UNIVERISTY OF SCIENCE - VIETNAM NATIONAL UNIVERSITY OF HCM

Faculty of Information Technology

APPLIED MATHEMATICS AND STATISTICS FOR INFORMATION TECHNOLOGY

21CLC07

Report

COLOR COMPRESSION

LECTURER

Ngô Đình Hy
Nguyễn Văn Quang Huy
Trần Hà Sơn
Ngô Đình Thúc

MEMBER

Fullname : Phạm Hồng Gia Bảo
Student ID : 21127014

Academic year of 2022-2023

Mục lục

I. THÔNG TIN CÁ NHÂN

Họ và tên	Phạm Hồng Gia Bảo
MSSV	21127014
Lớp	21CLC07

II. THUẬT TOÁN K-MEANS

1. Tóm tắt thuật toán k-Means

- K-mean clustering là một phương pháp để **tìm các cụm và hạt nhân trung tâm** của cụm **trong một tập hợp dữ liệu không được gắn nhãn**. Người ta chọn số lượng hạt nhân cụm mong muốn phân chẳng hạn như k cụm. Thuật toán K-mean di chuyển lặp đi lặp lại các hạt nhân để giảm thiểu tổng số trong phương sai cụm.
- Trong thuật toán k-Means, mỗi cụm dữ liệu được đại diện bởi một tâm (centroid) là điểm trung tâm của tất cả các quan sát trong cụm. Tâm được xác định bằng cách tính giá trị trung bình của các quan sát.
- Thuật toán sử dụng khoảng cách để xác định nhãn cho mỗi quan sát bằng cách gán nó vào cụm có tâm gần nhất. Ban đầu, thuật toán sẽ khởi tạo ngẫu nhiên một số lượng xác định trước các tâm cụm. Sau đó, thuật toán sẽ liên tục cập nhật nhãn cho từng điểm dữ liệu và tâm cụm tương ứng dựa trên khoảng cách từ điểm đó đến các tâm cụm. Thuật toán sẽ dừng lại khi tất cả các điểm dữ liệu được phân vào đúng cụm hoặc khi số lần cập nhật tâm vượt quá một ngưỡng được xác định trước.

2. Các bước thực hiện của thuật toán K-Means

- Khởi tạo ngẫu nhiên k cụm
- Lặp cho đến khi dừng:
  - Bước 1: Chọn số lượng nhóm k cần phân loại.
  - Bước 2: Chọn ngẫu nhiên k điểm dữ liệu làm trung tâm ban đầu của các nhóm.
  - Bước 3: Gán các điểm dữ liệu vào nhóm gần nhất dựa trên khoảng cách Euclidean hoặc khoảng cách Manhattan tới các trung tâm nhóm.
  - Bước 4: Cập nhật lại các trung tâm nhóm bằng cách tính trung bình của tất cả các điểm dữ liệu trong nhóm đó.
  - Bước 5: Lặp lại các bước 3 và 4 cho đến khi các trung tâm nhóm không còn thay đổi nhiều hoặc đã đạt đến một số lần lặp tối đa.

III. BÀI TOÁN COLOR COMPRESSION

1. Áp dụng thuật toán K-Means vào Color Compression

- Áp dụng thuật toán K-means clustering sẽ nhóm các màu tương tự lại với nhau thành 'k' cụm (ví dụ k=64) các màu khác nhau (các giá trị RGB). Vì vậy,

mỗi tâm cụm đại diện cho vectơ màu trong không gian màu RGB của cụm tương ứng.

- Các tâm cụm 'k' này sẽ thay thế tất cả các vectơ màu trong các cụm tương ứng của chúng. Do đó, chúng ta chỉ cần lưu nhãn cho mỗi điểm ảnh cho biết nó thuộc về cụm nào. Ngoài ra, chúng ta cần lưu một bản ghi của các vectơ màu trung tâm của mỗi cụm.

