồ án số 01 – Color Compression**, chúng ta sẽ thực hiện việc nén màu ảnh hay nói cách khác là giảm số lượng màu ảnh bằng cách sử dụng thuật toán **K – Means**.

Như chúng ta đã biết, mỗi hình ảnh có thể lưu trữ dưới ma trận của các điểm ảnh. Mỗi điểm ảnh biểu diễn một đơn vị cơ bản của hình ảnh, chứa thông tin về màu sắc hoặc độ sáng. Hiện nay, có rất nhiều loại ảnh được sử dụng trong các lĩnh vực khác nhau, từ xử lý ảnh kỹ thuật số đến ứng dụng trong nghệ thuật và khoa học như : ảnh đơn sắc (ảnh đen trắng / ảnh xám) , ảnh màu, ảnh chỉ số màu, ảnh độ sâu, …

Ảnh xám là loại ảnh chỉ chứa thông tin về độ sáng mà không có thông tin về màu sắc. Mỗi điểm ảnh trong ảnh xám được biểu diễn bằng một giá trị duy nhất, thường nằm trong khoảng từ 0 đến 255. Giá trị này đại diện cho mức độ sáng của điểm ảnh: 0 đại diện cho màu đen, 255 đại diện cho màu trắng, và giá trị trung gian (1 – 254) đại diện cho các mức xám giữa đen và trắng.

***Hình 1**. Ảnh xám của mathematical bridge* [1]

Ảnh màu chứa thông tin về màu sắc, mỗi **điểm ảnh** trong ảnh màu được đại diện bởi ba giá trị tương ứng với cường độ của ba màu cơ bản **RGB**, mỗi 1 kênh có chứa một màu: kênh màu đỏ R (Red), kênh màu xanh lục G (Green), kênh màu xanh dương B (Blue). Đây cũng chính là ba màu chính của ánh sáng khi tách ra từ lăng kính. Mỗi giá trị trong điểm ảnh màu cũng nằm trong khoảng từ 0 đến 255. Do đó, ta có 256 cách chọn cho R, 256 cách chọn G và 256 cách chọn B. Như vậy, số màu trong ảnh RGB có thể tạo ra là $256^3 \approx 1.7 \times 10^7$ màu.



Hình 2. *Ví dụ về ảnh màu RGB và tensor của nó.*

Tóm lại, với số lượng màu trong ảnh khá lớn, khi lưu trữ ảnh có thể sẽ tốn chi phí lưu trữ. Do đó, chúng ta cần phải tìm cách giảm số lượng màu trong ảnh sao cho nội dung biểu diễn của ảnh vẫn được bảo toàn nhất có thể. Đồng thời, việc giảm màu này giúp giảm chi phí tính toán, giảm overfitting, cải thiện khả năng tổng quát hóa, thích nghi với điều kiện ánh sáng thấp trong khi huấn luyện mô hình.

b) Giới thiệu chung về thuật toán K – Means

Đối với đồ án này, chúng ta sẽ sử dụng thuật toán **K – Means** để giảm số lượng màu cho ảnh.

Thuật toán **K – Means** là thuật toán phân cụm (clustering) thuộc nhóm các thuật toán học không giám sát (unsupervised learning) phổ biến trong học máy và phân tích dữ liệu. Nó được sử dụng để chia một tập dữ liệu thành k cụm khác nhau dựa trên các điểm dữ liệu gần nhau về mặt không gian.

Mục tiêu của thuật toán này là chia n điểm dữ liệu ban đầu thành k cụm dữ liệu sao cho tổng bình phương khoảng cách của mỗi điểm dữ liệu đến điểm trung tâm của cụm (centroid) là gần nhất. Tuy nhiên, do thuật toán này chúng ta không biết

nhãn (label) của từng điểm dữ liệu, nên chúng ta cần phải có phương pháp phân cụm hiệu quả.
Thuật toán K – Means hoạt động như sau:
Bước 01: Xác định input, tức là xác định tập dữ liệu X ban đầu có n điểm dữ liệu, k cụm cần phân chia cũng như cách tính khoảng cách hay cách chọn điểm dữ liệu làm trung tâm.
Bước 02: Khởi tạo. Chọn ngẫu nhiên k điểm dữ liệu làm tâm (centroids) của k cụm dữ liệu ban đầu.
Bước 03: Gán điểm dữ liệu vào các cụm.

- Tính khoảng cách giữa mỗi điểm dữ liệu và các điểm centroids. Phương pháp tính khoảng cách thường được tính bằng khoảng cách Euclidean.
- Gán các điểm vào các cụm sao cho khoảng cách từ mỗi điểm tới tâm là nhỏ nhất.

Bước 04: Cập nhật lại điểm trung tâm của từng cụm dữ liệu

- Tính trung bình các điểm dữ liệu trong mỗi cụm mới và cập nhật điểm centroid mới bằng với giá trị trung bình vừa tính được.
- Lặp lại bước gán cụm và cập nhật centroids cho đến khi các centroids của từng cụm không thay đổi hoặc sự thay đổi là rất nhỏ (dưới một ngưỡng nhất định).

