UNIVERSITY OF SCIENCE
FALCUTY OF INFORMATION TECHNOLOGY

SUBJECT: Applied Mathematics and Statistics

PROJECT 1: COLOR COMPRESSION

Name: Nguyễn Quốc Huy
Class: 20CLC02
Student-ID: 20127188

Lecturers: **VŨ QUỐC HOÀNG**
NGUYỄN VĂN QUANG HUY
TRẦN THỊ THẢO NHI
PHAN THỊ PHƯƠNG UYÊN

Table of Contents

I) Tổng quan đồ án

1. Giới thiệu đồ án

- 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, …
- Ảnh màu được sử dụng phổ biến là ảnh RGB, trong đó, mỗi điểm ảnh sẽ lưu trữ 3 thông tin kênh màu (mỗi kênh màu 1 byte) là: R (red - đỏ), G (green - xanh lá), B (blue - xanh dương)
- Như vậy, số màu trong ảnh RGB có thể là 256^3. Với số lượng màu khá lớn, khi lưu trữ ảnh có thể sẽ tốn chi phí lưu trữ. Do đó bài toán đặt ra là giảm số lượng màu để biểu diễn ảnh sao cho nội dung ảnh được bảo toàn nhất có thể.
- 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 (R^3) và chọn ra đại diện cho từng nhóm. Như vậy, bài toán trên trở thành gom nhóm các vec-tơ.

2. Yêu cầu đồ án

- Trong đồ án này, chúng ta phải thực hiện cài đặt chương trình giảm số lượng màu cho ảnh sử dụng thuật toán K-Means.
- Các thư viện được phép sử dụng là: NumPy (tính toán ma trận), PTL (đọc, ghi ảnh), matplotlib (hiển thị ảnh).

II) Chi tiết đồ án

1. Môi trường làm việc

- Ngôn ngữ: Python (version 3.9.6)
- Biên dịch: VSCode

2. Ý tưởng

- Đầu tiên người dùng sẽ nhập tên file cần được giảm màu (Cấu trúc nhập là tenfile.duoifile ví dụ abc.png, đuôi file là png hoặc jpg/jpeg tùy ý) sau đó sẽ nhập đến dữ liệu cần xuất (ở đây có 3 kiểu dữ liệu có thể xuất đó là png/jpg/pdf)
- Chạy vòng lặp với 3 cluster chính là 3 5 7, tính toán hàm kmeans để giảm hóa màu sắc trong ảnh. Trong hàm gồm có 2 kiểu giảm hóa chính là “random” và “in-pixels”.
- Chạy vòng lặp while với số max_iter mà người dùng sẽ nhập vào (ở đây em chọn 30 vì đây là con số ổn định ở máy em với hình 200 x 200px), trong vòng lặp chúng ta sẽ tính toán khoảng cách từ điểm ảnh đến centroids, tìm ra khoảng cách nhỏ nhất từ điểm đó đến centroids gần nhất và gán label cho nó để nó biết mình thuộc về nhóm màu nào, sau đó tính toán lại centroids bằng các vector trung bình và cứ như thế lặp lại vòng lặp
- Sau mỗi vòng lặp max_iter sẽ trừ đi 1 đến khi nào < 0 sẽ thoát vòng lặp và trả về 2 giá trị centroids và labels

3. Mô tả các hàm sử dụng

Bước 1: Import thư viện:

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

Ở đây em dùng 3 thư viện chính là matplotlib (dùng để hiển thị hình ảnh), numpy (dùng để tính toán, chuyển hóa các ma trận) và PIL (dùng để mở ảnh và convert về dạng "RGB")

Bước 2: Thuật toán Kmeans

- Các bước thực hiện chi tiết thuật toán:

