TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG CÔNG THÔNG TIN

BÁO CÁO ĐỒ ÁN 1

Color Compression

Chủ đề: Toán ứng dụng – thống kê

Lớp: 22CLC01

Sinh viên: **Trương Thuận Kiệt – 22127224**

TP. Hồ Chí Minh, tháng 6 năm 2024

Mục lục

1. Ý tưởng thực hiện

a. Hàm K means

- **Mã giả**

```
function kmeans(img_1d, k_clusters, max_iter, init_centroids):
    Get height_width and num_channels from img_1d
    Initialize centroids based on init_centroids method
    Initialize labels array with zeros

    for each iteration in max_iter:
        Calculate distances between each pixel and centroids
        Assign labels to each pixel based on nearest centroid
        Initialize new centroids array

        for each cluster in k_clusters:
            Calculate mean of all pixels in the cluster
            Set new centroid to the mean

        if new centroids equal centroids:
            Break loop

        Update centroids and labels

    Return centroids and labels
```

- **Các bước chi tiết**

+ Step 1: Khởi tạo centroids, nếu init_centroid là **random** thì centroids sẽ là mảng 2 chiều gồm **num_channels (số lượng kênh màu) cột** và **k_clusters (k cụm điểm) dòng** chứa các giá trị ngẫu nhiên từ 0 – 255 cho mỗi cluster, nếu init_centroid là **in_pixels** thì centroids sẽ là **mảng 2 chiều (k_clusters, num_channels)** được **lấy ngẫu nhiên từ img_1d gồm height*width dòng ra k_clusters dòng** chứa các giá trị điểm ảnh từ hình ảnh gốc, đại diện cho các centroid ban đầu.

+ Step 2: Khởi tạo labels là rỗng, và tạo vòng lặp tới max_iter

+ Step 3: Trong đó, gán nhãn cho mỗi pixel, chúng ta tính khoảng cách Euclidean giữa mỗi pixel và tất cả các centroids. Khoảng cách này thường được tính bằng cách lấy hiệu của các giá trị màu tương ứng của pixel và centroid, và sau đó lấy bình phương và tổng của chúng.

+ **Step 4:** Sau khi tính được khoảng cách từ mỗi pixel đến tất cả các centroids, chúng ta chọn centroid gần nhất (tức là centroid có khoảng cách nhỏ nhất) bằng hàm **argmin** và gán nhãn tương ứng cho pixel đó.
+ **Step 5:** Cập nhật centroids, sau khi mỗi pixel đã được gán nhãn, chúng ta cần cập nhật lại vị trí của centroids bằng cách **tính trung bình của tất cả các pixel trong cùng một nhóm** (cùng một nhãn) bằng hàm **mean**. Kết quả là mỗi centroid mới sẽ di chuyển đến vị trí trung bình của các pixel trong nhóm đó.
+ **Step 6:** Kiểm tra sự hội tụ, sau khi cập nhật centroids, chúng ta kiểm tra xem liệu các centroids mới có thay đổi nhiều so với các centroids cũ hay không. Nếu không có sự thay đổi đáng kể, chúng ta coi thuật toán đã hội tụ và dừng lại. Cách thức kiểm tra này thường là so sánh các centroids mới với các centroids cũ.
+ **Step 7:** Cuối cùng gán lại centroids mới và labels mới và lặp lại step 3

2. Ý nghĩa từng hàm

a. Read_img

```
function read_img(img_path):
    Open image from img_path
    Convert image to NumPy array
    Return NumPy array
```

- Đọc file và trả về mảng numpy của hình thô
- Ví dụ giá trị hàm trả về:
  Input: 'path/to/image.jpg'
  Output: [[R, G, B], [R, G, B], ...] (2D array representing the image)

b. Show_img

```
function show_img(img_2d):
  Use matplotlib to display img_2d
```

- Hiển thị ảnh đã đọc với mảng numpy của hình thô lấy được từ hàm read_img ở trên với chiều x, y
- Ví dụ:
- **Input**: [[R, G, B], [R, G, B], ...] (2D array representing the image)
  img_2d:
  [[[173 171 211]
  [171 169 209]
  [172 170 210]
  ...
  [165 161 198]
  [161 156 196]
  [165 160 200]]

  [[183 181 221]
  [180 178 218]
  [180 178 218]

```
 ...
 [168 163 203]
 [161 156 196]
 [163 158 198]]

[[186 184 224]
 [183 181 221]
 [183 181 221]
 ...
 [172 167 208]
 [162 157 198]
 [162 157 198]]

...

[[ 45 116  14]
 [ 46 118  16]
 [ 49 120  18]
 ...
 [ 49 121  13]
 [ 49 121  13]
 [ 49 121  13]]

[[ 38 106   7]
 [ 38 108   9]
 [ 43 111  12]
 ...
 [ 46 118  10]
 [ 46 118  10]
 [ 46 118  10]]

[[ 30  97   0]
 [ 32  99   2]
 [ 35 102   5]
 ...
 [ 40 112   4]
 [ 40 112   4]
 [ 39 111   3]]]
```