Bước 05: Kiểm tra sự hội tu. Thuật toán sẽ dừng lại khi các điểm centroids không còn thay đổi hoặc có sự thay đổi rất ít giữa các lần lặp lại (iterations) hoặc số lần lặp lại đạt được sự tối đa (iteration ≥ max_iter).

c) Trình bày ý tưởng thực hiện

Từng bước thực hiện ý tưởng và quá trình thực hiện:

- **Bước 01**: Cho nhập đường dẫn hình ảnh cần giảm số lượng màu (img_path), số lượng k cụm cần phân chia (k_clusters), số lần hồi tụ tối đa (max_iter), kiểu khởi tạo các điểm centroids (centroids initialization type) như là các tham số cho hàm kmeans.
- **Bước 02**: Chúng ta sẽ dùng hàm đọc ảnh (img_2d) thông qua đường dẫn để xác định được shape của ảnh đầu vào như chiều dài (height), chiều rộng (width) của ma trận pixels và số lượng kênh màu của tấm ảnh.
- **Bước 03**: Chúng ta sẽ chuyển đổi ảnh đầu vào 2D có dạng (height, width, channels) thành ảnh 1D (height * width, channels) vì việc chuyển đổi này giúp chúng ta đơn giản hóa cấu trúc dữ liệu, làm cho các phép toán xử lý dễ dàng hơn. Đồng thời, do chúng ta đang sử dụng thuật toán K – Means để giảm số lượng màu mà thuật toán này lại yêu cầu dữ liệu là một tập hợp các điểm dữ liệu, mỗi điểm dữ liệu là một vector. Đối với ảnh mỗi pixel là một điểm dữ liệu và các kênh màu của nó tạo thành một vector đặc trưng (embedding) cho pixel đó.

- **Bước 04**: dùng thuật toán K – Means để gom k nhóm dữ liệu (gom nhóm k màu) với k là số cụm cần phân chia đã được nhập ở bước 01. Đồng thời, về cách hoạt động của thuật toán K – Means, tôi đã giới thiệu như phần giới thiệu chung về thuật toán K – Means đã giới thiệu phía trên hoặc sẽ được giải thích chi tiết hơn ở phần mô tả hàm gom nhóm màu sử dụng thuật toán K - Means.
- **Bước 05**: Sau khi chúng ta đã gom nhóm màu sử dụng K – Means, chúng ta cần phải tạo ảnh mới từ các màu trung tâm tìm được ở bước 04 bằng cách chuyển đổi ảnh 1D thành ảnh 2D.
- **Bước 06**: chúng ta sẽ nhập kiểu định dạng ảnh muốn lưu (1 – PNG, 2 – PDF, 3 – JPG) và lưu ảnh với đường dẫn mới.

II. Mô tả các hàm quan trọng trong đồ án

a) Import các thư viện liên quan

Đầu tiên, chúng ta cần phải cài đặt các thư viện liên quan trong quá trình thực hiện code:

```
import numpy as np  # import thu vien numpy (tinh toan ma tran)
from PIL import Image  # import thu vien pillow (doc, ghi anh)
import matplotlib.pyplot as plt  # import thu vien matplotlib (hien thi anh)
```

Hình 3. *Import các thư viện liên quan*

Trong đó:

- *Thư viện numpy*: dùng để tính toán các phép ma trận, vector, …
- *Thư viện pillow*: dùng để đọc ảnh và lưu ảnh.
- *Thư viện matplotlib*: dùng để hiển thị ảnh.

b) Hàm đọc ảnh thông qua đường dẫn ảnh

- **Tham số đầu vào:** img_path (đường dẫn ảnh) có kiểu string
- **Kết quả đầu ra:** ảnh dưới dạng mảng numpy 2D có dạng np.ndarray
- **Mô tả cách hoạt động:**

  Đối với hàm này, chúng ta cần kiểm tra các điều kiện tiên quyết trước:

  - Kiểm tra kiểu đường dẫn ảnh mà ta truyền vào có phù hợp hay không, tức là img_path có phải là kiểu string hay không.
  - Sau đó, ta kiểm tra xem file ảnh mà đường dẫn đưa tới có tồn tại hay không.

  Sau khi kiểm tra các điều kiện tiên quyết, chúng ta sẽ mở ảnh để đọc ảnh.

  - Sử dụng câu lệnh **‘try’** để xử lý bấy kỳ lỗi nào có thể xảy ra trong quá trình mở và đọc tệp ảnh.

- **'with Image.open(img_path) as img'**: mở tệp ảnh bằng thư viện Pillow (PIL). Sử dụng with statement để đảm bảo tệp được đóng đúng cách sau khi hoàn thành.

