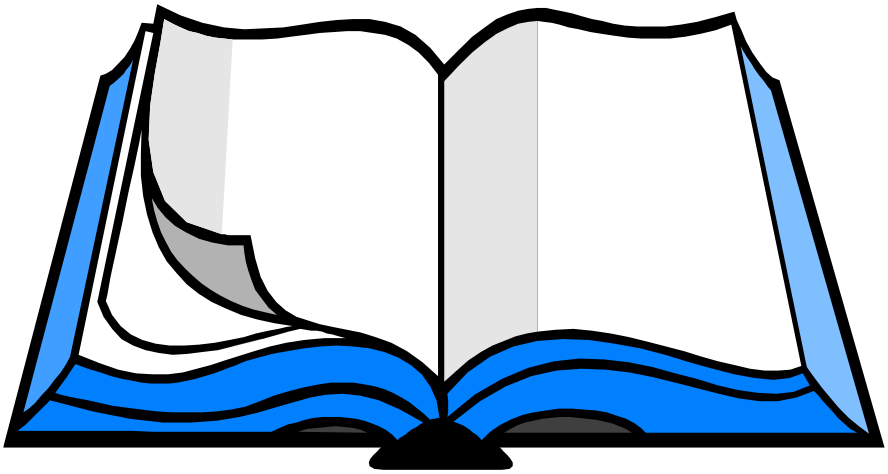
University of Science

Information Technology Department

PROJECT 01: COLOR COMPRESSION



Giảng viên: Nguyễn Văn Quang Huy, Ngô Đình Hy, Nguyễn Đình Thúc

Sinh viên: 21127667 – Trương Công Gia Phát

I. Nén ảnh là gì

Nén ảnh là quá trình giảm kích thước của tệp ảnh để tiết kiệm không gian lưu trữ và tăng tốc độ truyền dữ liệu. Trong quá trình nén, kích thước của tệp ảnh được giảm bằng cách loại bỏ hoặc giảm thiểu dữ liệu không cần thiết hoặc dùng các phép biến đổi số học để tạo ra một phiên bản ảnh gốc có chất lượng tương tự nhưng với kích thước nhỏ hơn.

II. Áp dụng giải thuật K-means vào việc nén ảnh

1. Giải thuật K-means là gì

Giải thuật K-means là một thuật toán phân cụm (clustering) dựa trên việc tìm ra các trung tâm (centroids) đại diện cho các nhóm trong một tập dữ liệu. Thuật toán K-means là một trong những thuật toán phân cụm phổ biến nhất trong lĩnh vực học máy và khai phá dữ liệu.

Quy trình chính của thuật toán K-means như sau:

1. **Khởi tạo trung tâm (centroid):** Ban đầu, chúng ta chọn ngẫu nhiên K điểm trong dữ liệu làm trung tâm ban đầu cho K cụm (cluster).
2. **Gán dữ liệu vào nhóm gần nhất:** Đối với mỗi điểm dữ liệu, chúng ta tính toán khoảng cách giữa điểm đó và các trung tâm và gán điểm đó vào nhóm có trung tâm gần nhất.
3. **Cập nhật trung tâm:** Sau khi gán các điểm dữ liệu vào từng nhóm, chúng ta tính toán lại trung tâm của mỗi nhóm bằng cách lấy trung bình của các điểm trong nhóm đó.
4. **Lặp lại bước 2 và 3:** cho đến khi trung tâm không thay đổi hoặc thay đổi rất ít hoặc đạt được số lần lặp tối đa (iteration).

2. Ý tưởng thực hiện nén ảnh sử dụng K-means

Đối với việc nén ảnh, tập dữ liệu ở đây là tập hợp các điểm pixel trong ảnh và mỗi pixel được biểu diễn bằng một vecto chứa giá trị các màu sắc.

Như vậy các bước trong việc sử dụng giải thuật K-means để nén ảnh vẫn sẽ là:

1. **Khởi tạo các trung tâm (centroid):** Đầu tiên ta sẽ chọn K màu mà chúng ta muốn trong ảnh, số lượng màu sẽ tương đương với số các cụm (cluster). Tiếp theo, chúng ta sẽ chọn ngẫu nhiên K vị trí trong tập hợp các pixel để làm trung tâm cho K cụm.
2. **Gán pixel vào cụm (cluster) gần nhất:** Đối với mỗi pixel trong ảnh, tính toán vị trí của nó tới các centroid và gán điểm đó vào centroid gần nhất.
3. **Cập nhật lại các trung tâm (centroid):** Sau khi gán các pixel, chúng ta tính toán lại các centroid bằng cách tính trung bình của các pixel trong cụm đó.

4. **Lặp lại bước 2 và 3** cho đến khi các centroid không còn thay đổi hoặc đến khi đạt được số lần lặp (iteration) tối đa.
5. **Khôi phục lại ảnh vừa được nén:** Ảnh bị nén sẽ có chất lượng hình ảnh thấp hơn ảnh gốc vì giải thuật K-means đã làm giảm số lượng màu sắc trong ảnh để giảm kích thước tệp ảnh

III. Sử dụng K-means để nén ảnh trong Python

Hàm đọc ảnh:

```
def read_image(im_path):
    img = plt.imread(im_path)
    img = img / 255.0
    return img
```

Đọc ảnh bằng matplotlib.pyplot.imread(), kết quả sẽ trả về một numpy array. Tiếp theo ta chuẩn hoá bằng cách chia tất cả pixel cho 255.0.

