ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐỒ ÁN 1 - COLOR COMPRESSION

TOÁN ỨNG DỤNG VÀ THỐNG KÊ CHO CÔNG NGHỆ THÔNG TIN

LỚP 21CLC08

GIẢNG VIÊN: PHAN THỊ PHƯƠNG UYÊN

Nguyễn Hồng Hạnh 21127503

Mục lục

I Ý tưởng thực hiện

1 Hình ảnh đầu vào

- Lấy ví dụ hình ảnh có kích thước 600x338 như sau:

- Chuyển hình ảnh sang kiểu dữ liệu mảng NumPy, ta sẽ có được một mảng **3 chiều** như bên dưới:

```
[[[ 92 106 145]
  [ 92 106 143]
  [ 93 107 144]
  ...
  [ 94  74 101]
  [ 94  74  99]
  [ 94  74  99]]

 [[ 90 104 143]
  [ 91 105 142]
  [ 92 106 143]
  ...
  [ 95  72  98]
  [ 95  72  98]
  [ 95  72  98]]

 [[ 89 103 142]
  [ 89 103 140]
  [ 90 104 141]
  ...
  [ 95  72  98]
  [ 95  72  98]
  [ 95  72  98]]

 ...

 [[ 24  36  14]
  [ 25  38  18]
  [ 39  52  32]
  ...
  [ 17   8   0]
  [ 18   9   4]
  [ 22  12  11]]
```

```
[[ 25  40  17]
 [ 34  47  27]
 [ 35  46  29]
 ...
 [ 21  12   7]
 [ 18  11   5]
 [ 20  13   7]]

[[ 25  40  17]
 [ 34  47  27]
 [ 28  39  22]
 ...
 [ 21  12   7]
 [ 22  15   9]
 [ 27  20  14]]]
```

- Khi in ra ***shape*** của mảng trên, ta sẽ được một ***tuple*** có 3 giá trị lần lượt tương ứng với chiều cao (height), chiều dài (width) và số kênh màu (channel) của 1 điểm ảnh (pixel):

```
(338, 600, 3)
```

- Để thuận lợi cho việc phân cụm (clustering) các điểm ảnh, ta sẽ chuyển mảng 3 chiều thành mảng 2 chiều lưu một dãy gồm *chều cao * chiều rộng* điểm ảnh như sau:

```
[[ 92 106 145]
 [ 92 106 143]
 [ 93 107 144]
 ...
 [ 21  12   7]
 [ 22  15   9]
 [ 27  20  14]]
```

2 Thuật toán K-Means

a Dữ liệu đầu vào

- Mảng 2 chiều lưu một dãy các điểm ảnh
- Số lượng màu hay số lượng cụm để thực hiện gom nhóm các điểm ảnh
- Số vòng lặp tối đa để cập nhật centroids và phân cụm các điểm ảnh, mặc định là 10
- Cách thức khởi tạo centroids, bao gồm chọn điểm ảnh ngẫu nhiên từ không gian màu và chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

b Khởi tạo centroids

- Centroids sẽ được lưu dưới kiểu dữ liệu là mảng NumPy 2 chiều lưu một dãy gồm k điểm ảnh, với k là số lượng cụm để thực hiện gom nhóm các điểm ảnh

Chọn điểm ảnh ngẫu nhiên từ không gian màu

- Giá trị từng kênh màu của các điểm ảnh làm centroid là một số nguyên được lấy ngẫu nhiên trong đoạn [0, 255]

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

- Từ mảng các điểm ảnh của hình ảnh đầu vào, ta tạo một mảng lưu những điểm ảnh khác nhau có trong hình ảnh đầu vào đó
- Từng điểm ảnh làm centroid sẽ được lấy ngẫu nhiên từ mảng trên

c Phân cụm (clustering)