Cuối cùng, hàm trả về ảnh dưới dạng một mảng numpy 2D với kích thước (height, width, channels).

```python
# Hàm đọc ảnh thông qua đường dẫn ảnh cho sẵn
def read_img(img_path):
    """
    Read image from img_path
    Parameters
    ----------
    img_path : str
        Path of image
    Returns
    -------
        Image (2D)
    """
    # Kiểm tra xem kiểu đường dẫn ảnh đầu vào có phù hợp không.
    if not isinstance(img_path, str):
        raise ValueError('The provided path is not a string !!!')

    # Kiểm tra xem file có tồn tại hay không
    if not os.path.exists(img_path):
        raise FileNotFoundError('The specified image file does not exist !!!')

    try:
        # mở file ảnh để đọc
        with Image.open(img_path) as img:
            # sử dụng with statement để đảm bảo file được đóng đúng cách sau khi hoàn thành
            # type(img): <class 'PIL.JpegImagePlugin.JpegImageFile'>
            img_2d = np.array(img)  # chuyển đổi dạng list thành dạng numpy
    except Exception as e:
        raise IOError(f'An error occurred while reading the image: {e}') # hiển thị lỗi trong lúc đọc ảnh
    return img_2d  # trả về ma trận điểm ảnh dạng numpy 2D với img_2d.shape = (height, width, channels) với channels thường là 3.
```

Hình 4. *Hàm đọc ảnh thông qua đường dẫn ảnh cho sẵn*

c) Hàm hiển thị ảnh

- **Tham số đầu vào**: ảnh (img_2d) dưới dạng mảng numpy 2D có shape (height, width, channels)
- **Kết quả đầu ra**: hình ảnh được hiển thị bằng matplotlib
- **Mô tả cách hoạt động**

  Đối với hàm show_img(img_2d), nó có nhiệm vụ hiển thị ảnh bằng matplotlib.

  Đầu tiên ta sẽ kiểm tra ảnh img_2d có dạng np.ndarray có kiểu phù hợp hay không.

  Sau đó, ta sẽ cài đặt các tham số:

  - **plt.figure(figsize=(8, 8))** đặt kích thước của hình ảnh hiển thị để cải thiện trải nghiệm người dùng.
  - Thêm tiêu đề cho hình ảnh.
  - **plt.imshow(img_2d)** để hiển thị hình ảnh.
  - **plt.axis("off")** tắt trục để hình ảnh trông đẹp hơn.

- **plt.show()** để hiển thị đồ thị chứa hình ảnh.

```
# Ham hien thi anh
def show_img(img_2d):
    """
    Show image

    Parameters
    ----------
    img_2d : <your type>
        Image (2D)
    """
    # Kiểm tra xem img_2d có phải là mảng numpy (np.ndarray) hay không ?
    if not isinstance(img_2d, np.ndarray):
        raise ValueError("img_2d should be a numpy array")

    plt.figure(figsize=(8, 8))
    plt.title("Original Image")
    plt.imshow(img_2d)  # hiển thị hình ảnh 2D có shape là (height, width, channels)
    plt.axis("off")  # tắt trục => nhằm làm cho hình ảnh trông đẹp mắt hơn.
    plt.show()  # hiển thị đồ thị chứa hình ảnh
```

Hình 5. *Hàm hiển thị ảnh*

d) Hàm lưu ảnh vào đường dẫn ảnh mới

- **Tham số đầu vào**
  - Ảnh **img_2d** dưới dạng mảng numpy 2D có shape là (height, width, channels)
  - Đường dẫn ảnh mới **img_path** có dạng kiểu string.
- **Kết quả đầu ra**: hình ảnh được lưu vào đường dẫn mới.
- **Mô tả hoạt động**

  Đối với hàm **save_img(img_2d, img_path),** hàm sẽ giúp chúng ta lưu ảnh thông qua một đường dẫn sẵn đầu vào.

  Đầu tiên, chúng ta cần phải kiểm tra các tham số đầu vào có thỏa không
  - Kiểm tra xem img_2d có phải là mảng numpy (np.ndarray) hay không?
  - Kiểm tra xem img_path có phải là kiểu string hay không?

  Sau đó, kiểm tra xem thư mục đích có tồn tại hay không? Nếu không, tạo thư mục đó bằng **os.makedirs(directory)**.

  Tiếp theo, chúng ta sẽ chuyển đổi màng numpy thành một đối tượng **'Image'** của PIL và lưu nó bằng **'img.save(img_path)'**

```python
# Ham luu anh co shape (height, width, channels) va voi duong dan moi img_path
def save_img(img_2d, img_path):
    """
    Save image to img_path

    Parameters
    ----------
    img_2d : <your type>
        Image (2D)
    img_path : str
        Path of image
    """
    # Kiểm tra xem img_2d có phải là một mảng numpy hay không ?
    if not isinstance(img_2d, np.ndarray):
        raise ValueError('img_2d should be a numpy array')

    # Kiểm tra xem kiểu đường dẫn mới của ảnh có phù hợp không ?
    if not isinstance(img_path, str):
        raise ValueError("img_path should be a string")

    # Kiểm tra xem thư mục của đường dẫn có tồn tại hay không nếu không thì tạo thư mục đó
    directory = os.path.dirname(img_path)
    if not os.path.exists(directory):
        os.makedirs(directory)

    img = Image.fromarray(img_2d)  # Đọc ảnh từ np array
    img.save(img_path)
```

Hình 6. *Hàm lưu ảnh vào đường dẫn mới*

e) Hàm chuyển đổi ảnh 2D (height, width, channels) thành ảnh 1D (height × width, channels)

- **Tham số đầu vào**: ảnh img_2d dưới dạng một mảng numpy (np.ndarray) có shape(height, width, channels)
- **Kết quả đầu ra**: ảnh img_1d dưới dạng một mảng numpy (np.ndarray) có shape (height * width, channels)
- **Mô tả cách hoạt động**

  Đối với hàm convert_img_to_1d(img_2d), hàm này sẽ giúp chúng ta chuyển đổi ảnh 2D (height, width, channels) thành ảnh 1D (height * width, channels) giúp chúng ta dễ dàng tính toán và áp dụng cho thuật toán K – Means.