2. Ưu điểm của thuật toán k-Means

- **Tính linh hoạt**
  Thuật toán k-Means cho phép thiết lập các dữ liệu đầu vào tùy ý. Ta có thể tùy ý chỉnh số lượng clustering, số lượng centroid cũng như cách thức khởi tạo các centroids ban đầu.
- **Hiệu quả và nhanh chóng**
  Bởi có thể xử lý các hình ảnh lớn mà không yêu cầu nhiều tài nguyên tính toán hoặc bộ nhớ.
- **Có thể đạt được mức độ nén cao**
  K-means có thể giảm số lượng màu sắc đáng kể trong hình ảnh, giúp đạt được mức độ nén cao.

3. Hạn chế của thuật toán k-Means

- **Không xác định được giá trị k tối ưu nhất cho thuật toán**
  Chúng ta không có bất cứ định lượng vào về giá trị k bởi toàn bộ dữ liệu của chúng ta chưa được gán nhãn nên do đó chúng ta thử lần lượt với các giá trị k khác nhau.
- **Đòi hỏi kích thước bộ nhớ lớn**
  Bởi ta phải tính khoảng cách Euclid từ một điểm đến vị trí tất cả các tâm cụm để tìm ra tâm cụm gần nhất.
- **Vị trí tâm cụm phụ thuộc vào vị trí khởi tạo ban đầu**
  Vị trí khởi tạo khác nhau có thể dẫn tới cách phân cụm khác nhau, mặc dù thuật toán có cùng thiết lập số cụm

4. Thuật toán K-means có thể cải thiện ảnh sau nhiều lần chạy?

- K-means là một thuật toán tối ưu hóa, nghĩa là nó **tìm cách tối thiểu hóa một hàm mất mát** được định nghĩa trên tập dữ liệu. Trong trường hợp của K-means, hàm mất mát được định nghĩa bằng tổng bình phương khoảng cách giữa các điểm dữ liệu và trung tâm cụm.
- Khi chạy thuật toán K-means, **các tâm cụm được cập nhật liên tục để giảm thiểu hàm mất mát**. Các điểm dữ liệu được gán vào các cụm tương ứng với tâm cụm gần nhất. Sau đó, tâm cụm được tính lại bằng cách lấy trung bình

của các điểm dữ liệu trong cụm. Quá trình này được **lặp lại cho đến khi tâm cụm không còn thay đổi nhiều hoặc đã đạt đến số lần lặp tối đa**.

- Khi thuật toán được lặp lại nhiều lần, các tâm cụm được cập nhật để tối thiểu hóa tổng bình phương khoảng cách giữa các điểm dữ liệu và tâm cụm. Các cụm sẽ được phân loại tốt hơn và các điểm dữ liệu sẽ được phân loại vào cụm tương ứng tốt hơn. Kết quả là, ảnh sẽ hiển thị rõ ràng hơn và đẹp hơn sau mỗi lần chạy thuật toán K-means.

IV. MÃ GIẢ VÀ CÀI ĐẶT BÀI TOÁN COLOR COMPRESSION

1. Pseudocode

- Tải ảnh cần nén
- Khởi tạo số lượng cụm (k)
- Khởi tạo trọng tâm cho mỗi cụm
- Lặp lại cho số lần lặp tối đa:
  - Gán từng điểm ảnh vào trọng tâm gần nhất
  - Tính lại trọng tâm của mỗi cụm là trung bình cộng của tất cả các điểm ảnh được gán vào cụm đó
- Gán từng điểm ảnh vào trọng tâm gần nhất của nó
- Thay thế từng điểm ảnh bằng trọng tâm tương ứng
- Lưu ảnh đã nén theo lựa chọn format file yêu cầu.

2. Cài đặt thuật toán

a. def kmeans(img, clusteringNumber, max_iter, centroid)

- **Hàm này dùng để cài đặt thuật toán k-means**.
- Đầu tiên, ta khởi tạo vị trí các centroids. Nếu centroid được thiết lập thành **"random"**, các tâm cụm sẽ được khởi tạo ngẫu nhiên. Nếu centroid được thiết lập là **"in_pixels"**, các tâm cụm sẽ được khởi tạo thành các điểm ảnh được chọn ngẫu nhiên từ ảnh đầu vào.

```
if centroid == "in_pixels":
    centroids = img[
        np.random.choice(img.shape[0], size = clusteringNumber, replace = False)
    ]
# Init random
elif centroid == "random":
    centroids = np.random.randint(0, 255, size = (clusteringNumber, img.shape[1]))
else:
    raise ValueError('centroid must be "random" or "in_pixels"')
```

