ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

○♦♦♦ 🕮 ♦♦♦○

BÁO CÁO ĐỒ ÁN 1

COLOR COMPRESSION

Môn học: Toán ứng dụng và thống kê

♣GIÁO VIÊN HƯỚNG DẪN♣

TS. Lê Thanh Tùng

ThS.Phan Thị Phương Uyên

ThS.Vũ Quốc Hoàng

ThS. Nguyễn Văn Quang Huy

Thành phố Hồ Chí Minh - 2021

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

○♦♦♦ 🕮 ♦♦♦○

BÁO CÁO ĐỒ ÁN 1

COLOR COMPRESSION

Môn học: Toán ứng dụng và thống kê

Thành phố Hồ Chí Minh - 2021

MỤC LỤC

MỤC LỤC 3
THÔNG TIN CÁ NHÂN 4
Ý TƯỞNG THỰC HIỆN – MÔ TẢ CÁC HÀM 4
KẾT QUẢ THỬ NGHIỆM 9
NHẬN XÉT 12
TÀI LIỆU THAM KHẢO 14

THÔNG TIN CÁ NHÂN

Mã số sinh viên	Họ và tên	Chú thích
20127662	Nguyễn Đình Văn	20127662@student.hcmus.edu.vn

Ý TƯỞNG THỰC HIỆN – MÔ TẢ CÁC HÀM

Ý tưởng: Các bức ảnh màu khi được máy tính đọc vào dưới dạng ma trận. Đối với ảnh RGB với mỗi phần tử hay còn gọi là điểm ảnh (Pixel) được lưu với các giá trị của kênh màu RBG có giá trị [0..255] (Số màu trong ảnh RBG có thể là $256^3 \approx 1.7 \times 10^7$). Trong đồ án này em sử dụng thuật toán K-Means để giảm số lượng màu cho ảnh, thuật toán gọm cụm các điểm ảnh lại với nhau, tạo thành 1 cluster nhờ đó giảm lại số lượng màu cần dùng cho ảnh. Với thuật toán K-Means Clustering là chúng ta không biết những điểm ảnh(Pixel) cho trước được gom cụm theo tiêu chí nào, thuật toán chỉ trả về k-cluster được đưa vào do đó ta hoàn toàn chủ động về việc giới hạn số lượng màu có thể để giảm số màu ảnh.

Thuật toán K-means clustering:

- Bước 1: Khởi tạo các Centroid (K: Số centroid tùy ý người dùng).
- Bước 2: Phân các cụm, phân các pixel dựa vào mỗi centroid tương ứng.
- Bước 3: Cập nhật lại Centroid mới dựa vào các cụm đã được phân trước đó (Giá trị của Centroid mới bằng giá trị trung bình của cụm tương ứng).
- Bước 4: Kiểm tra đã đạt kết quả hội tự chưa. Nếu chưa thì quay lại bước 2

Mô tả các hàm:

- Hàm Kmeans: Hàm xử lí chính thuật toán K-means, đưa ma trận 3 chiều của ảnh RGB về ma trận 2 chiều để dễ xử lí ma trận với thư viện Numpy. Sau đó ta tiến thành việc tạo Centroid dựa vào init_centroids do người dùng truyền vào. Vòng lặp là việc thực hiện lặp đi lặp lại việc xác định Centroid và label cho đến khi thu được kết quả hội tụ mong muốn (Dựa vào Centroid xác định Label sau đó cập nhật lại Centroid mới và tiếp tục lặp lại). Sau khi kết thúc lặp (Thu được kết quả hội tụ(hasConverged = True) hoặc max_iter (số lượng lặp) bằng 0) thì ta tiến hành cập nhật lại các điểm ảnh mới cho ảnh, đưa về lại ma trận 3 chiều và trả về Centroid và các điểm ảnh vừa cập nhật.
  - Input: img (ma trận điểm ảnh), k_clusters (Số cluster), maxIter (số vòng lặp tối đa), init_centroids(Kiểu khởi tạo centroid).
  - Output: centroid (ma trận các điểm ảnh được chọn), newImg (ma trận ảnh mới).

```
def kmeans(img, k_clusters, maxIter, init_centroids):
    rowImg = img.shape[0]
    columnImg = img.shape[1]
    channelImg = img.shape[2]
    img = img.reshape(rowImg * columnImg, channelImg)
    #init centroids
    centroid = initCentroids(img, k_clusters, init_centroids)
    labels = []
    while True:
        #assign label of data point
        new_label = label(img, centroid)
        labels = new_label
        new_centroid = updateCentroid(img, labels, k_clusters, channelImg)
        #if compare against the absolute difference between new centroid and old centroid or maxIter = 0 then break loop.
        if hasConverged(centroid, new_centroid) or not(maxIter):
            break
        centroid = new_centroid
        maxIter-=1
    newImg = updateImage(img, k_clusters, labels, centroid).reshape(rowImg, columnImg, channelImg)
    return centroid, newImg
```