  Đầu tiên, chúng ta sẽ kiểm tra img_2d có phải là một mảng numpy (np.ndarray) hay không và img_2d có đủ 3 chiều là (height, width, channels).

Sau đó, chúng ta sẽ lấy ra các tham số chứa chiều cao height (= img_2d.shape[0]), chiều rộng width (= img_2d.shape[1]) và số kênh màu channels (= img_2d.shape[2]) của ảnh 2D.
Cuối cùng, ta sẽ chuyển đổi ảnh 2D thành ảnh 1D bằng 'img_2d.reshape(height * width, channels)' và trả về ảnh 1D dưới dạng mảng numpy có shape (height * width, channels).

```python
# Ham huyển đổi ảnh từ kích thước 2D (height, width, channels) sang 1D (height * width, channels)
def convert_img_to_1d(img_2d):
    """
    Convert 2D image to 1D image

    Parameters
    ----------
    img_2d : <your type>
        Image (2D)

    Returns
    -------
        Image (1D)
    """
    # Kiểm tra xem img_2d có phải là một mảng numpy và có đủ ba chiều (height, width, channels) không
    if not isinstance(img_2d, np.ndarray) or len(img_2d.shape) != 3:
        raise ValueError("img_2d should be a 3D numpy array with shape (height, width, channels)")

    height, width, channels = img_2d.shape  # lấy ra height, width, channels của ảnh 2D
    img_1d = img_2d.reshape(height * width, channels)  # chuyển đổi ảnh 2D thành ảnh 1D
    return img_1d
```

Hình 7. *Hàm chuyển đổi ảnh 2D sang ảnh 1D*

f) Hàm gom nhóm màu sử dụng thuật toán K – Means

- ❖ Tham số đầu vào:
  - img_1d: ảnh 1D có dạng mảng numpy (np.ndarray) shape (height * width, channels) sau khi chuyển đổi từ ảnh 2D thành ảnh 1D.
  - k_clusters: số lượng nhóm màu sau khi giảm.
  - max_iter: số lần hội tụ tối đa.
  - init_centroids: phương pháp để xác định các centroids cho thuật toán K – Means ('random' / 'in_pixels')
- ❖ Kết quả đầu ra:
  - centroids: mảng numpy (np.ndarray) các điểm trung tâm có **shape (k_clusters, num_channels).**
  - labels: mảng numpy (np.ndarray) có shape (height * width,) chứa các nhán của từng pixel trong hình ảnh gốc.
  - durations: thời gian thực thi thuật toán K – Means.
- ❖ Mô tả hoạt động

  Bước 01: Chọn ngẫu nhiên k điểm dữ liệu làm tâm (centroids) của k cụm dữ liệu ban đầu theo hai phương pháp: **'random'** (các điểm centroids sẽ

được khởi tạo với các giá trị ngẫu nhiên trong khoảng từ 0 đến 255) hay **'in_pixels'** (các điểm centroids sẽ được chọn ngẫu nhiên từ các điểm ảnh trong hình gốc cho mỗi kênh màu). Đồng thời, các điểm centroids này không được trùng nhau (replace = False) hay sẽ báo lỗi nếu không tìm thấy phương pháp chọn các điểm centroids

```
# Choose k random centroids
# length = img_1d.shape[0] = height * width
length = img_1d.shape[0]
# dim = img_1d.shape[1] = channels
dim = img_1d.shape[1]
if init_centroids == "random":
    centers = np.random.randint(0, 256, size=(k_clusters, dim))
elif init_centroids == "in_pixels":
    centers = img_1d[np.random.choice(length, size = k_clusters, replace=False)]

else:
    raise ValueError("Invalid value for init_centroids. Use 'random' or 'in_pixels'.")
```

***Hình 8**. Chọn k centroids ngẫu nhiên*

Bước 02: Tính khoảng cách của từng điểm ảnh đến từng centroid gần nhất

```
# Bước 01: tính khoảng cách của từng pixel đến centroid gần nhất
distances = np.linalg.norm(img_1d[:, np.newaxis, :] - centers, axis=2)
```

***Hình 9**. Tính khoảng cách của từng pixel đến centroid gần nhất*

Bước 03: Tạo mảng lables chứa nhãn của mỗi pixel (như là một vector chứa các giá trị của mỗi kênh màu) trong tập img_1d tương ứng với tâm có khoảng cách nhỏ nhất tới từng img_1d[i], nghĩa là tâm thứ j có khoảng cách ngắn nhất tới từng centroids thì labels[i] = j.

```
# Bước 02: tạo mảng labels chứa nhãn mỗi điểm trong tập img_1d
labels = np.argmin(distances, axis=1)
```

***Hình 10**. Tạo mảng labels chứa nhãn của mỗi pixels*

Bước 04: Cập nhật lại centroids mới của mỗi cụm bằng cách tính trung bình các điểm trong cụm.