Hàm tạo ngẫu nhiên các trung tâm (centroid):

```
def initialize_centroid(img, k_clusters):
    w, h, d = img.shape
    new_size = w * h
    imArr = img.reshape(new_size, d)
    centroid = np.zeros((k_clusters, d))
    for i in range(k_clusters): #assigning random centroids
        centroid[i] = np.mean(imArr[np.random.choice(new_size, size=10, replace=False)], axis=0)
    return imArr, centroid
```

Tiếp theo ta biến đổi ma trận ảnh img thành một ma trận 2 chiều bằng **img.reshape(new_size, d)** với **new_size** là tổng các điểm ảnh và **d** là số kênh màu. Ta tạo một ma trận centroid có giá trị ban đầu là các số 0. Với mỗi vòng lặp for hàm np.random.choice() sẽ chọn ngẫu nhiên 10 số không trùng lặp trong khoảng các điểm ảnh và ta sẽ tính giá trị trung bình của các điểm ảnh đó bằng np.mean() và gán nó vào từng centroid một cách ngẫu nhiên.

Hàm tính khoảng cách:

```
def euclidean_distance(a1, b1, a2, b2):
    d = np.square(a1 - a2) + np.square(b1 - b2)
    return np.sqrt(d)
```

Hàm sẽ trả về khoảng cách giữa 2 điểm ảnh.

Hàm K-means:

```
def k_means(imArr, centroid, max_iter, k_cluster):
    im_size = imArr.shape[0]
    index = np.zeros(im_size)
    for t in range(max_iter):
        for j in range(im_size):
            distance = float('inf')
            for k in range(k_cluster):
                x1, y1 = imArr[j, 0], imArr[j, 1]
                x2, y2 = centroid[k, 0], centroid[k, 1]
                if euclidean_distance(x1, y1, x2, y2) <= distance:
                    distance = euclidean_distance(x1, y1, x2, y2)
                    index[j] = k

        for k in range(k_cluster):
            cluster_points = imArr[index == k]
            if len(cluster_points) > 0:
                centroid[k] = np.mean(cluster_points, axis=0)

    return centroid, index
```

Hàm K-means sẽ nhận vào các tham số tương ứng là **imArr** (ma trận hình ảnh numpy), **centroid** (mảng chứa các centroid ngẫu nhiên đã được tạo trước đó), **max_iter** (số lần lặp tối đa) và **k_cluster** (số các cụm). Đầu tiên ta sẽ lấy kích cỡ của ma trận ảnh **imArr** và tạo mảng **index** để lưu lại cluster của từng điểm ảnh. Tiếp theo vòng lặp for trong khoảng max_iter xác định số lần tối đa ta cập nhật lại các **centroid**. Tiếp đến ta xác đinh từng điểm ảnh sẽ ứng với cluster nào bằng cách duyệt qua từng điểm ảnh và tính khoảng cách của nó đến các centroid bằng hàm **euclidean_distance** sau đó gán nó vào các cluster ứng với centroid gần nhất, ta dung mảng index để tra xem điểm ảnh thuộc vào cluster nào. Sau khi hoàn thành vòng lặp gán điểm dữ liệu vào từng cụm, vòng lặp **for k in range(k_cluster)** duyệt qua từng cụm và tính toán lại trung tâm của mỗi cụm bằng cách lấy trung bình của các điểm trong cụm đó.

Hàm nén ảnh:

```
def image_compression(centroid, index, img):

    klusters = np.array(centroid*255.0, dtype = np.uint8)
    new_image = klusters[index.astype(int), :]
    new_image = new_image.reshape(img.shape)
    return new_image
```

Đầu tiên nhân lại mảng centroid với 255.0 để chuyển đổi các giá trị pixel về với khoảng giá trị đúng của ảnh. **klusters[index.astype(int), :]** sử dụng mảng **index** để chọn các centroid tương ứng với mỗi pixel trong quá trình nén. Dòng lệnh sẽ trả về mảng **new_image** chứa các giá trị pixel ứng với mỗi centroid. Sau đó ta định hình lại ảnh trở lại như kích cỡ gốc bằng hàm reshape.

Hàm main:

```
if __name__ == '__main__':
    image_path = str(input("Nhập đường dẫn ảnh: "))
    img = read_image(image_path)

    clusters = int(input("Nhập số màu bạn muốn: "))
    image_format = str(input("Nhập định dạng ảnh mà bạn muốn lưu: "))
    img_1d, centroid = initialize_centroid(img,clusters)
    centroid, index = k_means(img_1d, centroid, 10,  clusters)
    new_image = image_compression(centroid, index, img)


    picture = im.fromarray(new_image, 'RGB')
    picture.save('output.' + image_format)

    plt.imshow(new_image)
    plt.show()
```

Ta sẽ lần lượt nhập vào đường dẫn ảnh để đọc ảnh, số màu có trong ảnh và định dạng ảnh của ảnh nén. Sau đó xuất ảnh nén ra màn hình bằng **matplotlib.pyplot.imshow()**.

IV. Kết quả

Ảnh gốc:

Ảnh được nén với K = 3:

Ảnh được nén với K = 5:

Ảnh được nén với K = 7:

Ảnh được nén với K = 10:

Như vậy ta có thể thấy được, khi ta gia tăng số lượng màu lên, hình ảnh càng trở nên rõ nét và các chi tiết sẽ càng giống như ảnh gốc.

V. Nguồn tham khảo

https://www.geeksforgeeks.org/image-compression-using-k-means-clustering/

https://www.youtube.com/watch?v=shu4pYQb-ps&t=177s

https://towardsdatascience.com/image-compression-using-k-means-clustering-aa0c91bb0eeb

http://tutorials.aiclub.cs.uit.edu.vn/index.php/2021/09/08/kmeans-clustering/