



ĐỒ ÁN 1: COLOR COMPRESSION

21127528 - Nguyễn Thị Minh Minh

Ngày 16 tháng 7 năm 2023

Giáo viên hướng dẫn:

Phan Thị Phương Uyên Vũ Quốc Hoàng

Mục lục

1	Ý tưởng thực hiện	2
1.1	Giới thiệu chung:	2
1.2	Xử lý ma trận ảnh	2
1.3	Thuật toán K-means:	2
1.4	Các thư viện đã dùng:	2
2	Mô tả các hàm	3
2.1	Hàm <code>get_image_1d(image_path)</code> :	3
2.2	Hàm <code>get_original_dimension(img_1d, row, column, num_channel = 3)</code> :	3
2.3	Hàm <code>init_centroid(img_1d, k_cluster, method = "random")</code> :	3
2.4	Hàm <code>label_pixel(img_1d, centroid)</code> :	4
2.5	Hàm <code>update_centroid(img_1d, centroid, k_cluster, label)</code> :	5
2.6	Hàm <code>assign_new_color(img_1d, centroid, label, k_cluster)</code> :	5
2.7	Hàm <code>kmeans(img_1d, k_cluster, max_iter, init_centroids="random")</code> :	6
2.8	Hàm <code>main()</code> :	7
3	Hình ảnh kết quả	7
4	Nhận xét kết quả	12
	Tài liệu tham khảo	13

1 Ý tưởng thực hiện

1.1 Giới thiệu chung:

Để thực hiện bài toán giảm số lượng màu để biểu diễn ảnh sao cho nội dung ảnh vẫn đảm bảo nhất có thể, em sử dụng thuật toán K-means. Ở thuật toán này, chúng ta cần phải phân loại các pixel màu vào k cụm (`k_clusters`) sao cho mỗi cụm chứa các pixel màu có tính chất giống nhau và `centroid` của mỗi cụm chính là màu trung hòa giữa tất cả pixel trong cùng 1 cụm. Thực hiện dán nhãn (`label_pixel`) và cập nhật các centroid (`update_centroid`) cho đến khi hội tụ, thì ta dừng lại và thay đổi màu của các pixel với các `label` đã gán.

1.2 Xử lý ma trận ảnh

Dầu tiên, để có thể sử dụng thuật toán k-means giảm số lượng màu ảnh, em sẽ chuyển ảnh từ 2 dimensions thành 1 dimensions. Ở ma trận 1d mới này, ta có số hàng chính là tổng số lượng pixel có trong ảnh và sẽ có 3 giá trị đại diện cho bộ ba số tương ứng (R, G, B).

1.3 Thuật toán K-means:

- Bước 1: Chọn ra `k centroids` với `k` cho trước bằng 2 phương pháp `random` hoặc `in_pixels`.
- Bước 2: Tiến hành dán nhãn (`label_pixel`):

Tính khoảng cách Euclidean từ mỗi pixel đến `k centroids` đã tạo.

Sau đó, chọn ra centroid có khoảng cách Euclidean với pixel là ngắn nhất và dán label để phân ra thành `k cluster`.

- Bước 3: Cập nhật `centroid`:

Chọn ra các `centroid` mới bằng cách tính trung bình ma trận của các pixel thuộc cùng 1 `cluster`.

- Bước 4: Lặp lại bước 2 và bước 3 cho đến khi hội tụ, nghĩa là các centroid được cập nhật không có thay đổi đáng kể so với centroid trước đó.
- Bước 5: Tiến hành thay đổi màu của mỗi pixel với label đã gán với mỗi pixel. Chuyển lại chiều gốc của ảnh để lấy ảnh đã được xử lý.

1.4 Các thư viện đã dùng:

- `numpy`: Dùng để thực hiện tính toán trên ma trận.
- `PIL`: Dùng để đọc ghi ảnh.
- `matplotlib`: Dùng để hiển thị ảnh
- `OS`: Dùng để chuyển đổi file được xuất ra.
- `time`: Dùng để đo thời gian chạy của thuật toán kmeans.

2 Mô tả các hàm

2.1 Hàm `get_image_1d(image_path)`:

```
def get_img_1d(image_path):
    image = Image.open(image_path)
    img_1d = np.array(image)
    img_1d = img_1d.reshape(img_1d.shape[0] * img_1d.shape[1], img_1d.shape[2])
    return img_1d
```

Hình 1: Hàm chuyển ảnh về ma trận 1 chiều với các phần tử màu.