- Sau đó, ta thực thi vòng lặp chạy trong khoảng max_iter lần lặp. Trong mỗi vòng lặp, bằng cách sử dụng khoảng cách Euclide, tính toán khoảng cách giữa mỗi điểm ảnh trong ảnh đầu vào và mỗi tâm cụm.

```
# tính khoảng cách giữa các pixel và các centroids
distance = np.linalg.norm(img - centroids[:, np.newaxis], axis=2)
# gán nhãn cho các pixel
labels = np.argmin(distance, axis=0)
```

- Sau đó, chúng ta gán từng điểm ảnh vào tâm cụm gần nhất dựa trên khoảng cách Euclide tính được. Chúng ta cập nhật lại các tâm cụm dựa trên giá trị trung bình của tất cả các điểm ảnh được gán cho mỗi tâm cụm.

```
means = []
for j in range(clusteringNumber):
    means.append(img[labels == j].mean(axis=0))
means = np.array(means)
for i in range(clusteringNumber):
    if len(means[i]) != 0:
        centroids[i] = means[i]
```

- Sau khi vòng lặp đã hoàn thành, hàm trả về các tâm cụm cuối cùng và nhãn được gán cho mỗi điểm ảnh.

b. **def resizeImg(raw_img)**

- Hàm dùng để chuyển đổi hình ảnh ban đầu thành một mảng numpy img. Sau đó, thay đổi hình ảnh thành một mảng hai chiều, với mỗi hàng của mảng đại diện cho một điểm ảnh trong hình ảnh ban đầu.
- Tóm lại, hàm có tác dụng biến đổi ảnh đầu vào thành một mảng hai chiều có thể được sử dụng để thực hiện các thuật toán nén ảnh.

```
def resizeImg(raw_img):
    img = np.array(raw_img)
    height, width = img.shape[0], img.shape[1]
    img = img.reshape(height * width, img.shape[2])
    return img, height, width
```

c. **def processingImg(raw_img, formatPic, centroids_list)**

- Xử lý hình ảnh bằng cách sử dụng thuật toán k-means để nén hình ảnh ban đầu thành các phiên bản nhỏ hơn với số lượng màu sắc khác nhau.
- Đầu tiên, khởi tạo một danh sách img_sol chứa hình ảnh ban đầu. Lặp qua các giá trị số lượng cụm clusters khác nhau (3, 5 và 7). Thực hiện thuật toán k-means trên hình ảnh ban đầu với số lượng cụm là colorClustering, sử dụng trọng tâm được khởi tạo từ centroids_list.

```
resultImg, height, width = resizeImg(raw_img)
centroids, labels = kmeans(resultImg, colorClustering, 10, centroids_list)
```

- Gán lại nhãn cho hình ảnh dựa trên tâm cụm được tính toán từ thuật toán k-means.

```
for k in range(centroids.shape[0]):
    resultImg[labels == k] = centroids[k]
```

- Chuyển đổi lại hình ảnh đã nén về định dạng uint8 và kích thước ban đầu. Lưu hình ảnh đã nén với số lượng màu sắc tương ứng vào định dạng được chỉ định. Thêm hình ảnh đã nén vào danh sách img_sol.

```
resultImg = resultImg.astype("uint8")
resultImg = resultImg.reshape(height, width, 3)
img_sol.append(resultImg.copy())
```

- Vẽ hình ảnh kết quả trên plot với số lượng màu sắc tương ứng.

```
for i, k in [(0, 3), (1, 5), (2, 7)]:
    axis[i].set_title(f"centroids {centroids_list} with K = {k}")
    axis[i].imshow(img_sol[plot_iter])
    plot_iter += 1
plt.tight_layout()
```

⇨ **Tóm lại,** hàm **processingImg()** được sử dụng để nén hình ảnh ban đầu bằng thuật toán k-means với số lượng cụm khác nhau và lưu các phiên bản nhỏ hơn với số lượng màu sắc khác nhau. Hàm cũng sẽ trả về một bảng vẽ plot để hiển thị các phiên bản hình ảnh đã nén.