- Với mỗi điểm ảnh của hình ảnh đầu vào trong mảng 2 chiều, ta tiến hành phân cụm bằng cách tính khoảng cách Euclid (Euclidean distance) và phân điểm ảnh vào cụm có centroid gần với điểm ảnh đó nhất (khoảng cách Euclid nhỏ nhất)

$$\text{Euclidean distance} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

- Cụm mà điểm ảnh đó được phân vào sẽ được lưu vào mảng 1 chiều ở vị trí tương ứng với vị trí của điểm ảnh đó

d Cập nhật centroids

- Sau khi hoàn thành phân cụm các điểm ảnh, ta sẽ tiến hành cập nhật centroids
- Đối với cụm không có điểm ảnh nào được phân vào, centroid sẽ được giữ nguyên cho lần phân cụm kế tiếp
- Đối với các cụm đã có điểm ảnh được phân vào, centroid sẽ được cập nhật thành trung vị (mean) ở từng kênh màu của tất cả điểm ảnh trong của cụm đó

e Điều kiện dừng

- Vòng lặp để cập nhật centroids và phân cụm các điểm ảnh sẽ dừng khi đã đạt số vòng lặp tối đa hoặc khi các kênh màu của các centroids hiện tại và centroids mới tìm được ở cụm tương ứng chênh lệch nhau không quá 10 đơn vị
- Centroids sẽ liên tục cập nhật và phân cụm lại các điểm ảnh cho đến khi thoả điều kiện dừng ở trên

II Mô tả các hàm

1 Thư viện sử dụng

- **NumPy:** Thực hiện các thao tác trên ma trận
- **PIL:** Dọc và lưu ảnh
- **matplotlib.pyplot:** hiển thị hình ảnh
- **os:** Kiểm tra và tạo đường dẫn lưu file

2 def img_1d_generate(input)

- Hàm ***img_1d_generate*** sẽ nhận input là mảng NumPy 3 chiều chứa các điểm ảnh và sẽ trả về mảng 2 chiều gồm 1 dãy các điểm ảnh

```
def img_1d_generate(input):
  return input.reshape((-1, input.shape[2]))
```

3 def euclidean_distance(px, centroids)

- Hàm ***euclidean_distance*** được dùng để tính khoảng cách Euclid của điểm ảnh và các centroids
- Hàm sẽ nhận vào mảng 1 chiều lưu điểm ảnh (*px*) và mảng 2 chiều lưu các centroids (*centroids*), sau đó trả về mảng 1 chiều gồm k phần tử tương ứng với khoảng cách của điểm ảnh tới k centroids

```
def euclidean_distance(px, centroids):
  return np.sqrt(np.sum((px - centroids)**2, axis=1))
```

4 def kmeans(img_1d, k, init_centroids, max_iter=10)

- Hàm ***kmeans*** sẽ dùng thuật toán K-means để phân cụm các điểm ảnh
- Hàm sẽ nhận dữ liệu đầu vào là mảng 2 chiều lưu một dãy các điểm ảnh (*img_1d*), số lượng cụm (k) và cách thức khởi tạo centroids (*init_centroids*: 0 là chọn ngẫu nhiên từ không gian màu và 1 là chọn ngẫu nhiên từ hình ảnh đầu vào). Số vòng lặp tối đa (*max_iter*) sẽ đặt mặc định là 10

```
def kmeans(img_1d, k, init_centroids, max_iter=10):
  if init_centroids == 0:
    centroids = np.random.randint(266, size=(k, len(img_1d[0])))
  else:
    unique = np.unique(img_1d, axis=0)
    centroids = unique[np.random.randint(len(unique), size=k), :]
```

- Khi bắt đầu, hàm sẽ khởi tạo k centroids bằng cách thức tương ứng với dữ liệu được truyền vào:

Chọn điểm ảnh ngẫu nhiên từ không gian màu

+ Ta sẽ dùng hàm ***np.random.randint*** để tạo mảng 2 chiều centroids lưu k điểm ảnh, mỗi kênh màu của từng điểm ảnh là giá trị ngẫu nhiên trong đoạn [0, 255]

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

+ Từ mảng *img_1d*, ta dùng hàm ***np.unique*** để lấy các điểm ảnh **không trùng lặp** của hình ảnh đầu vào và lưu vào mảng *unique*