Hàm `get_image_1d` được truyền vào đường dẫn ảnh `image_path`. Sau đó dùng `np.array` để đọc ảnh thành dạng ma trận. Sau đó dùng hàm `np.reshape` để chuyển ma trận về dạng ma trận 2 chiều với:

- Chiều thứ nhất: là tích của chiều thứ nhất và chiều thứ hai của ma trận gốc.
`img_1d.shape[0]*img_1d.shape[1]`
- Chiều thứ hai: là chiều thứ ba của ma trận gốc.
`img_1d.shape[2]`

Lúc này, ta sẽ được ma trận có mỗi dòng đại diện cho 1 pixel với 3 giá trị biểu diễn cho bộ 3 (R, G, B) và số phần tử của của dãy là tổng số pixel của hình.

2.2 Hàm `get_original_dimension(img_1d, row, column, num_channel = 3)`:

```
def get_original_dimension(img_1d, row, column, num_channel = 3):
    orig_img = img_1d.reshape(row, column, num_channel)
    return orig_img
```

Hình 2: Hàm chuyển về hình ảnh để hiển thị.

Hàm này sẽ được sử dụng sau khi xử lý xong toàn bộ ma trận ảnh theo đề bài để lấy lại ma trận ảnh gốc trước khi xuất ra kết quả.

2.3 Hàm `init_centroid(img_1d, k_cluster, method = "random")`:

```
def init_centroid(img_1d, k_cluster, method = "random"):
    centroid = np.empty((k_cluster, img_1d.shape[1]), dtype = img_1d.dtype)
    if method == "random":
        return np.random.choice(256, size = (k_cluster, img_1d.shape[1]), replace=False)
    elif method == "in_pixels":
        for i in range(k_cluster):
            centroid[i] = img_1d[np.random.choice(img_1d.shape[0], replace = False)]
    return centroid
```

Hình 3: Hàm thiết lập centroid.

Ở hàm này, 3 tham số được truyền vào:

- `img_1d`: dãy pixel của hình ảnh.
- `k_cluster`: số cluster được nhập.
- `method`: random hoặc in-pixels.

Hai phương pháp:

- `random`: dùng hàm `np.random.choice` để random giá trị cho 3 cột (R, G, B) (`img_1d.shape[1]`) trong khoảng từ [0, 256], `replace= False` để giá trị random sẽ đảm bảo khác nhau, không bị lặp lại.
- `in_pixels`: cho chạy 1 vòng lặp từ 0 đến `k_cluster-1` và gán random cho mỗi centroid là 1 pixel từ `img_1d`, `replace=False` để giá trị random sẽ đảm bảo khác nhau, không bị lặp lại.

2.4 Hàm `label_pixel(img_1d, centroid)`:

```
def label_pixel(img_1d, centroid):
    #parameter centroid is a array: [[a b c] [d e f]]
    #calculate Euclidean distance
    # np.linalg.norm in numpy to calculate

    distances = np.linalg.norm(centroid[:, np.newaxis] - img_1d, axis = 2)
    labels = np.argmin(distances, axis = 0)
    return labels
```

Hình 4: Hàm dán nhãn pixel.

Ở hàm này, em sẽ tính khoảng cách Euclidean bằng cách sử dụng hàm `np.linalg.norm`.

- `distances = np.linalg.norm(centroid[:, np.newaxis] - img_1d, axis = 2)`

Nhiệm vụ chính của câu lệnh trên chính là tính khoảng cách từ mỗi phần tử trong mảng `centroid[]` đến mỗi phần tử trong mảng `img_1d`. Trong phần `centroid[:, np.newaxis]-img_1d`, em đã tiến hành sử dụng phép `broadcasting` để thực hiện mở rộng mảng để tương thích trong việc thực hiện phép trừ, sau đó hàm sẽ đi tính độ lớn norm của `centroid[:, np.newaxis] -img_1d`. Kết quả, lúc này sẽ trả về một mảng mới là `distances`, mảng này sẽ lưu kết quả tính khoảng cách Euclidean của từng phần tử pixel với từng centroids. Việc thực hiện phép trừ đã giảm thời gian chạy của phép tính di rất nhiều so với sử dụng phép lặp.

