UNIVERSITY OF SCIENCE
FALCUTY OF INFORMATION TECHNOLOGY

SUBJECT: APPLIED MATH AND STATISTICS

PROJECT 1: COLOR COMPRESSION
CLASS 20CLC11

Student: Mai Quý Trung
ID: 20127370

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

Table of Contents

I) Tổng quan

1. Giới thiệu đồ án

- 1 bức ảnh có thể lưu trữ dưới ma trận của các điểm ảnh.
- Có nhiều loại ảnh được sử dụng trong thực tế: ảnh xám, ảnh màu, …
- Ảnh màu tồn tại dưới nhiều dạng màu (màu rgb, rgba …) hoặc nhiều định dạng khác nhau (file png, jpg …)
- Phổ biến nhất là ảnh RGB, trong đó mỗi điểm ảnh sẽ lưu trữ 3 thông tin kênh màu (mỗi kênh màu là 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 tương đương với 17 triệu màu.
- Với số lượng màu khá lớn được nêu trên, 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 và chọn ra đại diện cho từng nhóm. Như vậy, bài toán trên trở thành gom nhóm các vector pixel màu.

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 (K-Means clustering algorithm).
- 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 sử dụng: Python (version 3.10.4)
- Text edit và trình biên dịch: Jupyter Notebook
- Source code: 20127370.ipynb

2. Ý tưởng

- Tổng quát:
  + Chúng ta sẽ cho người dùng nhập vào file ảnh và định dạng file muốn xuất (png/pdf), sau đó từ ảnh raw, ta dùng hàm **imread** từ thư viện **matplotlib** để chuyển đổi từ file raw sang ma trận 2 chiều chứa các pixel màu gồm 1 mảng các màu **rgb** (ảnh png) hoặc **rgba** (ảnh jpg/jpeg).
  + Tiếp theo, chúng ta sẽ sử dụng hàm **k_means_clustering_algorithm** (key function) để lấy được 2 thông số là **centroids** và **labels** sẽ được giải thích kĩ hơn ở phần sau.

+ Cuối cùng, ảnh sẽ được apply vào **centroids** và **labels** từ thuật toán k means và xuất ra định dạng file từ input người dùng, kết quả thu được ảnh với số lượng cluster theo thứ tự: 7, 5, 3.

- Hàm K-Means: Chúng ta sẽ viết lại thuật toán K Means (K Means clustering algorithm) theo quy tắc 'random' hoặc 'in-pixels'. Cơ chế của thuật toán này là ta sẽ chọn ra 1 số điểm đại diện cho các group (hay các cluster) và đó cũng chính là số lượng màu chúng ta muốn bức ảnh được compress. Sau đó thuật toán sẽ tính toán khoảng cách và gom nhóm các pixel màu đến centroid mà gần nó nhất. Các centroid sau đó được update đến 1 vị trí gần đó để có thể gom nhóm lại tốt hơn, quá trình

3. Chi tiết các hàm

Bước 1: Import thư viện

- Em sử dụng các thư viện như sau:

+ **matplotlib**: dùng **plt** để xuất ảnh dưới dạng figure và lưu lại dưới dạng png/pdf, dùng **img** để import ảnh và chuyển ảnh thành ma trận các pixel màu

+ **numpy**: thư viện dùng để xử lí các thông tin trên các ma trận n chiều và random choice

scipy: dùng cdist để tính khoảng cách gần nhất từ các điểm ảnh với các centroid sử dụng thuật toán **euclidean**

Bước 2: Viết thuật toán K Means

- Các bước thực hiện chi tiết thuật toán:

1. Các thông số truyền vào:

+ **img_1d**: ma trận 1 chiều các pixel màu của ảnh

+ **k_clusters**: số lượng màu mong muốn (số lượng cluster để phân nhóm)

+ **max_iter**: giá trị tối đa số lần lặp lại các bước phân nhóm

+ **init_centroids**: chọn centroid theo 1 trong 2 kiểu **random** hoặc **in-pixels** (default là `random`)

2. Giá trị trả về: Hàm sẽ trả về các giá trị sau đây

+ **centroid**: mảng chứa số lượng màu tương ứng với số nhóm được phân chia (cluster)

+ **labels**: mảng ma trận các pixel màu với mỗi pixel được dán nhãn nhóm mà pixel đó thuộc về

3. Chi tiết hàm:

+ Đầu tiên, tạo biến centroids là mảng trống, với centroid sẽ là các pixel màu đại diện cho từng group (cluster)