- Hàm initCentroids: Hàm khởi tạo centroid, dựa vào kiểu khởi tạo centroid mà người dùng truyền vào ta có 2 kiểu khởi tạo *'in_pixels'* và *'random'* ở loại *'in_pixels'* các centroid được khởi tạo sẽ dựa vào điểm ảnh có trên ảnh, còn ở loại *'random'* các centroid sẽ được khởi tạo một các ngẫu nhiên với các giá trị [0 → 255]. Sử dụng hàm np.rand.randint để tạo ra ma trận số nguyên. Kết quả trả về là ma trận các centroid.
  - Input: img(ma trận điểm ảnh), k_cluster(Số cluster), InitCentroidType(Kiểu khởi tạo centroid).
  - Output: Ma trận Centroid.

```python
def initCentroids(img, k_cluster, InitCentroidType):
    rowImg = img.shape[0]
    columnImg = img.shape[1]
    if InitCentroidType == 'random':
        #return matrix K_cluster * 3 centroid random (value 0 -> 255)
        return np.random.randint(0, 255, (k_cluster, columnImg))
    if InitCentroidType == 'in_pixels':
        #return matrix K_cluster * 3 centroid of original image
        return img[np.random.randint(0, rowImg, k_cluster)]
```

- Hàm label: Hàm xác định khoảng cách của mọi điểm ảnh với từng centroid tương ứng, sau đó trả về ma trận chứa thông tin vị trí của centroid nào gần pixel đang xét nhất.
  - Input: img (Ma trận điểm ảnh), centroid (Ma trận Centroid).
  - Output: Ma trận khoảng cách ngắn nhất giữa Centroid với mỗi Pixel

```python
def label(img, centroid):
    rowImg = img.shape[0]
    #distance from pixels to centroid
    dist = np.sqrt(((img - centroid[0]) ** 2).sum(axis=1)).reshape((rowImg, 1))
    #for i = 1 to (numbers of centroid - 1) to calc distance with orther centroids
    for i in range(1, centroid.shape[0]):
        temp = np.sqrt(((img - centroid[i]) ** 2).sum(axis=1)) #axis = 1 ~~ Row
        temp = temp.reshape((rowImg, 1))
        #concat new distance matix with old distance matrix
        dist = np.concatenate((dist,temp),axis = 1)
    #return smallest distance's label centroid for each pixel
    return np.argmin(dist,axis = 1)
```

- Hàm updateCentroid: Dựa vào label truyền vào ta kiểm tra xem cluters có chứa dữ liệu hay không nếu có thì cập nhật lại Centroid bằng hàm tính trung bình, nếu không thì bỏ qua.
  - Input: img(Ma trận điểm ảnh), label(Ma trận label), k_cluster (số k_cluster), channel (Số chiều)
  - Output: Ma trận Centroid.

```python
def updateCentroid(img, label, k_cluster, channel):
    centroid = np.zeros((k_cluster, channel))
    for i in range(0, k_cluster):
        #assign k cluster from img for temp
        temp = img[label == i]
        #if cluster have data then update centroid
        if np.size(temp) != 0:
            centroid[i] = np.average(temp, axis = 0)
    return centroid
```