- `labels = np.argmin(distances, axis = 0)`

Lúc này, chúng ta sẽ tiến hành tìm chỉ số của phần tử có giá trị Euclidean nhỏ nhất trong mảng `distances` theo cột (`axis = 0`) với hàm `np.argmin`.

2.5 Hàm update_centroid(img_1d, centroid, k_cluster, label):

```
def update_centroid(img_1d, centroid, k_cluster, label):
    updated_centroid = np.empty((k_cluster, img_1d.shape[1]))
    for i in range(k_cluster):
        pixels_in_cluster = img_1d[label == i]
        if pixels_in_cluster.shape[0] > 0:
            updated_centroid[i] = np.mean(pixels_in_cluster, axis=0)
    return updated_centroid
```

Hình 5: Hàm cập nhật centroid.

Ở hàm này, em sẽ cho chạy vòng lặp for từ 0 đến $k_cluster-1$ sau đó sẽ dùng 1 mảng biến mới `pixels_in_cluster` để lưu tất cả các pixel trong `img_1d` mà có `label` bằng với chỉ số của `centroid`. Nghĩa là, lúc này ta sẽ phân cụm pixel với `label` đã gán ở bước trên. Tiếp theo, dùng hàm `np.mean` để tính giá trị trung bình của các centroid trong cùng 1 cụm và lưu lại giá trị trung bình mới đó trong mảng trả về là `updated_centroid`.

2.6 Hàm assign_new_color(img_1d, centroid, label, k_cluster):

```
def assign_new_color(img_1d, centroid, label, k_cluster):
    for i in range(k_cluster):
        idx = np.where(label == i)
        img_1d[idx] = centroid[i]

    return img_1d
```

Hình 6: Hàm cập nhật màu mới cho pixel.

Trong hàm này, ta tiến hành gán lại cho mỗi pixel trong mảng `img_1d` có `label = i` bằng màu của `centroid[i]` tương ứng.

2.7 Hàm `kmeans(img_1d, k_cluster, max_iter, init_centroids="random")`:

```

def kmeans(img_1d, k_clusters, max_iter, init_centroids="random"):

    centroid = init_centroid(img_1d, k_clusters, init_centroids)
    label = label_pixel(img_1d, centroid)
    pre_centroid = centroid.copy()
    converge_cnt = 0

    while converge_cnt < max_iter:
        centroid = update_centroid(img_1d, centroid, k_clusters, label)
        if np.allclose(pre_centroid, centroid, atol = 1e-05):
            print("Converge after ",converge_cnt," iterations")
            break

        pre_centroid = centroid.copy()
        label = label_pixel(img_1d, centroid)
        converge_cnt += 1

    if converge_cnt == max_iter:
        print("Image stop iterating after ",converge_cnt, " - max_iter.")

    res_arr = assign_new_color(img_1d, centroid, label, k_clusters)
    return res_arr

```

Hình 7: Hàm KMeans.

Hàm `kmeans` là hàm chính giúp chúng ta thực hiện thuật toán để giảm màu ảnh.

- Đầu tiên, chúng ta sẽ thiết lập `k centroids` đầu tiên bằng cách chọn ngẫu nhiên `in_pixels` hoặc là `random`, dán nhãn cho `centroid` và tạo 1 mảng mới có tên là `pre_centroid` để copy lại mảng `centroid` đã tạo/ update.
- Chạy vòng lặp với điều kiện dừng lại hoặc là đã đạt đến `max_iter` hoặc là đã hội tụ.
- Để kiểm tra hội tụ sau mỗi vòng lặp, em dùng hàm `np.allclose` để tiến hành so sánh mảng `centroid` mới cập nhật và mảng `pre_centroid` để check hội tụ. Và hai mảng này sẽ được so sánh một cách tương đối trong khoảng sai số tuyệt đối `atol = 1e-05`.
- Sau khi thoát vòng lặp, em tiến hành gán lại màu mới cho pixel của mảng `img_1d` với hàm `assign_new_color` và lưu lại trong mảng kết quả `res_arr`.