- Dùng filter lấy ra tập các điểm của từng cụm.
- Tính tâm mới là giá trị trung bình của các điểm trong từng cụm

```
# Bước 03: Cập nhật centroids bằng np.mean
centers_new = np.array([np.mean(img_1d[labels == k], axis = 0) for k in range(k_clusters)])
```

***Hình 11**. Cập nhật lại centroids*

Bước 05: Kiểm tra điều kiện dừng hội tụ. Nếu không có tâm mới nào thay đổi: dừng vòng lặp, trả ra centers & lables, ngược lại: cập nhật lại tập centers = centers_new

```
# Bước 04: Kiểm tra điều kiện dừng
if np.array_equal(centers, centers_new):
    break

centers = centers_new
```

Hình 12. *Kiểm tra điều kiện dừng*

g) Hàm tạo ảnh mới từ các màu trung tâm tìm được thông qua gom nhóm màu sử dụng thuật toán K – Means

- **Tham số đầu vào**
  - img_2d_shape: kích thước hình ảnh đầu vào với dạng tuple (height, width, channels = 3)
  - centroids: mảng numpy (np.ndarray) các điểm trung tâm có **shape (k_clusters, num_channels)** được cập nhật ở từng cụm từ hàm gom nhóm màu sử dụng thuật toán K – Means với '**k_clusters'** là số lượng cụm và **'num_channels'** là số lượng kênh màu (trong trường hợp này là 3 cho RGB)
  - labels: mảng numpy (np.ndarray) có shape (height * width,) chứa các nhán của từng pixel trong hình ảnh gốc. Mỗi giá trị trong 'labels' là một số nguyên từ **0** đến **k_clusters – 1** chỉ ra cụm nào mà pixel đó thuộc về.
- **Kết quả đầu ra:** ảnh img_2d mới là một mảng numpy (np.ndarray) có shape (height, width, channels).
- **Mô tả hoạt động**

  Đối với hàm generate_2d_img(img_2d_shape, centroids, labels), hàm này có tác dụng tạo ảnh mới từ các màu trung tâm có từ hàm gom nhóm màu sử dụng thuật toán K – Means.

  Đầu tiên, lấy các thông số kích thước hình ảnh đầu ra chuyển đổi với chiều cao height = img_2d_shape.shape[0], chiều rộng width = img_2d_shape.shape[1] và số kênh màu channels = img_2d_shape.shape[2].

  Sau đó, chúng ta sẽ chuyển đổi nhãn màu sắc **'img_1d = centroids[labels].astype(np.uint8)'**

  - Lấy màu sắc trung bình từ 'centroids' dựa trên nhãn trong 'labels'. Nó sử dụng các giá trị trong 'labels' làm chỉ số để truy xuất vào centroids, do đó, mỗi giá trị trong 'labels' sẽ được thay thế bằng màu sắc trung bình tương ứng từ 'centroids'

- Chuyển đổi kết quả thành kiểu dữ liệu ‘unit8’, là kiểu dữ liệu chuẩn cho hình ảnh (số nguyên không dấu 8 – bit, có giá trị từ 0 đến 255)

Cuối cùng ta sẽ reshape thành hình dạng hình ảnh 2D bằng ‘img_2d = img_1d.reshape((height, width, channels))’ và trả về hình ảnh 2D được tạo ra từ quá trình trên trong đó, mỗi pixel được gán màu sắctừ các màu trung tâm cụm ‘centroids’ tương ứng với nhãn của pixel đó.

```
# Ham tạo ảnh mới từ các màu trung tâm có từ thuật toán k - Means
def generate_2d_img(img_2d_shape, centroids, labels):
    """
    Generate a 2D image based on K-means cluster centroids

    Parameters
    ----------
    img_2d_shape : tuple (height, width, 3)
        Shape of image
    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)

    Returns
    -------
        New image (2D)
    """

    height, width, channels = img_2d_shape  # img_2d_shape (height, width, channels)

    img_1d = centroids[labels].astype(np.uint8)

    img_2d = img_1d.reshape((height, width, channels)) # reshape hinh anh 1d thanh hinh anh 2d
    return img_2d
```

***Hình 13**. Hàm tạo ảnh mới từ các màu trung tâm*

h) Hàm main

Hàm này cho phép người dùng sẽ nhập đường dẫn của ảnh muốn giảm số lượng màu (img_path), số lần hội tụ tối đa (max_iter), số lượng nhóm màu (k_clusters) và phương pháp chọn các điểm centroids (người dùng chọn 0 cho phương pháp ‘random’ và chọn 1 cho phương pháp ‘in_pixels’). Hãy đảm bảo rằng người dùng phải nhập đúng đường ứng với cú pháp như sau: “C:/Users/Desktop/Path/…” chứ không được sử dụng “C:\Users\Desktop\Path…”

```
# Input parameters
img_path = input("Enter the path of the image (use C:/): ")
k_clusters = int(input("Enter the number of k clusters: "))
max_iter = int(input("Enter the maximum number of iterations: "))
centroidsT = int(input("Enter the initial centroids type (0 - random or 1 - in_pixels): "))

if centroidsT == 0:
    centroid_type = 'random'
elif centroidsT == 1:
    centroid_type = 'in_pixels'
else:
    print('Invalid centroid initialization method')
    return
```