- Hàm updateImage: Hàm tạo ra ma trận mới sau đó với mỗi pixel có label k thì được thay thế giá trị của centroid ứng với cluster k.
  - Input: img(Ma trận điểm ảnh), k_clusters (Số cluster), label (Ma trận khoảng cách), centroid (Ma trận centroid)
  - Output: Ma trận các điểm ảnh mới ứng với K điểm màu.

```
def updateImage(img, k_clusters, label, centroid):
    rowImg = img.shape[0]
    columnImg = img.shape[1]
    temp = np.zeros((rowImg, columnImg))
    for i in range(0, k_clusters):
        #assign all pixels
        temp[label == i] = centroid[i]
    return temp
```

- Hàm imgArr: Hàm nhận tên ảnh từ bàn phím sau đó đọc ảnh và trả về ma trận các điểm ảnh.
  - Input: Tên ảnh do người dùng nhập từ bàn phím.
  - Output: Ma trận điểm ảnh

```
def imgArr ():
    nameFile = input("Input name of image: ")
    img = Image.open(nameFile)
    img = np.asarray(img)
    return img
```

- Hàm hasConverged: Hàm kiểm tra xem Centroid mới và Centroid cũ có hội tụ hay không, ở đây kiểm tra 2 ma trận có chênh lệch nhau quá 0.01 hay không nếu có thì trả về false, nếu không thì trả về true (hội tụ).
  - Input: centroids (Centroid cũ), newCentroids(Centroid mới).
  - Output: True(Hội tụ) hoặc false(Không hội tụ).

```
def hasConverged(centroids, newCentroids):
    for i in range(0, centroids.shape[0]):
        temp = centroids[i] - newCentroids[i]
        if (temp.any() > 0.01):
            return False
    return True
```

- Hàm showCentoidsLabels: Hàm in kết quả cuối cùng của Centroid và label ra màn hình console.

```
def showCentoidsLabels (centroids, label):
    print('Centroids: \n', centroids)
    print('Labels: \n', label)
```

- Hàm showImage: Hàm nhận vào ma trận điểm ảnh sau đó lưu về với tên mặc định “image.png”.
  - Input: Ma trận ảnh
  - Output: Ảnh ‘image.png’

```
def saveImage (img):
    plt.imsave('image.png', np.array(img, dtype='uint8'))
```

- Hàm main: Hàm nhận ma trận các điểm ảnh từ hàm imgArr và truyền cho hàm Kmeans thực hiện thuật toán, sau đó in ra kết quả ma trận các Centroid và Label các điểm ảnh mới.

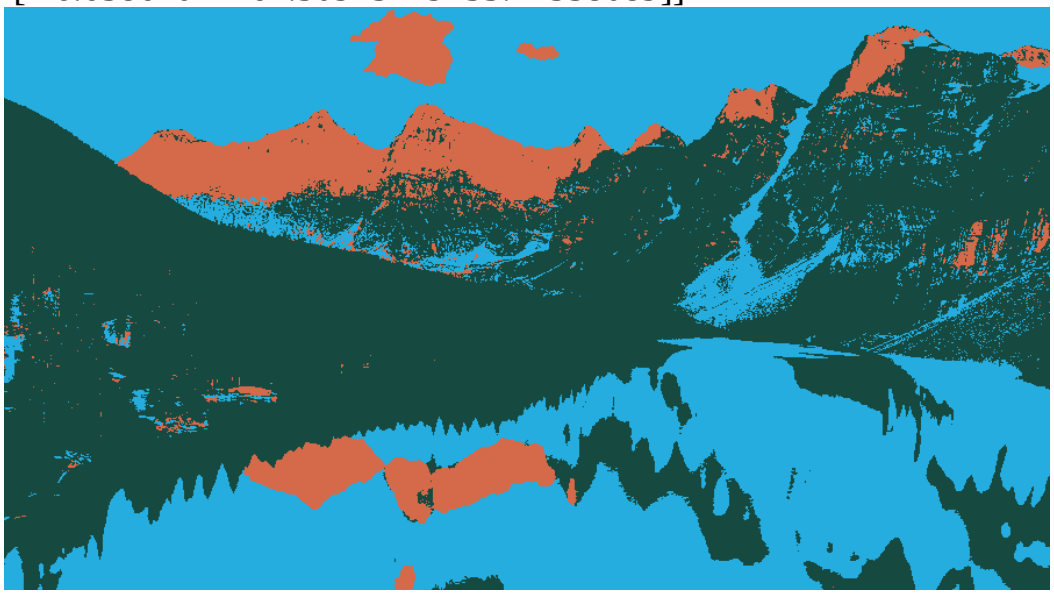
```
if __name__ == '__main__':
    img = imgArr()
    k_clusters = 3
    maxIter = 2000
    #init_centroids = 'in_pixels' OR init_centroids = 'random'
    init_centroids = 'in_pixels'
    newCentroid, newImg = kmeans(img, k_clusters, maxIter, init_centroids)
    showCentoidsLabels(newCentroid, newImg)
    saveImage(newImg)
    plt.imshow(newImg.astype(np.uint8))
    plt.show()
```