2.8 Hàm main():

```

def main():
    #get input picture
    img_path = input("Enter your image path: ")

    #show original picture
    image = plt.imread(img_path)
    image_arr = np.array(image)
    plt.figure(figsize=(12, 12))
    plt.subplot(2,2,1)
    plt.imshow(image_arr)
    plt.title("Original image",size = 9)

    #get output format
    #get the new output_path with original file name with new output format.
    img_name = os.path.basename(img_path).split('.')[0]
    output = input("Enter your output format (png, pdf): ")
    output_path = f"./{img_name}.{output}"

    #change matrix of picture and compress color
    img_1d = get_img_1d(img_path)
    max_iter = 200
    k_clusters = 7
        # record start time
    start = time.time()
    kmeans(img_1d, k_clusters, max_iter, init_centroids="in_pixels")
    result_picture = get_original_dimension(img_1d, image_arr.shape[0], image_arr.shape[1], image_arr.shape[2])
        #record end time
    end = time.time()
    print("The time of execution of kmeans algorithm is :",end-start, "s")

    #show picture after being compressed
    plt.subplot(2,2,2)
    plt.imshow(result_picture)
    title_res = "Compressed picture with " + str(k_clusters) + " clusters"
    plt.title(title_res,size = 9)

    #export new picture output file
    im = Image.fromarray(result_picture,'RGB')
    im.save(output_path)

```

Hình 8: Hàm main.

Hàm main này sẽ yêu cầu người dùng nhập vào đường dẫn ảnh `img_path` và `output format` là png hoặc pdf. Sau đó sẽ gọi hàm `kmeans` để thực hiện yêu cầu bài toán và trả lại kết quả.

3 Hình ảnh kết quả

1. Hình ảnh 1:

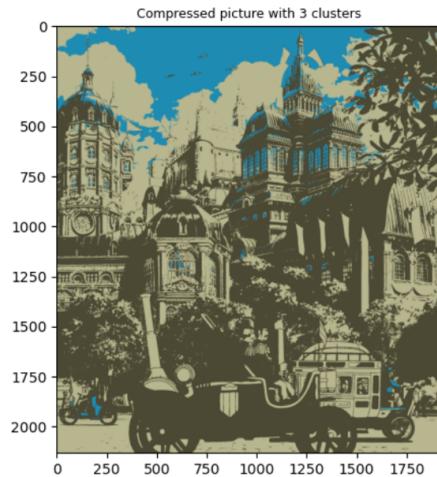
- Size info:

1920 x 2133 723.9 KB 96 dpi 24 bit



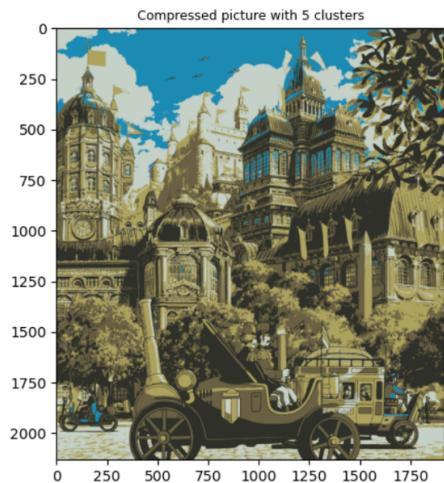
(a) Original picture

```
Enter your image path: glibi.jpg
Enter your output format (png, pdf): png
Converge after 19 iterations
The time of execution of kmeans algorithm is : 44.89142346382141 s
```



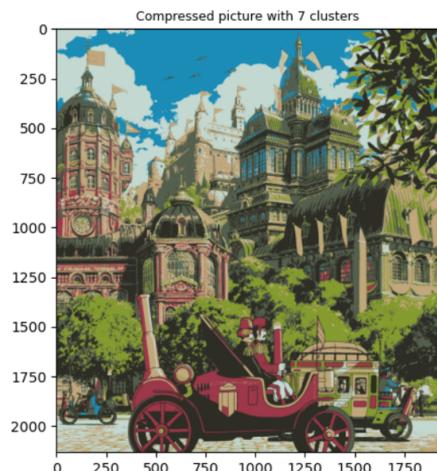
(b) 3 clusters

```
Enter your image path: glibi.jpg
Enter your output format (png, pdf): pdf
Converge after 72 iterations
The time of execution of kmeans algorithm is : 94.88311100006104 s
```



(c) 5 clusters

```
Enter your image path: glibi.jpg
Enter your output format (png, pdf): pdf
Converge after 56 iterations
The time of execution of kmeans algorithm is : 100.68844866752625 s
```