+ Sau đó, ta dùng hàm ***np.random.randint*** để lấy ngẫu nhiên k phần tử từ mảng unique và lưu vào mảng centroids

```
  for _ in range(max_iter):
    img_cluster = []

    for px in img_1d:
      px_distances = euclidean_distance(px, centroids)
      px_cluster = np.argmin(px_distances)
      img_cluster.append(px_cluster)

    img_cluster = np.array(img_cluster)

    img_cluster_id = []

    for i in range(k):
      img_cluster_id.append(np.argwhere(img_cluster == i))

    cluster_centers = []

    for i, id in enumerate(img_cluster_id):
      if len(id) == 0:
        cluster_centers.append(centroids[i])
      else:
        cluster_centers.append(np.mean(img_1d[id], axis=0, dtype=int)[0])

    if np.max(abs(centroids - np.array(cluster_centers))) < 10:
      break
    else:
      centroids = np.array(cluster_centers)

  return centroids, img_cluster
```

- Tiếp đến sẽ là vòng lặp để thực hiện phân cụm các điểm ảnh cũng như cập nhật centroids cho lần phân cụm tiếp theo

Phân cụm các điểm ảnh

```
    img_cluster = []

    for px in img_1d:
      px_distances = euclidean_distance(px, centroids)
      img_cluster.append(np.argmin(px_distances))

    img_cluster = np.array(img_cluster)

    img_cluster_id = []

    for i in range(k):
      img_cluster_id.append(np.argwhere(img_cluster == i))
```

+ Với mỗi điểm ảnh trong mảng *img_1d*, ta sẽ tính khoảng cách tới các centroids bằng hàm ***euclidean_distance*** và lưu các giá trị khoảng cách vào mảng *px_distances* ở vị trí tương ứng với vị trí của từng centroid trong mảng *centroids*. Như vậy, nhãn (label) đại diện cho mỗi cụm sẽ tương ứng với vị trí của centroid của mỗi cụm trong mảng *centroids*, đồng thời cũng là vị trí của khoảng cách Euclid từ điểm ảnh đến centroid đó trong mảng *px_distances*

+ Sau đó ta sẽ có mảng *img_cluster* để lưu nhãn của cụm mà mỗi điểm ảnh được phân vào tại vị trí tương ứng với vị trí của điểm ảnh đó trong mảng *img_1d*. Ta sẽ dùng hàm ***np.argmin*** để lấy vị trí có giá trị khoảng cách từ điểm ảnh tới centroid nhỏ nhất, cũng tức là nhãn của cụm tương ứng với centroid đó

+ Ngoài ra, ta sẽ dùng hàm ***np.argwhere*** để có được mảng chứa các vị trí của điểm ảnh thuộc cùng một cụm và lưu vào vị trí tương ứng với nhãn của cụm đó trong list *img_cluster_id*

Cập nhật centroids

+ Ta sẽ tạo một list rỗng *cluster_centers* để lưu các centroids mới sau khi cập nhật của các cụm ở vị trí tương ứng với vị trí của các centroid cũ trong mảng *centroids*

+ Đối với các cụm không có điểm ảnh nào được phân vào, đồng nghĩa với việc là không có phần tử nào trong mảng chứa các vị trí của điểm ảnh thuộc cụm đó ($len(id) == 0$), ta sẽ giữ nguyên centroid của cụm đó cho lần phân cụm sau và thêm vào list *cluster_ center*

+ Đối với các cụm đã có điểm ảnh được phân vào, ta sẽ dùng hàm ***np.mean*** để tìm centroid mới cho cụm đó. Centroid mới này sẽ có giá trị của từng kênh màu là giá trị trung vị kênh màu tương ứng của tất cả điểm ảnh trong của cụm đó, và sẽ được thêm vào list *cluster_ centers*

```
    cluster_centers = []

    for i, id in enumerate(img_cluster_id):
      if len(id) == 0:
        cluster_centers.append(centroids[i])
      else:
        cluster_centers.append(np.mean(img_1d[id], axis=0, dtype=int)[0])
```