KẾT QUẢ THỬ NGHIỆM

Em sử dụng ảnh bên dưới để thử nghiệm ảnh có kích thước [900 x 501]

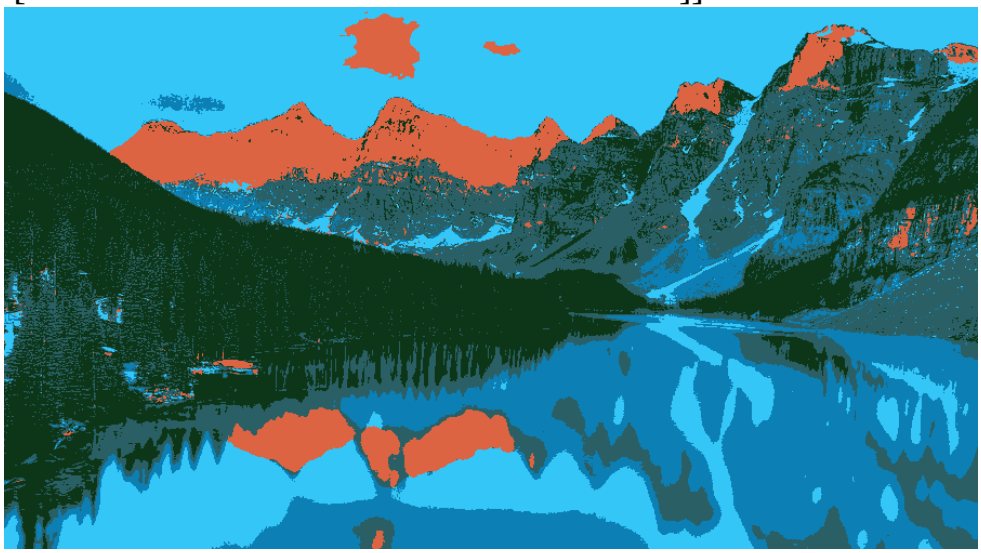
- Với K = 3

Centroids:
[[51.04184057 192.18085314 241.88331402]
[41.83980461 111.30381019 137.72659456]
[46.63867014 62.50945119 33.72558089]]