Output:

c. Save_img

```
function save_img(img_2d, img_path):
    Convert img_2d from NumPy array to image
    Save image to img_path
```

- Chuyển đổi mảng numpy trở lại thành đối tượng ảnh và lưu ngược lại theo folder của link directory của img_path
- Ví dụ:

  Input:

  Hình ảnh trước khi lưu (img_2d):

  [[[220 218 228]
  [220 218 228]
  [220 218 228]
  ...
  [167 167 113]
  [167 167 113]
  [220 218 228]]

  [[220 218 228]
  [220 218 228]

```
 [220 218 228]
 ...
 [220 218 228]
 [167 167 113]
 [167 167 113]]

[[220 218 228]
 [220 218 228]
 [220 218 228]
 ...
 [220 218 228]
 [167 167 113]
 [167 167 113]]

...

[[ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]
 ...
 [ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]]

[[ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]
 ...
 [ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]]

[[ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]
 ...
 [ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]]]
```

Output:

Hình ảnh sau khi lưu: <PIL.Image.Image image mode=RGB size=800x600 at 0x2762A856F90>

d. Convert_img_to_1d

```
function convert_img_to_1d(img_2d):
    Get height, width, and channels from img_2d
    Reshape img_2d to 1D array (height * width, channels)
    Return 1D array
```

- Dùng để biến đổi mảng có dạng [[[],[]]] thành [[],[]]
- Ví dụ:

  Input:

  [[255, 0, 0], [0, 255, 0]],[[0, 0, 255], [255, 255, 255]]]

  Output:

  [[255, 0, 0],[0, 255, 0],[0, 0, 255],[255, 255, 255]]

e. Kmeans

```
function kmeans(img_1d, k_clusters, max_iter, init_centroids):
    Get height_width and num_channels from img_1d
    Initialize centroids based on init_centroids method
    Initialize labels array with zeros

    for each iteration in max_iter:
        Calculate distances between each pixel and centroids
        Assign labels to each pixel based on nearest centroid
        Initialize new centroids array

        for each cluster in k_clusters:
            Calculate mean of all pixels in the cluster
            Set new centroid to the mean

        if new centroids equal centroids:
            Break loop

        Update centroids and labels

    Return centroids and labels
```

- Đầu tiên, là khai báo centroids dựa trên đầu vào init_centroids bằng **ý tưởng ở** trên được thực hiện ở **bước 1**
- Sau đó, tính toán khoảng cách từ centroid đến mỗi pixel và gán label cho pixel dựa trên centroid gần nhất
- Sau đó, lặp qua k_clusters để tính trung bình của tất cả pixels và xét xem đã có hội tụ điểm màu chưa nếu có thì thoát vòng lặp lớn ở ngoài
- Cuối cùng, cập nhật centroids và labels cho mỗi pixel

f. Generate_2d_img

```
function generate_2d_img(img_2d_shape, centroids, labels):
    Convert labels to 1D image using centroids
    Reshape 1D image back to 2D image based on img_2d_shape
    Return new 2D image
```

- Đầu tiên biến đổi labels ngược lại hình 1D dùng centroids và reshape trở về hình 2D
- Ví dụ

Input:

centroids:
[[167.20890273 167.19716276 113.76564265]
[74.17588809 80.68832116 30.93348928]
[220.09954679 218.03310584 228.34897894]]
labels:
[2 2 2 ... 1 1 1]

Output:

new_img_1d: [[220 218 228]
[220 218 228]
[220 218 228]
...
[74 80 30]
[74 80 30]
[74 80 30]]
new_img_2d: [[[220 218 228]
[220 218 228]
[220 218 228]
...
[167 167 113]
[167 167 113]
[220 218 228]]

[[220 218 228]
[220 218 228]
[220 218 228]
...
[220 218 228]
[167 167 113]
[167 167 113]]