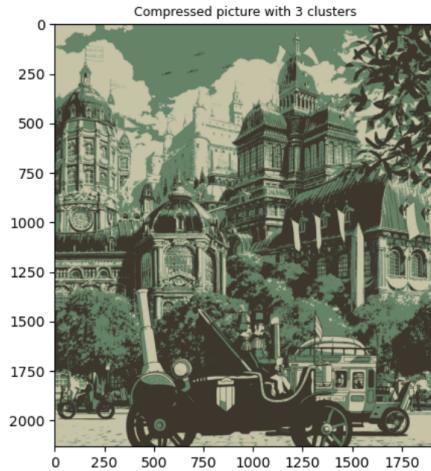
(d) 7 clusters

Hình 9: KMeans - In Pixel method



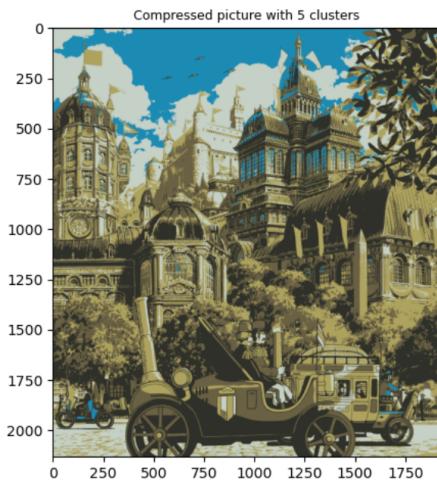
(a) Original picture

```
Enter your image path: glibi.jpg
Enter your output format (png, pdf): png
Converge after 19 iterations
The time of execution of kmeans algorithm is : 16.332508325576782 s
```



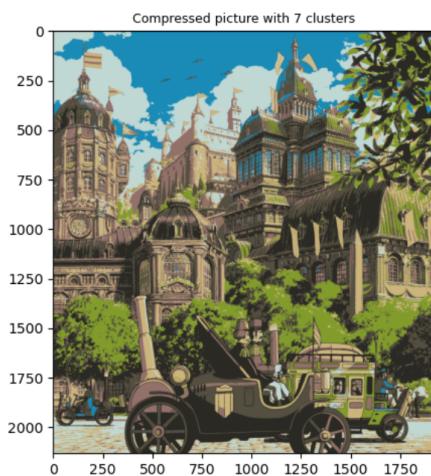
(b) 3 clusters

```
Enter your image path: glibi.jpg
Enter your output format (png, pdf): png
Converge after 65 iterations
The time of execution of kmeans algorithm is : 85.87508225440979 s
```



(c) 5 clusters

```
Enter your image path: glibi.jpg
Enter your output format (png, pdf): png
Converge after 68 iterations
The time of execution of kmeans algorithm is : 113.41300654411316 s
```