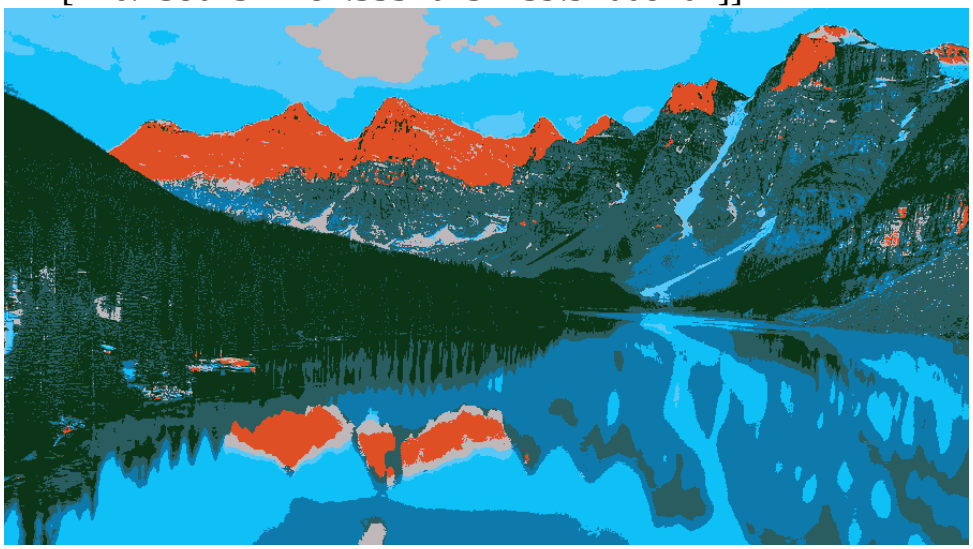
- Với K = 5

Centroids:
[[12.58247294 127.62046274 180.36168678]
[13.04589376 54.6478405   25.83229318]
[220.79029973 100.78849412 66.09247479]
[42.42305825 96.17031122 101.77814658]
[51.82644478 198.69971022 247.31559861]]



- Với K = 7

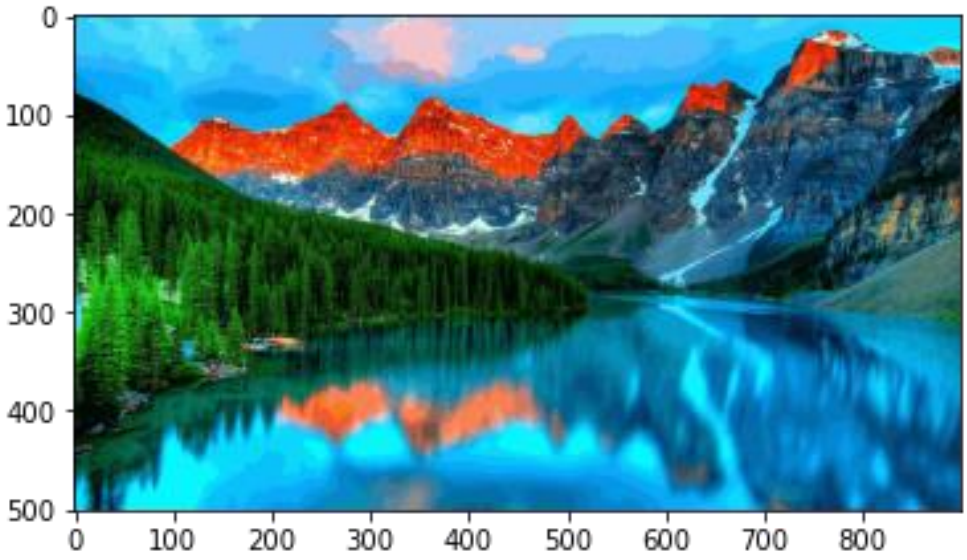
Centroids:
[[31.73698202 195.98767266 248.9173983]
[166.22850956 200.38441739 224.68123876]
[9.75570832 124.73256131 177.63424448]
[13.43706216 54.67423396 19.37864235]
[19.76710019 76.47909287 94.11454153]
[82.46898867 122.75558159 109.814315]
[226.15804311 82.33310734 39.94008707]]



- Với K = 100

Centroids:

[[6.36935546 119.21999228 127.70860672]
[4.99268704 50.16782299 30.33414588]
…….
[63.0007983 124.24667376 154.59127195]
[78.43445925 110.31453362 130.93616362]]



NHẬN XÉT

Qua tổng thể, thì em nhận xét được chương trình cho kết quả tốt, hình vẫn giữ được hình dạng ban đầu, ta vẫn nhận thấy được và tưởng tượng được các sự vật có trong hình. Tuy nhiên có một hạn chế là chương trình với số K (K_cluster) càng lớn thì thời gian chạy càng tăng.