1. Các thông số trong đề bài:
   - **img_1d**: Ma trận 1 chiều chứa các pixel ảnh
   - **k_clusters**: Số lượng màu để phân nhóm
   - **max_iter**: Số lần lặp trong việc phân nhóm
   - **init_centroids**: Chọn kiểu centroid, ở đây chúng ta có 2 kiểu là random và in-pixels
   - **Hai thông cần trả về cuối cùng của hàm kmeans là centroids và labels**
2. Giải thích các hàm:

Đầu tiên là kiểu centroids là random

```
if init_centroids == 'random':
    centroids = np.random.randint(255, size = (k_clusters, img_1d.shape[1]))
```

   - Do đây là kiểu random nên em sẽ dùng hàm randint trong thư viện numpy để có thể random ra một ma trận theo ý mình
   - 255 nghĩa là random từ 0 -> 255
   - k_clusters nghĩa là số lượng ma trận được in ra sẽ bằng số k
   - img_1d.shape[1] nghĩa là kích thước của ma trận sẽ bằng với số của giá trị của img_1d.shape[1]
   - Ví dụ: centroids = np.random.randint(255, size = (3, 2)) thì kết quả trả về là [254 255] [123 245] [255 255]

Tiếp theo là kiểu in-pixels

```
elif init_centroids == 'in_pixels':
    for i in range(k_clusters):
        list = []
        for _ in range(len(img_1d[0])):
            centroid_random = np.random.randint(0, 255)
            list.append(centroid_random)
        centroids.append(list)
```

   - Về cơn bản thì hàm in-pixels cũng sẽ giống với hàm random, em sẽ chạy vòng lặp để lấy ra k-cluster màu sau đó sẽ đưa vào list và từ list em sẽ truyền vào hàm centroids

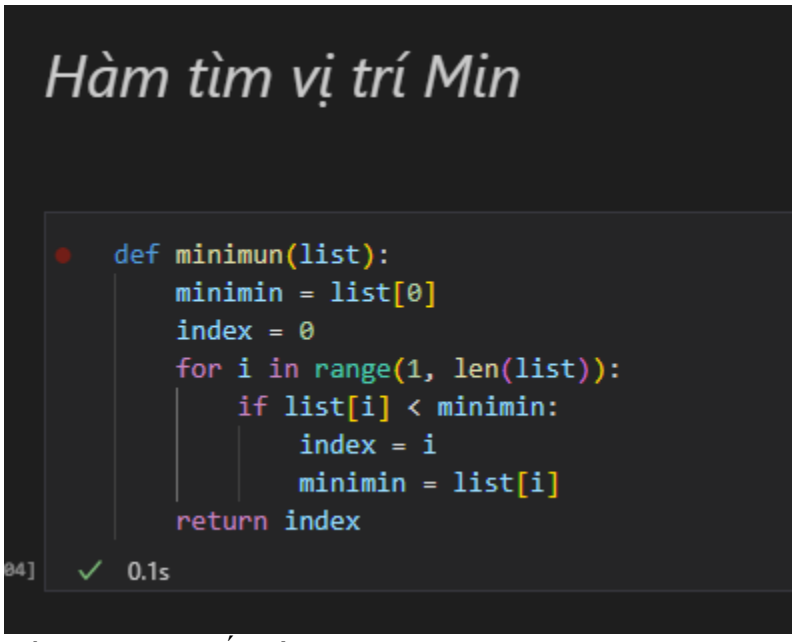
```
distance = []
for i in img_1d:
    dist = []
    for j in centroids:
        dist.append(np.linalg.norm(j - i))
    distance.append(dist)
```

- Ở đây em sẽ dùng hàm np.linalg.norm trong numpy để tính khoảng cách giữa các điểm ảnh và centroids

```
labels = []

for i in range(len(distance)):
    index = minimun(distance[i])
    labels.append(index)
```

- Sau khi có được khoảng cách thì em sẽ dùng minimun là một hàm em viết thêm bên ngoài để có thể trả về vị trí mà giá trị nhỏ nhất, từ đó truyền các giá trị vào trong labels