+ Tiếp theo, nếu **init_centroids** là **random**: tạo biến check = False, rồi sẽ chọn vị trí bắt đầu (k_index) và chọn ra **k_clusters** màu cho mảng centroids. Sau đó centroids sẽ được gán các pixel màu đó từ **img_1d**

+ Nếu **init_centroids** là **in-pixels**: sẽ lấy **k_clusters** màu bằng cách random từng màu R, G và B cho mỗi pixel, nếu ảnh là màu rgb thì centroids sẽ được random 3 pixel màu, nếu là rgba thì 4 màu

+ Sau khi có được mảng centroids, ta sẽ tính khoảng cách từ mỗi pixel trong **img_1d** đến các centroids sử dụng hàm **cdist** với thuật toán **euclidean**

+ Với khoảng cách đến các centroids của từng pixel màu, ta sẽ thực hiện việc nhán dãn (label) từng pixel màu với centroid gần với pixel đó nhất bằng hàm **argmin** của thư viện **numpy**

+ Sau khi label các pixel màu, ta sẽ cho chạy vòng lặp với số lần lặp là **max_iter**, gán centroids lại bằng mảng rỗng, với số lượng centroid và vị trí của chúng ban đầu, ta sẽ thiết lập vị trí mới của chúng để có thể tập trung các pixel màu chính xác hơn bằng hàm **mean** tính trung bình của các pixel xung quanh cluster tương ứng

+ Các vị trí centroid sau đó được update và ta sẽ thực hiện lại công việc tính khoảng cách đến các centroid và label từng pixel màu, cho đến hết số lần **max_iter**

+ Ngoài ra, với max_iter khá lớn, khi chạy vòng while, đến 1 lúc nào đó, các centroids sẽ không thay đổi vị trí nữa (hoặc đổi vị trí rất gần vị trí cũ), nên với các ảnh thuộc loại 'in-pixels', ta sẽ check xem vị trí centroid mới có trùng với vị trí centroid cũ hay không, nếu chúng gần nhau, ta sẽ trả về centroids và labels và kết thúc hàm.

- **Lưu ý:** *max_iter càng lớn, độ chính xác phân vùng các nhóm màu càng cao nhưng sẽ đi đôi với việc hàm sẽ chạy lâu hơn*

Bước 3: Hàm khởi tạo các giá trị ban đầu

- Hàm dưới đây sẽ hoạt động như sau:
  + Cho người dùng nhập vào tên file
  + Cho người dùng nhập vào định dạng file muốn xuất (png/pdf)
  + Sau đó hàm sẽ xử lí chuỗi lấy tên của file (không chưa extension) và định dạng file muốn xuất

Bước 4: Hàm xử lí ảnh, chuyển đổi từ file raw sang ma trận các pixel màu

- Các bước hàm thực hiện như sau:
  + Hàm nhận input argument là filename từ người dùng nhập vào
  + Sau đó sử dụng thư viện **matplotlib.image** hàm **imread** để chuyển ảnh sang ma trận 3 chiều: chiều dài, chiều rộng và mỗi điểm ảnh là 1 ma trận gồm 3 hoặc 4 phần tử (3 cho màu **rgb** và 4 cho màu **rgba**)
  + Chuyển đổi từ ma trận 2 chiều với các mảng màu rgb/rgba sang ma trận 1 chiều với hàm **reshape**
  + Sau đó lưu các thông số chiều cao, chiều rộng và số lượng màu của ảnh vào và trả về các giá trị đó

Bước 5: Hàm main, sử dụng K means với centroid ban đầu là 'random' hoặc 'in-pixels'

- Các bước thực hiện:

+ Lấy filename và định dạng file muốn xuất từ hàm **initialize** được nêu trên
+ Sau đó khởi tạo mảng với số lượng cluster theo thứ tự giảm dần: 7, 5, 3
+ Chạy vòng lặp for trong mảng cluster
+ Lấy ma trận 3 chiều các pixel màu, chiều cao, chiều rộng và loại màu từ hàm **handling_picture**
+ Sau đó dùng thuật toán **k_means_clustering_algorithm** với số lượng cluster tương ứng, 'max_iter = 30' và 'init_centroid = random'
+ Hàm trả về centroids và labels
+ Sau đó gán lại các giá trị màu từ centroids và labels vào ma trận ảnh
+ Lúc này ma trận ảnh đang là ma trận 1 chiều, ta thực hiện hàm **reshape** để đưa về ma trận 2 chiều ban đầu
+ Ảnh sau đó được thư viện **plt** thực hiện các hàm **imshow** và **show** để xuất ảnh ra figure và hàm **imsave** để lưu figure với định dạng tương ứng
+ Chúng ta sẽ thực hiện tương tự với 'init_centroid = in-pixels'