***Hình 14**. Người dùng nhập các inputs*

Mở ảnh sử dụng hàm **read_img()** và chuyển đổi ảnh từ 2D sang 1D sử dụng hàm **convert_img_to_1d()**.

```
# Open and read the image
img_2d = read_img(img_path)
# Convert the 2D image into 1D image
img_1d = convert_img_to_1d(img_2d)
```

***Hình 15**. Mở ảnh và chuyển đổi ảnh*

Bắt đầu xử lý hình ảnh bằng thuật toán **K - Means** lấy tất cả các biến đầu vào của người dùng trước đó làm đối số. Sau đó, hình ảnh được tạo mới từ các điểm trung tâm và nhãn mà hàm kmeans() trả về.

```
# KMeans processing
centroids, labels, durations = kmeans(img_1d, k_clusters, max_iter, centroid_type)

print("Time processing: ", durations)

# Generate the new 2D image
new_img_2d = generate_2d_img(img_2d.shape, centroids, labels)
```

***Hình 16**. Quá trình xử lý và tái tạo ảnh*

Quá trình xử lý đường dẫn ảnh mà người dùng đã nhập vào. Ví dụ nếu đường đãn ảnh là ‘C:/Users/phatl/OneDrive - VNU-HCMUS/Desktop/22127322/images/hcmus.png’ thì output_img = ‘hcmus’ và directory_path = ‘C:/Users/phatl/OneDrive - VNU-HCMUS/Desktop/22127322/images/’

```
# Processing the image name
output_img = (img_path.split('/')[-1]).split('.')[0]
directory_path = img_path.replace('/' + img_path.split('/')[-1], '')
```

***Hình 17**. Xử lý đường dẫn ảnh*

Và cuối cùng, chúng ta sẽ nhập vào kiểu định dạng ảnh là PNG, JPG, PDF và dùng hàm save_img() để lưu ảnh đến đường dẫn ảnh mới. Ngoài ra chúng ta có thể dùng hàm show_img() để hiển thị ảnh mới lưu.

```
save_format = int(input('Enter output file type (0 - PNG, 1 - JPG, 2 - PDF): '))

if save_format == 0:
    save_path = f"{directory_path}/{output_img}_{centroid_type}_{k_clusters}_generated.png"
elif save_format == 1:
    save_path = f"{directory_path}/{output_img}_{centroid_type}_{k_clusters}_generated.jpg"
elif save_format == 2:
    save_path = f"{directory_path}/{output_img}_{centroid_type}_{k_clusters}_generated.pdf"
else:
    print('Invalid save format.')
    return

# Save the generated image
save_img(new_img_2d, save_path)
print(f"New image saved successfully at {save_path}")

# show_img
show_img(new_img_2d)
```

***Hình 18**. Nhập định dạng ảnh và lưu ảnh*

i) Hàm testing

Chúng ta có thể dùng hàm này để kiểm tra việc giảm số lượng màu thông qua thuật toán k – means với số lượng màu là k = {3, 5, 7} và với hai phương pháp chọn các điểm centroids là ‘random’ hay ‘in_pixels’.

```
# Hàm testing
def testing(img_path, max_iter, initial_centroids):
    output_imgs = []
    titles = ['Original Image']
    k_values = [3, 5, 7]

    img_2d = read_img(img_path)
    output_imgs.append(img_2d)
    img_1d = convert_img_to_1d(img_2d)

    for k in k_values:
        centroids, labels, durations= kmeans(img_1d, k, max_iter, initial_centroids)
        new_img_2d = generate_2d_img(img_2d.shape, centroids, labels)
        output_imgs.append(new_img_2d)
        titles.append(f'{initial_centroids} with k = {k} \nTime: {durations:.4f} seconds')

    show_multi_testing_pics(output_imgs, titles)
```