Hàm minimun viết thêm

```
while max_iter > 0:
    new_centroids = []
    for i in range(len(centroids)):
        new_centroids += [np.mean(img_1d[np.array(labels) == i], axis = 0)]
    distance = []
    for i in img_1d:
        dist = []
        for j in new_centroids:
            dist.append(np.linalg.norm(j - i))
        distance.append(dist)

    labels = []
    for i in range(len(distance)):
        index = minimun(distance[i])
        labels.append(index)
    max_iter -= 1

return new_centroids, labels
```

Ở đây em sẽ tạo ra một mảng new_centroids rỗng sau đó sẽ dùng hàm np.mean trong numpy để tính trung bình cộng các vector với nhau. Sau khi có trung bình cộng em sẽ tính lại labels để có thể phân nhóm cho các điểm mới. Sau mỗi lần phân nhóm các điểm ảnh xong thì giá trị max_iter sẽ giảm một (Giá trị max_iter càng cao thì phân nhóm màu càng chính xác nhưng sẽ đòi hỏi nhiều thời gian hơn để chạy)

Cuối cùng là hàm Pixels và hàm Random

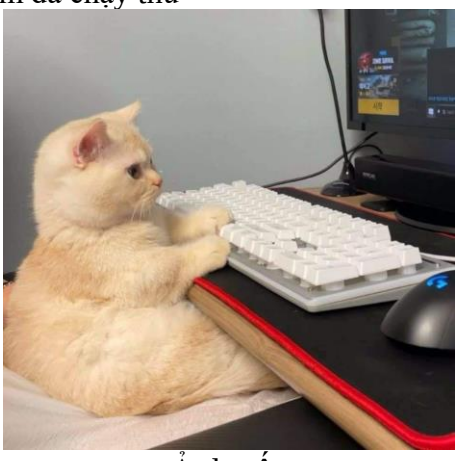
```
def Pixels():
    file = input("Nhap ten file: ")
    file_type = input("Kieu du lieu can xuat: ")
    k_clusters = [3, 5, 7]
    for i in range(len(k_clusters)):
        image = Image.open(file).convert("RGB")
        image = np.array(image)
        height, width, channel = image.shape[0], image.shape[1], image.shape[2]
        image = image.reshape(height * width, channel)
        centroids, labels = kmeans(image, k_clusters[i], 30, 'in_pixels')
        for j in range(image.shape[0]):
            image[j] = centroids[labels[j]]
        data = image.reshape(height, width, channel)
        plt.imshow(data)
        plt.imsave("cluster - " + str(k_clusters[i]) + " - in-pixels" + "." + file_type, data)
```

- Ở đây em viết 2 hàm nhưng về cơ bản thì nó sẽ y chang nhau chỉ khác lúc truyền thông số vào hàm kmeans thì thay vì em truyền 'in_pixels' thì trong hàm random em sẽ truyền vào 'random'

- Đầu tiên em sẽ dùng hàm input() để có thể lấy được file và kiểu dữ liệu mà người dùng muốn xuất ra
- Sau đó em chạy vòng lặp với 3 cluster tương ứng là 3, 5, 7
- Đầu tiên trong vòng lặp em sẽ dùng hàm Image.open() trong PIL và hàm convert("RGB") để có thể đưa ảnh về dạng RGB thông thường (vì theo em biết nếu đuôi file là dạng png thì ảnh sẽ là dạng RGBA)
- Sau đó em sẽ chuyển hình ảnh đọc được thành các vector thông qua hàm np.array() trong numpy
- Em lưu các thông số height, width, num_channels để xuống dưới có thể reshape (thư viện PIL) thành mảng 1 chiều để có thể truyền vào kmeans
- Sau khi đã chạy xong hàm kmeans em sẽ có 2 giá trị centroids và labels, em sẽ gắn 2 thông số này vào ma trận ảnh
- Do ảnh lúc này đang là ma trận 1 chiều nên em lại dùng hàm reshape để chuyển về dạng ảnh như lúc ban đầu
- Hàm imshow (thư viện matplotlib.pyplot) dùng để hiển thị hình ảnh và hàm imsave dùng để lưu hình ảnh