V. CÁCH SỬ DỤNG SOURCE CODE

- **Bước 1:** chọn cách thức khởi tạo các centroids

```
Choose  Init Centroids : 1

************************** Init Centroids ************************************
Enter 1: in_pixels --> centroid is a random pixels of original image
Enter 2: random    --> centroid has c channels, with c is initial random in [0,255]
```

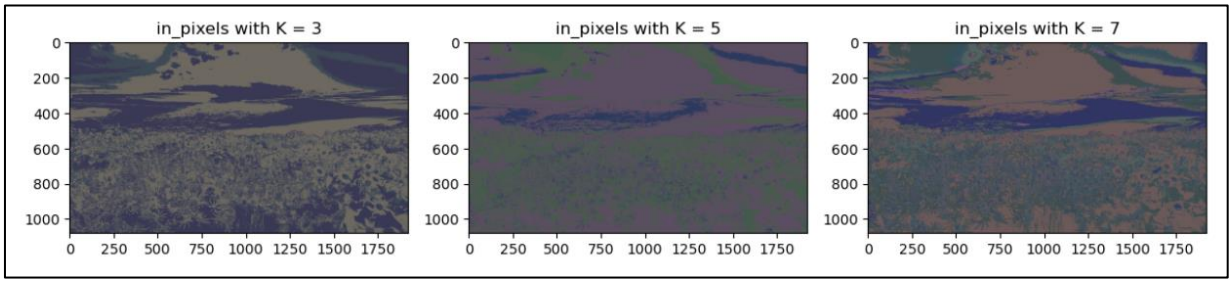
- **Bước 2:** nhập tên ảnh

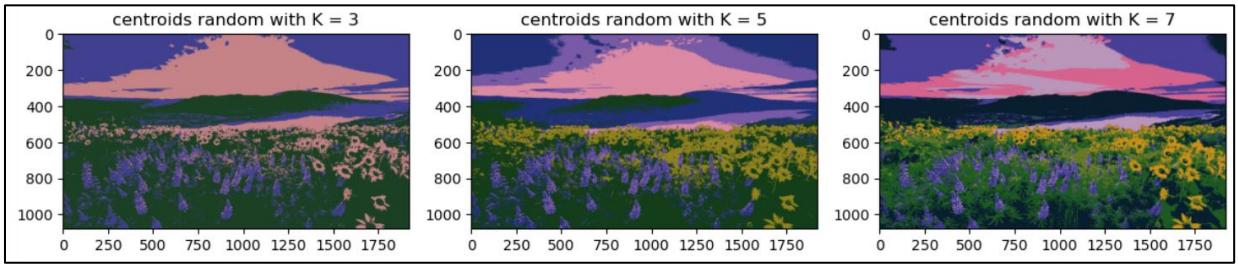
```
Choose Image Name : origin.jpg
```

- **Bước 3:** nhập định dạng output để lưu ảnh sau khi xử lý ảnh

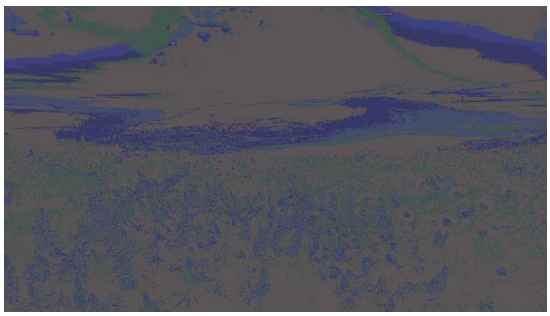
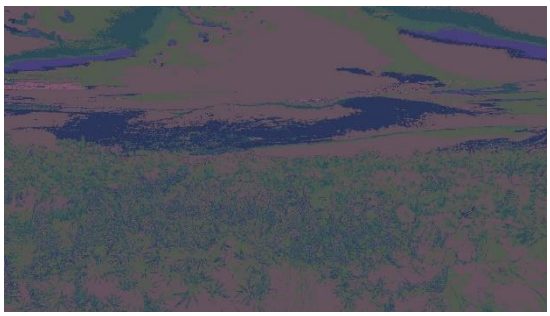
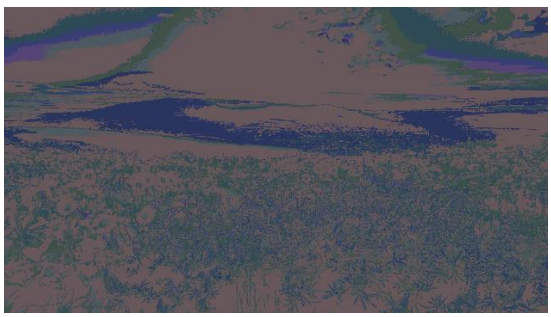
```
******************************* Output Format *********************************
Enter 1 : Output as PNG file
Enter 2 : Output as JPG file
Enter 3 : Output as PDF file

Choose option : 1
```