***Hình 19**. Hàm testing*

j) Hàm show_multi_testing_pics

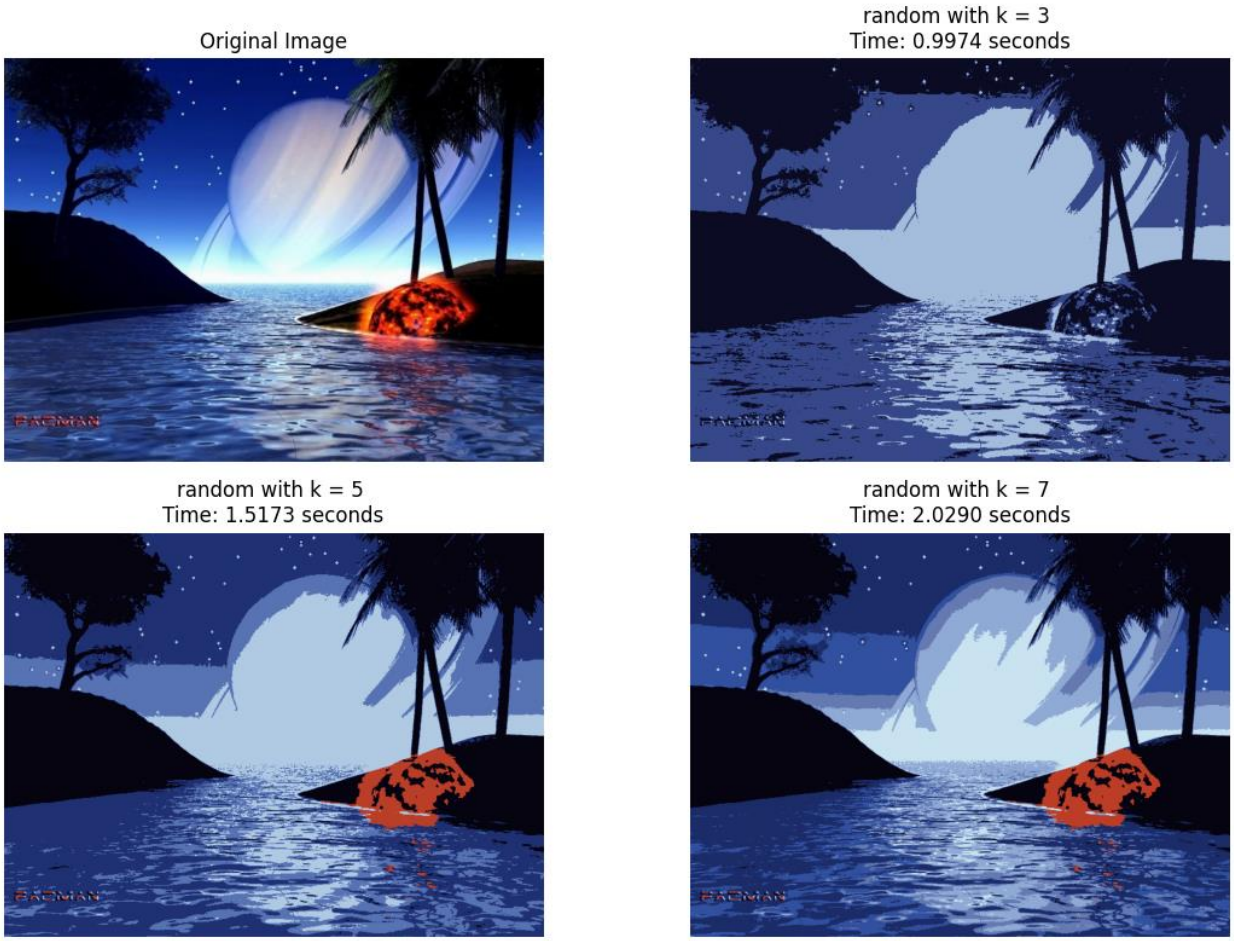
```python
# Hàm in hình trong trường hợp test k = {3, 5, 7} (gồm 4 hình)
def show_multi_testing_pics(output_imgs, titles):
    fig, axes = plt.subplots(2, 2, figsize=(12, 8))
    axes = axes.flatten()
    for ax, img, title in zip(axes, output_imgs, titles):
        ax.set_title(title)
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

***Hình 20**. Hàm in hình trong trường hợp test k = {3, 5, 7}*

III. Kết quả nghiên cứu đồ án

a) Hình ảnh kết quả với từng trường hợp

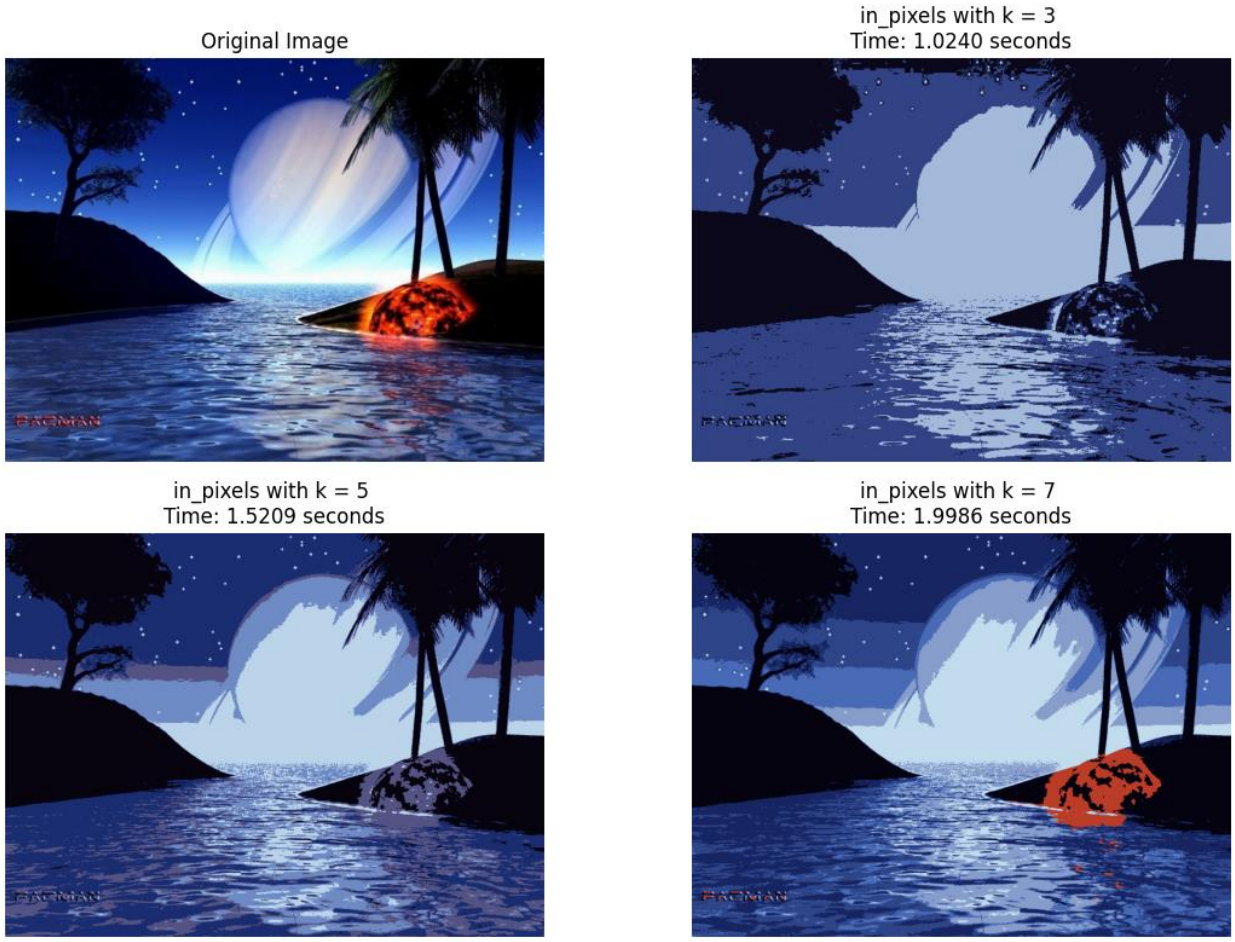
- Kết quả hình ảnh tương ứng với từng số lượng màu **k = {3; 5; 7}** sử dụng phương pháp **"Random"** các điểm centroids



***Hình 21**. Testing program with k = {3, 5, 7} and "random" centroids initialization*

k	Tổng thời gian chạy (10 iteration)	Thời gian trung bình / iteration
k = 3	0.9974 (s)	0.09974 (s)
k = 5	1.5173 (s)	0.15173 (s)
k = 7	2.0290 (s)	0.20290 (s)

- Kết quả hình ảnh tương ứng với từng số lượng màu **k = {3; 5; 7}** sử dụng phương pháp **"In_pixels"** các điểm centroids.



***Hình 22**. Testing program with k = {3, 5, 7} and "in_pixels" centroids initialization*

k	Tổng thời gian chạy (10 iteration)	Thời gian trung bình / iteration
k = 3	1.0240 (s)	0.10240 (s)
k = 5	1.5209 (s)	0.15209 (s)
k = 7	1.9986 (s)	0.19986 (s)

b) Nhận xét kết quả nghiên cứu