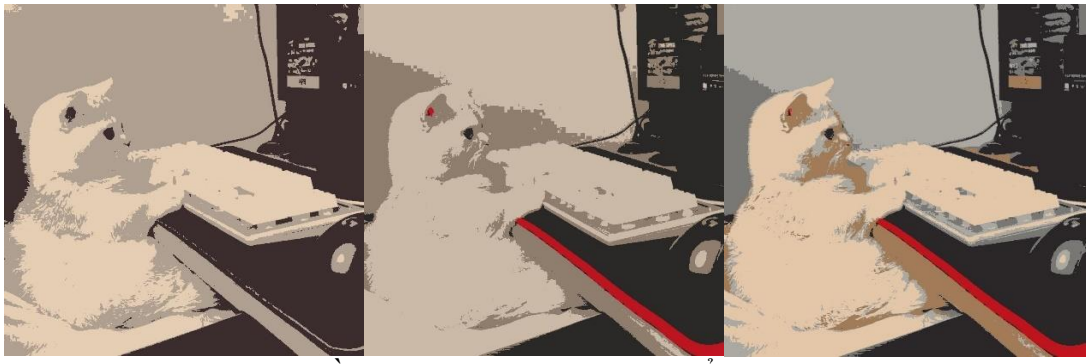
III) Phân tích và Nhận xét

Sau đây là hình ảnh mà em đã chạy thử



Ảnh gốc

Từ trái sang phải lần lượt là k = 3, 5, 7 với kiểu centroids là in-pixels



Từ trái sang phải lần lượt là k = 3, 5, 7 với kiểu centroids là in-pixels

- Nhận xét về kiểu in-pixels: Màu của kiểu in-pixels em thấy là rõ ràng và đẹp hơn kiểu random do một phần là hàm này chọn màu là trong ảnh luôn nên là bức hình ra sẽ gần giống với hình gốc hơn là random
- Nhận xét về kiểu random: Màu của random sẽ không thể đẹp bằng in-pixels nhưng được cái là em thấy nó chạy nhanh, nhanh là nhanh hơn in-pixels một tý vì em không biết là do máy em hay do vấn đề gì mà cả hai hàm đều chạy gần bằng nhau về thời gian. Nhưng nhìn chung là random vẫn nhanh. Một vấn đề nữa em không biết là do đôi khi nó random ra một màu nào đó lỗi hay xuất ra một giá trị không hợp lệ mà ảnh của em khi trả về đôi khi sẽ bị một màu (tương đương k = 1) nhưng em chạy lại lần 2 thì không bị
- Nhận xét chung: Em thấy 2 hàm này dùng khá tốt, mặc dù máy em chạy hơi chậm với các hình kích thước lớn, nhưng qua đồ án này em lại học thêm được nhiều điều mới, về cách xử lý ảnh, phân nhóm label, giảm thiểu số màu trong bức ảnh để tạo ra một hình ảnh nhỏ về dung lượng nhưng vẫn giữ nguyên về tổng thể. Em cảm thấy rất vui vì sau những giờ học tập căng thẳng trên lớp giờ đây em còn có thể căng thẳng hơn với deadline này.

IV) Nguồn tham khảo

- Định nghĩa k-means: https://en.wikipedia.org/wiki/K-means_clustering
- Hàm tính trung bình cộng: https://numpy.org/doc/stable/reference/generated/numpy.mean.html
- Hàm randint: https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html
- Hàm Imsave: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imsave.html
- Hàm norm: https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html
- Tham khảo ý tưởng: https://github.com/t3bol90/ST-MA-Lab02/blob/master/src/18127231.ipynb