Kiểm tra điều kiện dừng

+ Hàm ***np.max*** sẽ được dùng để lấy ra giá trị chênh lệch ở các kênh màu lớn nhất giữa các centroids cũ và các centroids mới tương ứng

+ Nếu giá trị chênh lệch lớn nhất nhỏ hơn 10, vòng lặp sẽ dừng và hàm sẽ trả về 2 mảng *centroids* và *img_ cluster*

```
    if np.max(abs(centroids - np.array(cluster_centers))) < 10:
      break
    else:
      centroids = np.array(cluster_centers)
```

+ Nếu chưa thoả điều kiện dừng, mảng *centroids* sẽ được cập nhật bằng các centroids mới ở list *cluster_ centers* và tiếp tục phân cụm các điểm ảnh

```
  return centroids, img_cluster
```

5 def main()

```
def main():
  print("Image path:")
  img_path = input()
  img = np.array(PIL.Image.open(img_path))
  print("Number of clusters (> 0):")
  k = int(input())
  print("Init centroids (0: random, 1: in pixels):")
  init_centroids = int(input())
  print("Image format (0: png, 1: pdf):")
  format_input = int(input())

  if k > 0 and init_centroids in range(0, 2) and format_input in range(0, 2):
    img_1d = img_1d_generate(img)
    centroids, img_cluster = kmeans(img_1d, k, init_centroids)
    new_img = []
    for id in img_cluster:
      new_img.append(centroids[id])

    img_name = img_path.split('/')
    img_format = "png" if format_input == 0 else "pdf"
    new_img_name = img_name[-1].split('.')[0] + "." + img_format
    if not os.path.exists("./output"):
      os.makedirs("./output")

    res = PIL.Image.fromarray(np.reshape(new_img, img.shape).astype(np.uint8))
    res.save(f"./output/{new_img_name}")
    plt.imshow(np.reshape(new_img, img.shape))
  else:
    print("Invalid input")
```

- Hàm ***main*** sẽ thực hiện lấy dữ liệu được nhập từ bàn phím và kiểm tra dữ liệu

- **Nếu dữ liệu thoả yêu cầu**:

+ Hàm ***kmeans*** sẽ được gọi và kết quả trả về dẽ được lưu vào hai biến centroids và img_cluster, từ đó tạo hình ảnh mới bằng cách thay các điểm ảnh bằng centroid của cụm chứa điểm ảnh đó vào từng vị trí tương ứng và lưu vào list *new_ img*

+ List *new_ img* qua hàm ***np.reshape*** sẽ thành mảng 3 chiều có kích thước là kích thước của hình ảnh ban đầu

+ Kết hợp hàm ***PIL.Image.fromarray*** và ***Image.save*** để lưu kết quả

+ Hình ảnh sau khi giảm số lượng màu cũng sẽ được in trên màn hình bằng hàm ***plt.imshow***

- **Nếu có dữ liệu sai yêu cầu**: hàm sẽ in ra màn hình thông báo: "Invalid input"