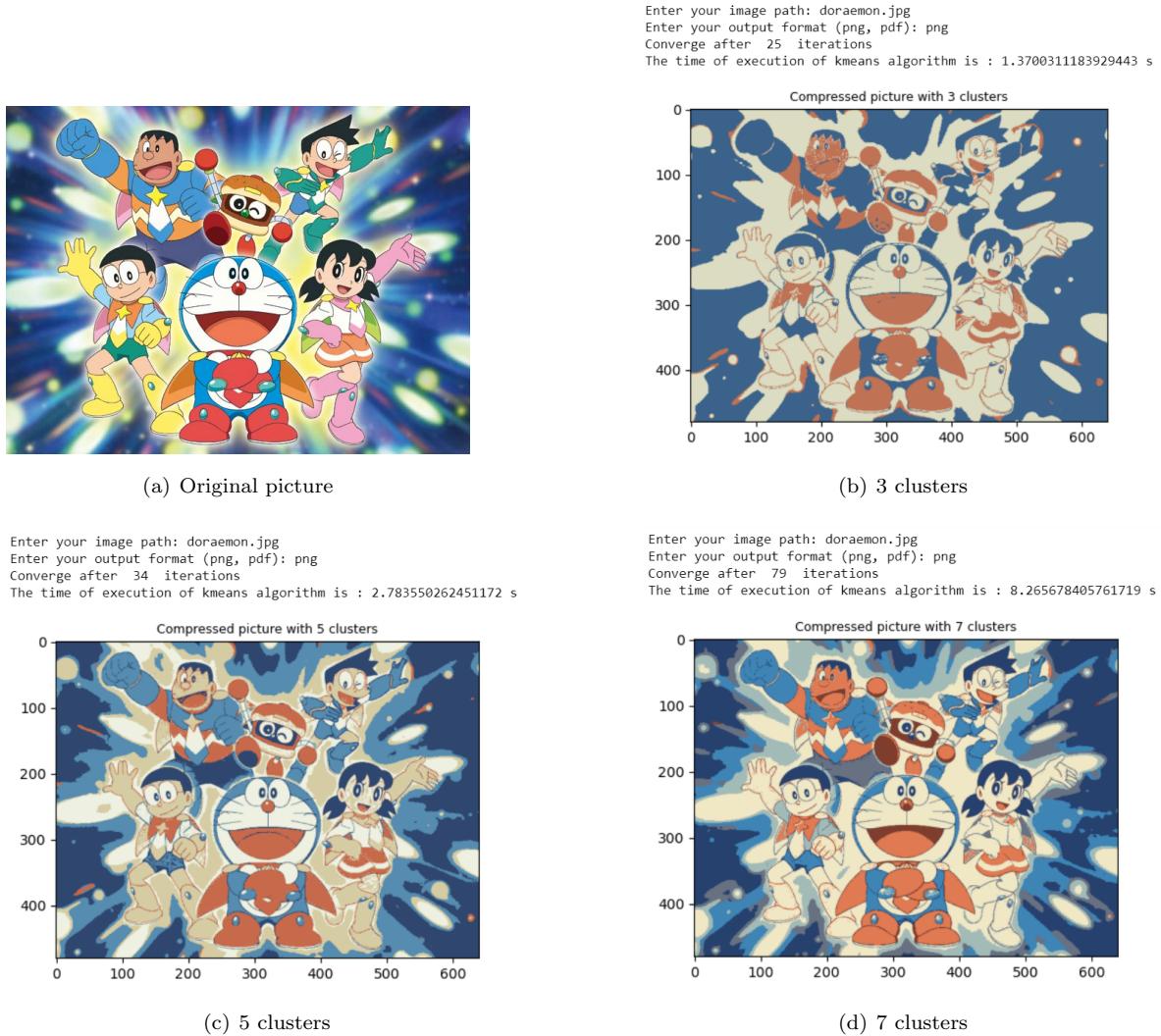
(d) 7 clusters

Hình 10: KMeans - Random method

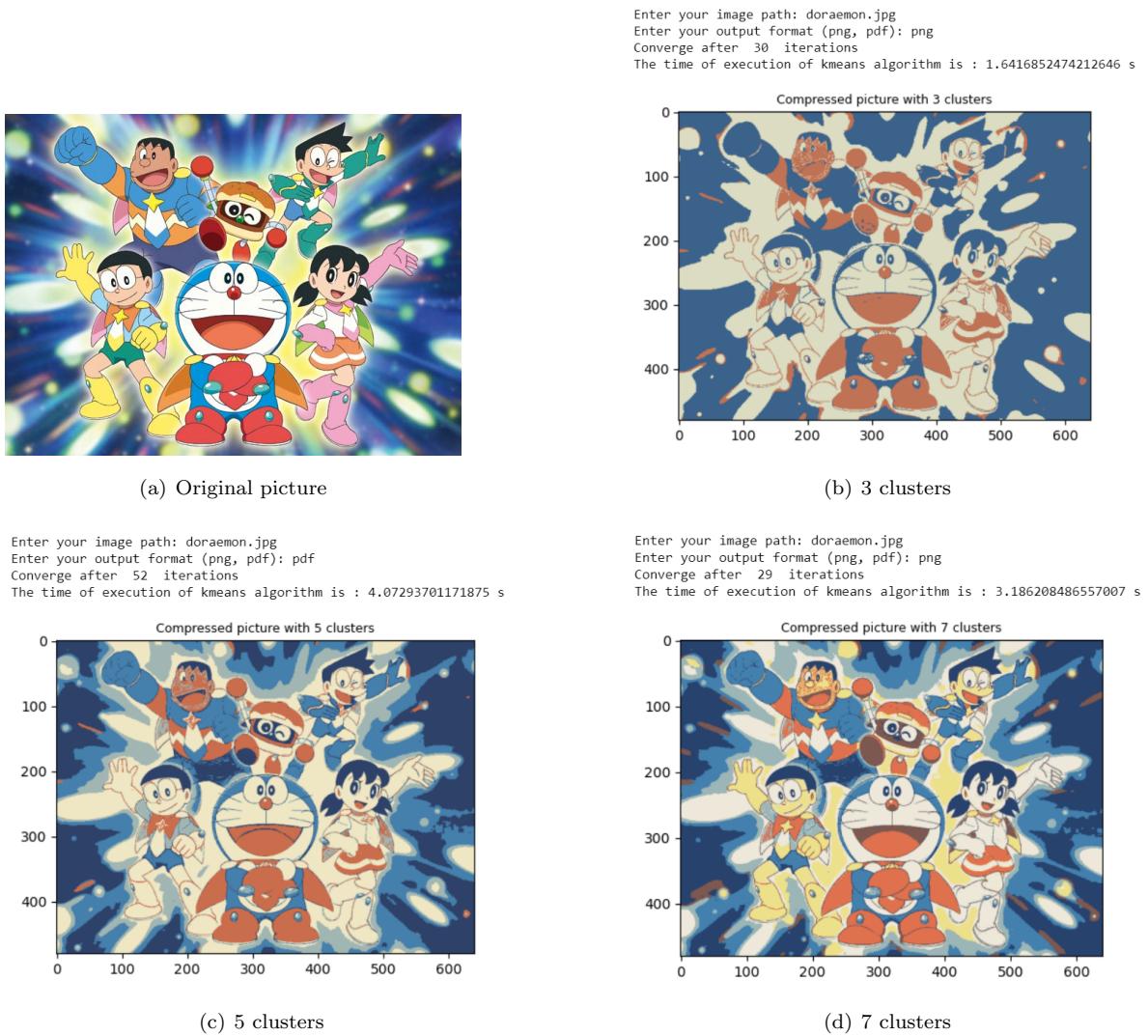
2. Hình ảnh 2:

- Size info:

640 x 480 66 KB 96 dpi 24 bit



Hình 11: KMeans - In Pixel method



Hình 12: KMeans - Random method

4 Nhận xét kết quả

- Đầu tiên, ta có thể thấy rằng thuật toán K-means để giảm số lượng màu ảnh đã được thực hiện khá thành công khi so sánh với thư viện `sklearn`. Số lượng màu ảnh đã được giảm với số clusters cho trước nhưng vẫn giữ được bao quát nội dung của ảnh.
- Với những ảnh có kích thước càng lớn với độ phân giải càng cao thì thuật toán sẽ chạy càng lâu so với những ảnh có kích thước, độ phân giải nhỏ hơn.

Cụ thể với hai hình ảnh ví dụ, với hình ảnh đầu tiên có kích thước 1920 x 2133 thì thời gian chạy thuật toán có thể lên đến hơn 100s. Nhưng với hình ảnh thứ hai với kích thước 640 x 480 thì thời gian thực hiện thuật toán chỉ cỡ khoảng dưới 10s cho các clusters 3, 5, 7.

- Thời gian chạy của thuật toán hầu như sẽ tỉ lệ thuận với số clusters để compress. Nghĩa là, số clusters tăng lên thì thời gian chạy thuật toán cũng lâu hơn, cũng như số bước hội tụ cũng tăng lên. Nhưng trong 1 số tình huống vẫn có trường hợp ngược lại. Cụ thể trong **Hình 12: KMeans - Random** thì với 5 clusters có thời gian chạy thuật toán 4.07s nhưng với 7 clusters thì chỉ có 3.18s.
- Với số clusters càng lớn thì hình ảnh sau khi được compress sẽ rõ nét hơn và đẹp hơn.
- Với hai phương pháp `init centroid` là `random` và `in_pixel` thì hình ảnh được cho ra đều khá tương tự nhau và thời gian chạy cũng tương đối xấp xỉ nhau.

Tài liệu

- [1] Image Segmentation using K Means Clustering.
Available at: <https://www.geeksforgeeks.org/image-segmentation>
- [2] K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks.
Available at: <https://towardsdatascience.com/kmeans-clustering>
- [3] How to calculate Euclidean distance.
Available at: <https://stackoverflow.com/calculate-euclidean-numpy>
- [4] Kieu Vu Minh Duc Github, May 3 2021.
Available at: <https://github.com/kvmduc/applied-math>
- [5] Ngoc Tien Github, Sep 20 2022.
Available at: <https://github.com/NgocTien0110/Applied-Mathematics-and-Statistics>
- [6] Numpy Library.
Available at: <https://numpy.org/>
- [7] Save file with different extension. Available at: <https://stackoverflow.com/save-file-extension>