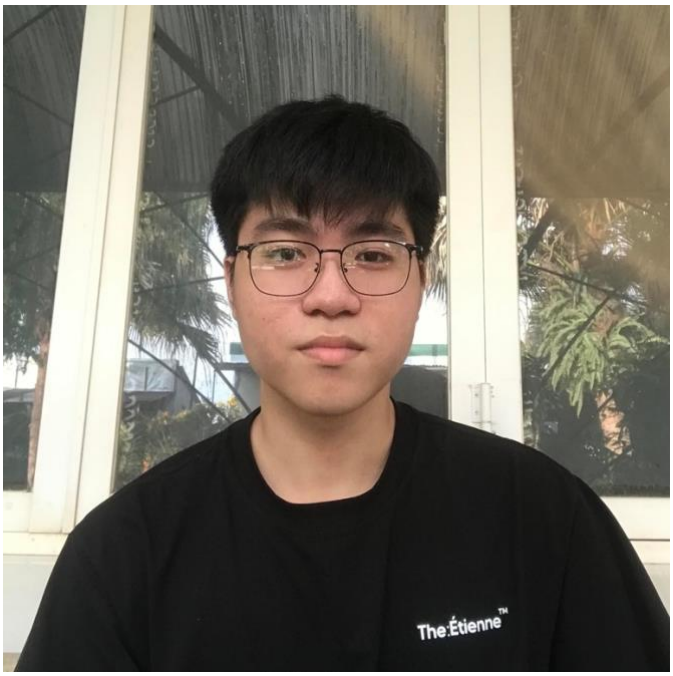
Bước 6: Gọi hàm main và thực thi chương trình

III) Phân tích và thống kê

1. Thử nghiệm và kết quả

Chúng ta sẽ thử nghiệm 1 số ảnh với các định dạng png, jpg, jpeg trên 2 kiểu centroid là 'random' và 'in-pixels' với số các cluster tương ứng: 7, 5, 3.

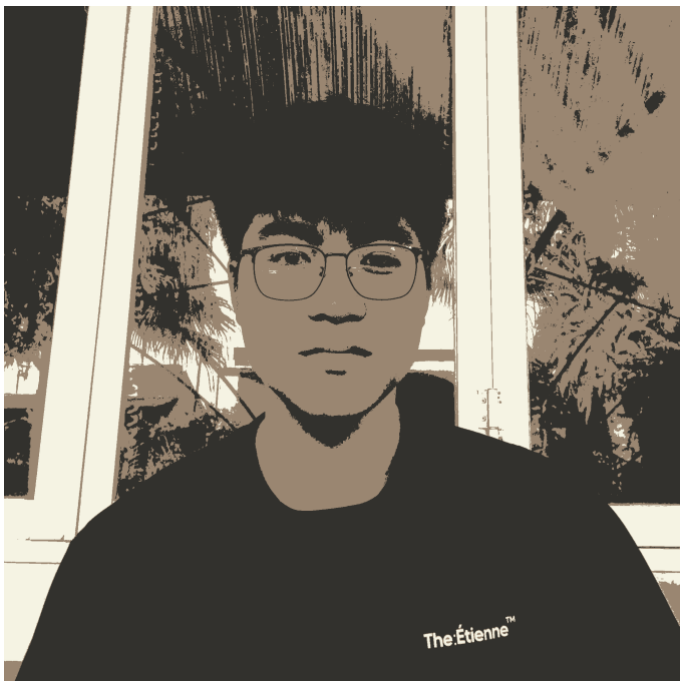
- Centroid ban đầu là 'random'
  + Ảnh 1:



- Cluster = 7

- Cluster = 5

- Cluster = 3



+ Ảnh 2:

- Cluster = 7

- Cluster = 5

- Cluster = 3

- Centroid ban đầu là ‘in-pixels’
  + Ảnh 1:

- Cluster = 7

- Cluster = 5

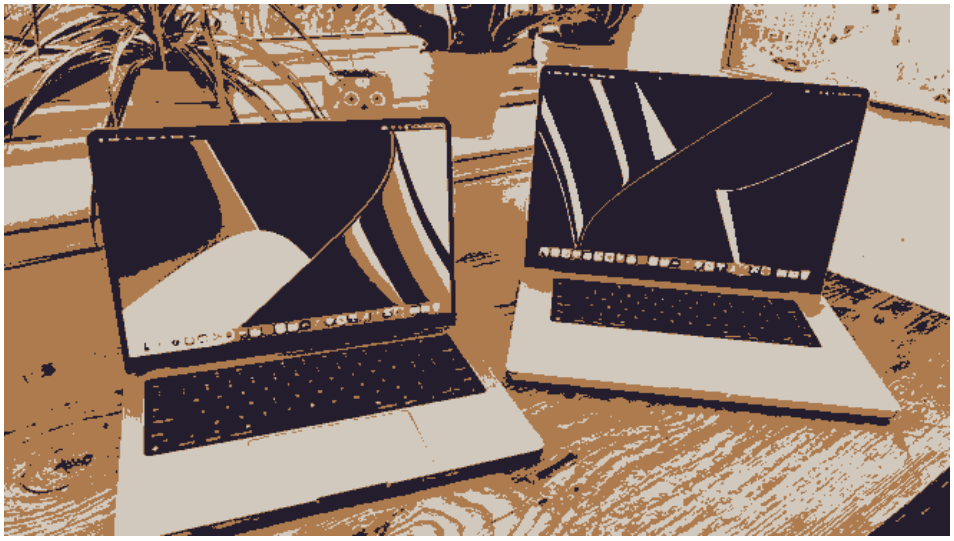
- Cluster = 3

+ Ảnh 2:

- Cluster = 7

- Cluster = 5

- Cluster = 3



2. Phân tích và nhận xét

- Nhận xét chung: Với đa số các ảnh 'random', màu sẽ không được đẹp như 'in-pixels' vì random sẽ chọn màu bất kì trong khoảng từ [0, 255] còn 'in-pixels' thì chọn màu từ chính trong ảnh nên 'in-pixels' sẽ cho ra ảnh có màu chính xác hơn. Tuy nhiên, 'random' sẽ cho ra ảnh nhanh hơn vì độ hội tụ nhanh chóng của nó.
- Nhận xét chi tiết cho dạng 'in-pixels': Ở dạng này, ảnh sẽ cho ra màu rất đẹp và chính xác đến tận số cluster = 2 thì nội dung ảnh vẫn gần như được giữ nguyên. Ngoài ra, chúng ta còn xét điều kiện nếu đến 1 thời điểm nào đó mà vị trí centroid mới không thay đổi so với centroid cũ thì ta sẽ dừng vòng lặp và cho ra kết quả, chính vì thế ở các ảnh độ phân giải lớn cũng sẽ không tốn quá nhiều thời gian để render ảnh.
- Nhận xét chi tiết cho dạng 'random': Ngược lại với 'in-pixels', dạng random mặc dù sẽ cho ra ảnh nhanh hơn 1 chút, tuy nhiên, đánh đổi với tốc độ chính là

độ chính xác không quá cao. Bên cạnh đó, ảnh chạy với 'random' sẽ có 1 tỉ lệ mảng labels sẽ xuất hiện các pixel có giá trị "NaN" (not a number) do các màu trong centroid sẽ không tồn tại trong file ảnh dẫn đến label có các pixel "NaN" sẽ cho ra ảnh đen. Điều này sẽ xảy ra với mức tỉ lệ khoảng 30-50% (trong 1 hình chạy số cluster 3, 5, 7) thì sẽ có thể bị lỗi từ 1 đến 2 hình. Tuy nhiên, tần suất ảnh bị đen không quá cao.

IV) Kết luận

- K Means clustering algorithm là 1 thuật toán thuộc machine learning, loại unsupervised, rất hữu ích trong việc phân rã các phần tử ra thành nhiều nhóm. Và giảm số lượng màu trong 1 ảnh là 1 ứng dụng cực kì hữu ích mà thuật toán này đã đem lại. Bên cạnh 'random' và 'in-pixels' sẽ còn tồn tại nhiều loại centroid khác đa dạng hơn rất nhiều. Cuối cùng, thông qua đồ án này, em đã trau dồi cho mình nhiều khía cạnh thú vị của machine learning và sự áp dụng của toán học vào công nghệ.

V) Nguồn tham khảo

- Hàm **cdist** của thư viện **scipy**: scipy.org/scipy.spatial.distance.cdist
- Hàm **mean** của thư viện **numpy**: numpy.org /numpy.mean
- Hàm **argmin** của thư viện numpy: numpy.org/ numpy.argmin
- Machine learning cơ bản: machinelearningcoban.com/kmeans
- K Means Clustering Algorithm: javatpoint.com/k-means-clustering-algorithm
- Thư viện **pyplot** của **matplotlib**: matplotlib.org/pyplot
- Hàm **imsave** của **matplotlib**: matplotlib.org/matplotlib.pyplot.imsave