20127166 - Nguyễn Huy Hoàn

Đồ án 1: Color Compression

Giáo viên hướng dẫn

- ThS. Phan Thị Phương Uyên
- ThS. Nguyễn Văn Quang Huy

Các thư viện sử dụng trong đồ án này

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

Các bước thực hiện thuật toán K-Means

- Bước 1: Khởi tạo centroid
- Bước 2: Tính toán độ lệch giữa các điểm và centroid
- Bước 3 : Cập nhật centroid mới, giá trị của centroidi mới bằng giá trị trung bình (means,median,...) của cluster mới
- Bước 4 : Thực hiện lặp lại cho đến khi kết quả hội tụ.

Mô tả các hàm trong chương trình

```
# khởi tạo centroids
def init_centroid(img_1d, k_cluster, init_centroid_type = 'in_pixels'):
    if init_centroid_type == 'in_pixels':
        return img_1d[np.random.choice(img_1d.shape[0], k_cluster, replace= False)]
    elif init_centroid_type == 'random':
        return np.random.choice(256, size = (k_cluster, img_1d.shape[1]), replace=False)
```

1. Ở bước này chúng ta sẽ khởi tạo k centroid, do yêu cầu thực hiện 2 cách khởi tạo khác nhau, ta có như sau :

- Đối kiểu khởi tạo là random, chúng ta sẽ return k centroid với mỗi giá trị là integer thuộc đoạn [0,255], mỗi centroid có số channel của một pixel.
- Đối kiểu khởi tạo là in_pixels, chúng ta sẽ return k centroid với mỗi giá trị thuộc ảnh Có thể sử dụng hàm `numpy.random.choice(replace = False)` để không bị trùng giá trị centroid.

```
def cal_distance(img_1d, centroid):
    #norm-2 between pixel and centroid
    distance = np.linalg.norm(img_1d - centroid[0], axis=1)
    distance = distance.reshape((img_1d.shape[0], 1))
    for i in range(1,centroid.shape[0]):
        temp = np.linalg.norm(img_1d - centroid[i], axis=1)
        temp = temp.reshape((img_1d.shape[0], 1))
        distance = np.concatenate((distance,temp),axis=1)
    #return smallest distance's label centroid for each pixel
    return np.argmin(distance,axis = 1)
```

2. Ở bước thứ hai, với mỗi centroid, chúng ta sẽ thực hiện việc tính độ chênh lệch giữa toàn bộ pixel đến centroid bằng cách dùng hàm `linalg.norm()` , sau đó lưu lại ở một vector có kích thước là `count(pixel) * 1` , lặp lại với k centroid và ghép các vector lại với nhau. Sau cùng hàm numpy.argmin() sẽ trả về index của centroid làm cho độ chênh lệch ít nhất.

- numpy.linalg.norm Document

```
def update_centroid(img_1d, label, k_cluster, channel):
    centroid = np.zeros((k_cluster,channel))
    for k in range(k_cluster):
        #slice cluster k from img_1d
        cluster_k = img_1d[label == k]
        #if cluster have 0 data point -> pass update centroid
        if len(cluster_k) == 0:
            continue
        #update centroid
        centroid[k] = np.mean(cluster_k, axis=0)
    return centroid
```

3. Ở bước thứ ba, chúng ta sẽ cập nhật lại centroid mới từ một cluster đã được tìm ra, với mỗi giá trị label k , chúng ta sẽ trả ra những pixel được gán label k trong ma trận ảnh, sau đó tính giá trị của centroid mới bằng hàm `numpy.mean()` hoặc `numpy.median()` hoặc `numpy.average()` của những pixel có label k đó. Nếu xảy ra trường hợp cluster k không có một pixel nào thì có thể bỏ qua việc cập nhật cluster đó

```
def converge_check(centroid, new_centroid):
    if np.array_equal(centroid, new_centroid):
        return True
    else:
        return False
```

4. Ở bước tiếp theo, chúng ta sẽ xét giá trị của centroid của 2 lần tính toán gần nhất, nếu khoảng cách giữa các giá trị của centroid lần tính trước và centroid vừa tính hiện tại bằng nhau thì giá trị centroid vừa tính xem như đủ tốt để dừng chương trình

```
def update_data_point(img_1d,k_clusters,label,centroids):
    new_img = np.zeros(img_1d.shape)
    for k in range(k_clusters):
        new_img[label == k] = centroids[k]
    return new_img
```

5. Ở bước này, với mỗi pixel có label k thì được thay thế bằng giá trị của centroid của cluster k.

```
def kmeans(img_1d, k_clusters, init_centroids='in_pixels'):
    #flatten array
    max_iter = 1000
    row = img_1d.shape[0]
    column = img_1d.shape[1]
    channel = img_1d.shape[2]
    img_1d = img_1d.reshape(img_1d.shape[0] * img_1d.shape[1], img_1d.shape[2])
    #init centroids
    centroid = [init_centroid(img_1d, k_clusters, init_centroids)]
    label = []
    while True and max_iter:
        #assign label of datapoint
        new_label = cal_distance(img_1d, centroid[-1])
        label.append(new_label)
        new_centroid = update_centroid(img_1d,label[-1],k_clusters, channel)
        if converge_check(centroid[-1],new_centroid) :
            break
        max_iter-=1
        centroid.append(new_centroid)
    new_img = update_data_point(img_1d,k_clusters,label[-1],centroid[-1])
    new_img = new_img.reshape(row,column,channel)
    return centroid[-1], new_img
```

6. việc xử lý một ma trận nhiều chiều khá phức tạp (với ảnh RGB khi đọc vào sẽ có số chiều là 3), do đó chúng ta cần thực hiện việc chuyển ma trận ảnh về 2 chiều như trên. sau đó thực hiện

việc khởi tạo centroid. Tiếp đến thực hiện thuật toán trên lần lượt theo một số lượng lần max_iter cho trước (Ở đây là 1000 lần), hoặc đến khi kết quả hội tụ (điều kiện hội tụ được thể hiện bên dưới).

```
if __name__ == '__main__':
    img = input()
    img_1d = Image.open(img)
    img_1d = np.asarray(img_1d)
    k_clusters = 15
    # =50
    row = img_1d.shape[0]
    column = img_1d.shape[1]
    channel = img_1d.shape[2]
    #init_centroids = 'in_pixels' OR init_centroids = 'random'
    new_centroid, new_img = kmeans(img_1d, k_clusters,init_centroids = 'in_pixels')
    print('new centroid = ', new_centroid)
    print('new img = ', new_img)
    plt.imshow(new_img.astype('uint8'))
    plt.show()
```