- **Bước 4:** xem kết quả trả tra màn hình console và các ảnh được lưu trong folder theo định dạng đã yêu cầu





VI. KẾT QUẢ THỰC HÀNH

	Centroid in pixels	Centroid in random
K=3		
K=5		
K=7		

- **Nhận xét kết quả**
  - Với cách khởi tạo các centroids random, hình ảnh rõ nét hơn so với khởi tạo in_pixels. Màu sắc các bức ảnh với random centroid tươi sáng

hơn hẳn, các chi tiết trong ảnh hiện lên rõ ràng, cụ thể hơn so với các centroid in_pixels.

- Khi khởi tạo centroids một cách random, với các giá trị k tăng dần (k=3, 5, 7), độ chi tiết của bức tranh hiện lên ngày càng rõ, chất lượng ảnh được cải thiện khi k tăng dần. Đồng thời, với nhiều màu sắc (k=5, 7), ta có thể phân biệt được 2 loại hoa trong hình. Tuy nhiên với k=3, ta hoàn toàn không thể phân biệt được 2 loại hoa này.
- Khi khởi tạo centroids bằng cách in_pixels, mặc dù các bức ảnh hiện lên rõ ràng khi ta tăng giá trị k. Tuy nhiên, bằng mắt thường ta vẫn rất khó để có thể phân biệt được các chủ thể có trong ảnh. Ta hoàn toàn không nhìn ra được 2 loại hoa như khi phân tích centroids random.

VII. MỘT SỐ THUẬN TOÁN COLOR COMPRESSION PHỔ BIẾN

1. Tổng quan:

Color compression cho phép giảm số lượng màu sắc trong một hình ảnh để giảm kích thước tệp ảnh. Có rất nhiều thuật toán nén màu sắc phổ biến:

a. Vector quantization

Đây là một phương pháp nén màu sắc khác, trong đó tập hợp các màu sắc được xác định trước được lưu trữ trong một bảng màu. Mỗi điểm ảnh trong hình ảnh được gán cho một màu sắc trong bảng màu.

b. Octree quantization

Thuật toán này sử dụng một cây Octree để phân chia không gian màu sắc của hình ảnh. Các nút lá của cây đại diện cho các khối màu sắc riêng biệt, và mỗi điểm ảnh được ánh xạ vào một nút lá tương ứng.

c. Median cut

Đây là phương pháp nén màu sắc khác, trong đó không gian màu sắc của hình ảnh được chia thành các khối con. Mỗi khối con được đại diện bằng một màu sắc trung bình, và mỗi điểm ảnh được gán cho một màu sắc tương ứng với khối con của nó.

2. Thuật toán Median Cut

a. Thuật toán

Chia không gian màu của hình ảnh thành một tập hợp 3D, mỗi hộp đại diện cho một vùng không gian màu. Hộp ban đầu bao gồm toàn bộ không gian màu của hình ảnh. Tính toán màu trung bình của mỗi hộp theo chiều dài nhất của phạm vi màu của nó. Điều này tạo ra hai hộp mới chia nhỏ hộp ban đầu theo chiều này. Lặp lại bước 2 cho mỗi hộp cho đến khi đạt được số lượng màu mong muốn. Thay thế mỗi điểm ảnh trong hình ảnh bằng màu sắc của hộp mà nó thuộc về.