Ta xét với từng giá trị của K [2 → 25] thì thu được thời gian:

With K = 2 - elapsed time: 0.85545121861267[sec]

With K = 3 - elapsed time: 4.929551654293823[sec]

With K = 4 - elapsed time: 5.322567272545[sec]

With K = 5 - elapsed time: 6.4742542954545129[sec]

With K = 6 - elapsed time: 11.9065456152004[sec]

With K = 7 - elapsed time: 8.232402420043945[sec]

With K = 8 - elapsed time: 14.99388740272522[sec]

With K = 9 - elapsed time: 11.848001956939697[sec]

With K = 10 - elapsed time: 34.901201009750366[sec]

With K = 11 - elapsed time: 18.4078311920166[sec]

With K = 12 - elapsed time: 46.30548071861267[sec]

With K = 13 - elapsed time: 45.72952604293823[sec]

With K = 14 - elapsed time: 77.15225672721863[sec]

With K = 15 - elapsed time: 64.74254298210144[sec]

With K = 16 - elapsed time: 117.06662487983704[sec]

With K = 17 - elapsed time: 140.32402420043945[sec]

With K = 18 - elapsed time: 123.38168740272522[sec]

With K = 19 - elapsed time: 101.39834499359131[sec]

With K = 20 - elapsed time: 116.34881949424744[sec]

With K = 21 - elapsed time: 101.17928862571716[sec]

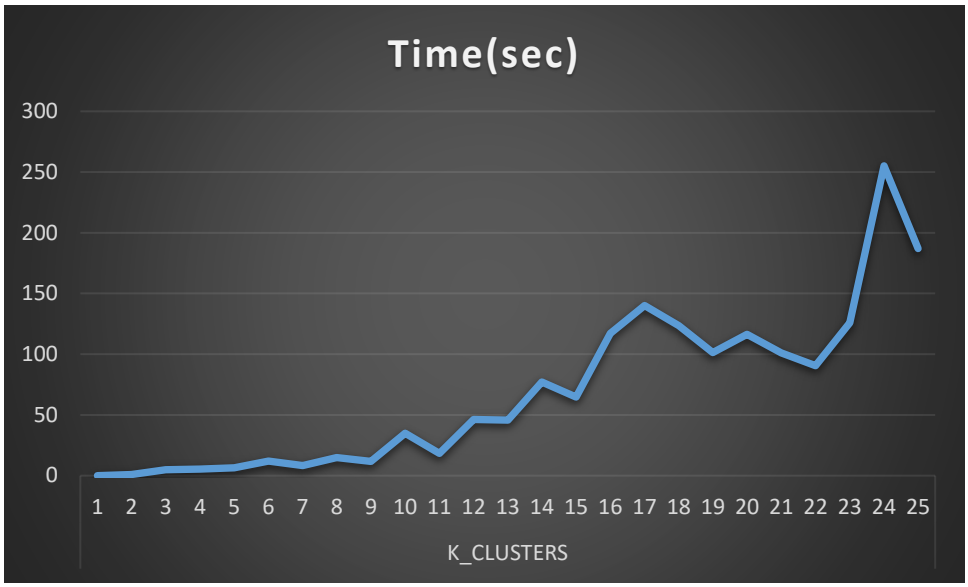
With K = 22 - elapsed time: 90.60803127288818[sec]

With K = 23 - elapsed time: 125.97713780403137[sec]

With K = 24 - elapsed time: 225.12159991264343[sec]

With K = 25 - elapsed time: 187.09570956230164[sec]

Qua biểu đồ



Ta thấy nhìn chung với giá trị K càng tăng thì thời gian càng lâu. Thời gian thực thi nhanh nhất là K = 2 với 0.85s và lâu nhất với K = 24 với 255.121s. Với các giá trị K gần nhau thì có độ chênh lệch không lớn, nhưng có lúc thời gian thực thi tăng, có lúc giảm so với lần trước đó. Nguyên nhân có thể do lúc random Centroid ra được kết quả gần với điểm hội tụ.

TÀI LIỆU THAM KHẢO

https://machinelearningcoban.com/2017/01/01/kmeans/

https://vi.wikipedia.org/wiki/Ph%C3%A2n_c%E1%BB%A5m_k-means

https://www.youtube.com/watch?v=4b5d3muPQmA

https://codelearn.io/sharing/tim-hieu-thu-vien-numpy-trong-python

https://github.com/kvmduc/applied-math/blob/main/Applied%20Math/Lab02/kmeans.py