III Kết quả

1 Hình 1

Kích thước hình ảnh: 500 * 300

k = 3

Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

k = 5

Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

k = 7

Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

k = 15

Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

k = 27

Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

Nhận xét

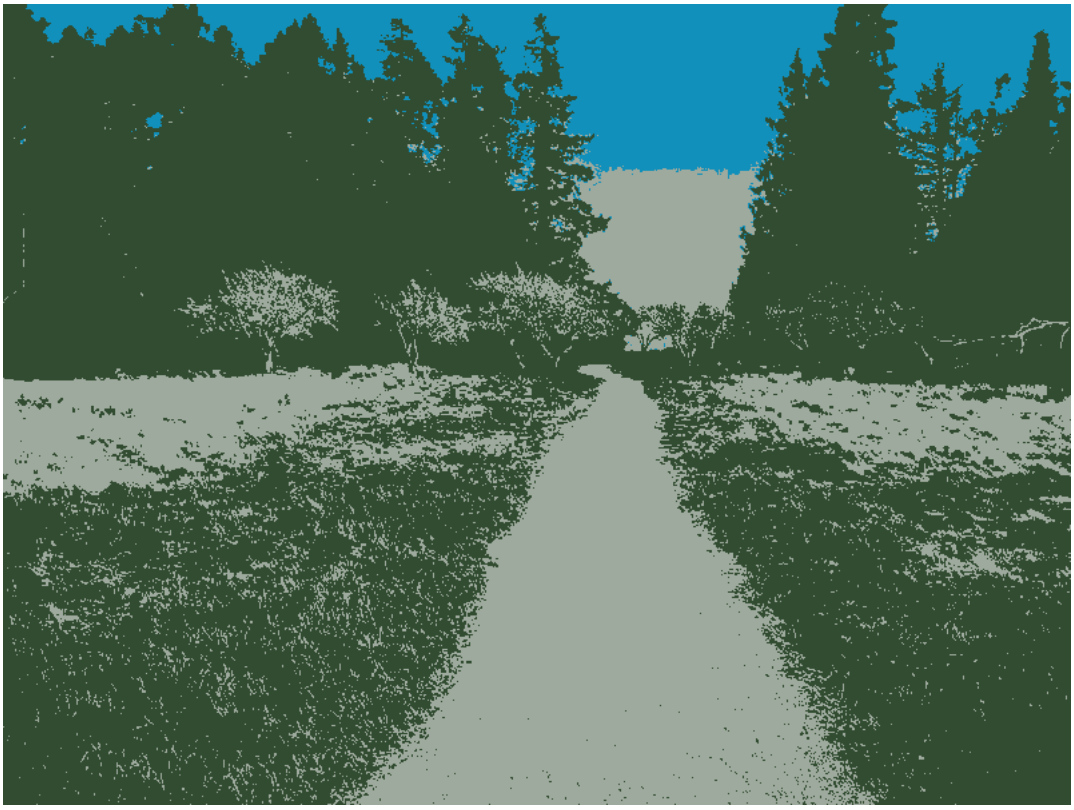
- Với hỉnh ảnh kích thước nhỏ, thời gian xuất hình vừa phải (nằm trong khoảng 6-7 giây)

- Đối với hình ảnh có các màu sáng tối khá "gần"nhau, hình ảnh cho ra không quá tốt khi k = 3. Do điều kiện dừng chưa tốt nên trong trường hợp centroids được lấy ngẫu nhiên từ không gian màu, khi k nhỏ thì centroids chọn chưa được tốt

2 Hình 2

Kích thước hình ảnh: 800 * 600

k = 3



Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

k = 5

Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

k = 7

Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

k = 15

Chọn điểm ảnh ngẫu nhiên từ không gian màu

Chọn điểm ảnh ngẫu nhiên từ hình ảnh đầu vào

Nhận xét

- Với hình ảnh kích thước trung bình, thời gian xuất hình khá lâu (khoảng 25 giây đối với k = 3 và hơn gần 40 giây đối với k = 15)

- Đối với số hình ảnh có các màu tương phản rõ rệt hơn, trong trường hợp centroids được lấy ngẫu nhiên từ không gian màu khi k nhỏ có thể sự khác biệt giữa các màu ảnh, tuy nhiên màu ảnh có thể có sự khác biệt lớn so với ảnh gốc

Tổng kết

- Vì centroid chỉ lấy ngẫu nhiên nên có thể ra những trường hợp centroid không tốt, đặc biệt là khi lấy ngẫu nhiên trong không gian màu, dẫn đến hình ảnh ra đôi khi chưa tốt

- Điều kiện dừng chưa tốt nên đối với trường hợp k nhỏ thì centroids cuối cùng vẫn chưa được tối ưu

IV Tài liệu tham khảo

1. https://www.youtube.com/watch?v=5w5iUbTlpMQ - **K-Means Clustering From Scratch in Python (Mathematical)**

2. https://numpy.org/ - **Các hàm của NumPy**