b. Cách cài đặt

- Di chuyển tất cả các pixel vào một nhóm lớn duy nhất.
- Tìm kênh màu (đỏ, lục hoặc lam) trong ảnh có phạm vi lớn nhất.
- Sắp xếp các pixel theo giá trị kênh đó.
- Tìm trung vị và cắt vùng theo pixel đó.
- Lặp lại quy trình cho cả hai nhóm cho đến khi bạn có số lượng màu mong muốn.

c. Một số hàm chính trong Median Cut

- **def median_cut_quantize()**

  Thuật toán chia không gian màu thành các khối nhỏ hơn và chọn màu đại diện (trung vị) của mỗi khối. Hàm bắt đầu bằng cách kiểm tra xem đã đạt đến kích thước khối nhỏ nhất (tức là cuối của phân chia không gian màu), nếu đúng thì hàm giảm số lượng màu bằng cách đặt giá trị RGB trung bình cho tất cả các pixel trong khối. Sau đó, hàm lặp qua tất cả các pixel trong khối và đặt giá trị RGB của chúng thành giá trị trung bình.

- **def split_into_buckets(img, img_arr, depth)**

  Hàm này được sử dụng để chia không gian màu thành các khối nhỏ hơn và chọn màu đại diện (trung vị) của mỗi khối. Hàm này sử dụng đệ quy để tiếp tục chia các khối cho đến khi đạt được số lượng màu cần giảm xuống. Trong hàm này, đầu tiên, ta cần kiểm tra nếu mảng của ảnh không có phần tử nào thì nó sẽ kết thúc. Nếu độ sâu bằng 0, nó sẽ sử dụng hàm **median_cut_quantize()** để giảm số lượng màu. Sau đó, ta sẽ tính toán phạm vi của màu đỏ, màu xanh lá cây và màu xanh dương của tất cả các pixel trong mảng ảnh và tìm ra màu có phạm vi cao nhất. Sau đó, ta sẽ sắp xếp các pixel trong mảng theo màu có phạm vi cao nhất và tìm chỉ số trung vị. Hàm sử dụng chỉ số trung vị này để chia mảng thành hai phần và gọi đệ quy hàm **split_into_buckets()** với hai phần này. Cuối cùng, hàm này sử dụng vòng lặp để tạo một mảng chứa các pixel của ảnh và gọi hàm **split_into_buckets()** với mảng này và số lượng màu cần giảm xuống được truyền vào làm tham số.

d. Ưu điểm của Median Cut so với các thuật toán nén màu sắc khác

- **Bảo tồn chi tiết và kết cấu tinh tế**

  Median cut có thể giữ lại các chi tiết và kết cấu tinh tế của hình ảnh sau khi nén màu, đặc biệt là so với các thuật toán nén màu sắc khác như vector quantization.

- **Đơn giản và hiệu quả**
  Median cut là một thuật toán đơn giản và hiệu quả, không yêu cầu nhiều tài nguyên tính toán hoặc bộ nhớ.
- **Không cần phải xác định số lượng cụm trước**
  Median cut không yêu cầu xác định trước số lượng cụm đối với K-means clustering, điều này làm cho thuật toán này có tính linh hoạt hơn.
- **Không yêu cầu giả định về phân phối của màu sắc**
  Median cut không yêu cầu giả định về phân phối của các màu sắc, điều này giúp nó hoạt động tốt trên các loại hình ảnh khác nhau.

e. Nhược điểm của Median Cut so với các thuật toán nén màu sắc khác

- **Không thể đạt được mức độ nén cao**
  Median cut không hiệu quả trong việc giảm số lượng màu sắc so với một số thuật toán nén màu sắc khác như K-means clustering hoặc vector quantization.
- **Có thể dẫn đến các hiện tượng thị giác**
  Median cut có thể dẫn đến các hiện tượng thị giác như các đường biên răng cưa hoặc hiện tượng blocky nếu không được áp dụng các kỹ thuật tối ưu hóa như dithering.

VIII. TÀI LIỆU KHAM KHẢO

1. File hướng dẫn lab02.ipynb
2. https://machinelearningcoban.com/2017/01/01/kmeans/
3. https://blog.vietnamlab.vn/untitled-11/