- **Thời gian chạy:** Nếu k càng lớn thì tổng thời gian thực thi càng lớn dẫn đến thời gian thực thi trung bình trên mỗi lần hội tụ càng lớn. Lý do là do sự tăng lên của số lượng phép tính khoảng cách và thời gian cần thiết để cập nhật các tâm cụm. vì vậy chúng ta cần phải chọn k cho hợp lý để đảm bảo phân cụm hiệu quả và giảm thiểu thời gian tính toán.
- **Kích thước ảnh:** Kích thước ảnh càng lớn (độ phân giải càng lớn) thì tốc độ thực thi càng lâu vì số lượng điểm ảnh quá nhiều làm tăng số lượng phép tính khoảng cách và giá trị trung bình.
- **Chất lượng hình ảnh:** Với số lượng màu k càng lớn thì hình ảnh sẽ càng chất lượng hơn và giữ được nhiều chi tiết hơn so với số lượng màu k nhỏ. Tuy nhiên, với số lượng màu k nhỏ vẫn có thể giữ lại được bố cục cơ bản nhất của hình ảnh.
- Nếu ta sử dụng phương pháp ‘random’ thì k càng lớn thì thời gian càng lâu tuy nhiên nếu sử dụng phương pháp ‘in_pixels’ thì k càng lớn thì thời gian thực thị sẽ bé hơn so với phương pháp’randoms’

TÀI LIỆU THAM KHẢO

Trong quá trình thực hiện **đồ án nghiên cứu 01 – Color Compression**, tôi đã sử dụng và tham khảo một số tài liệu sau đây:

[1] Deep Learning cơ bản – Bài 05: Giới thiệu về xử lý ảnh. *(ngày truy cập: 15/06/2024)*

[2] K – means clustering *(ngày truy cập: 17/06/2024)*

[3] Bài lab02 – project01.ipynb của cô Phan Thị Phương Uyên *(ngày truy cập: 13/06/2024)*

[4] Bài 4: K – means clustering – Machine Learning Cơ bản của Vũ Khắc Tiệp *(ngày truy cập 14/06/2024)*