[[220 218 228]
[220 218 228]

```
 [220 218 228]
 ...
 [220 218 228]
 [167 167 113]
 [167 167 113]]

...

[[ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]
 ...
 [ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]]

[[ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]
 ...
 [ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]]

[[ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]
 ...
 [ 74  80  30]
 [ 74  80  30]
 [ 74  80  30]]]
```

g. Main

```
function main():
    # Input the image path
    img_path = input("Enter the image path: ")

    # Check if the path is valid
    check_valid_path(img_path)

    # Read the image
    img_2d = read_img(img_path)
```

```
  # Show the image
  show_img(img_2d)

  # Get the file name from the path
  file_name = get_file_name_without_extension(img_path)

  # Define the number of clusters and input the maximum number ofiterations
     k_clusters = [3, 5, 7]
  max_iter = input("Enter the maximum number of iterations: ")

  # Convert the 2D image to a 1D array
  img_1d = convert_img_to_1d(img_2d)

  # Loop through each number of clusters
  for i from 0 to length of k_clusters:
     print("Using self-made KMeans algorithm")

     # Get the directory of the image path
     save_path = get_directory_path(img_path)

     # Perform K-means clustering
     centroids, labels = kmeans(img_1d, k_clusters[i], max_iter,
'in_pixels')

     # Display the centroids and labels
     print("centroids: \n", centroids)
     print("labels: \n", labels)

     # Generate a new 2D image from the clustering results
     img_2d_new = generate_2d_img(img_2d.shape, centroids, labels)

     # Display the number of clusters
     print("k_clusters: ", k_clusters[i])

     # Show the new image
     show_img(img_2d_new)

     # Input the format to save the image
     save_format = input("Enter the format to save the image: ")

     # Create the save path for the new image
     save_path = create_save_path(save_path, file_name, i,save_format)
     # Print the save path
     print(save_path)
```

```
    # Save the new image
    save_img(img_2d_new, save_path)
```

- Đầu tiên, chúng ta lấy đường dẫn của file ảnh muốn xử lý và kiểm tra xem đuôi ảnh có phải là .png, .jpg và jpeg không, nếu không thì kết thúc chương trình
- Sau đó, đọc file, hiển thị hình ảnh sau khi đọc, nhận vào số lượng max_iter và chuyển ảnh 2D về lại 1D
- Tiếp tục, dùng kmeans để lấy ra được centroid và label, từ centroid và label cs được tiếp tục biến đổi lại thành ảnh 2D và save lại ảnh

h. Check_valid_path

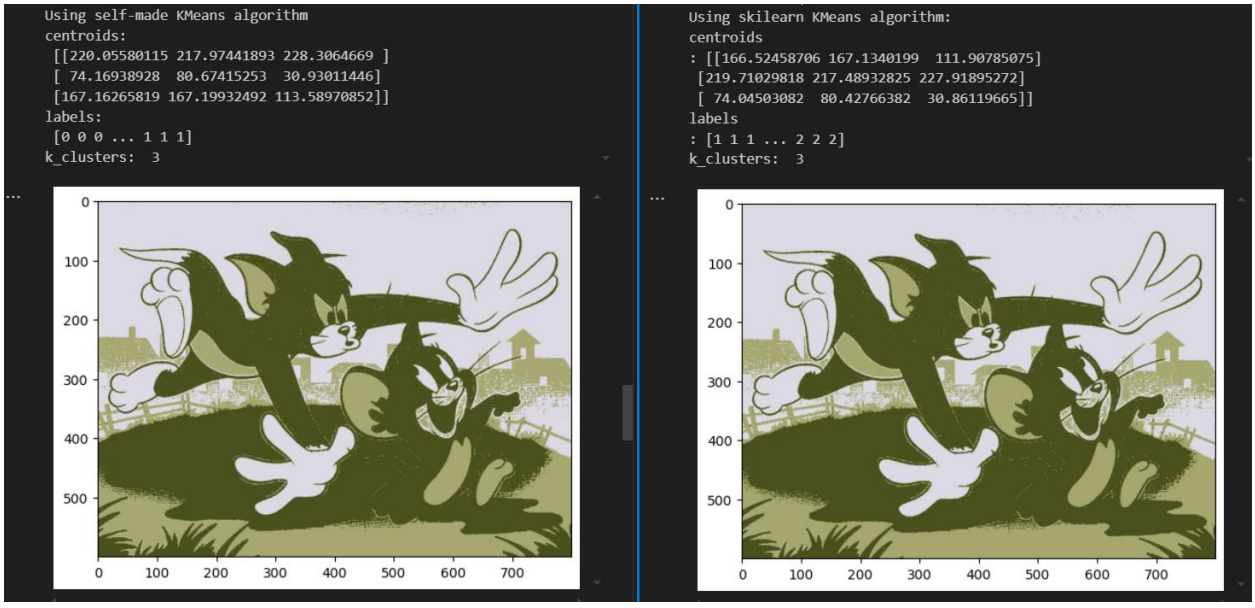
```
function check_valid_path(path):
  # Check if the path ends with .png, .jpg, or .jpeg
  if path ends with '.png' or path ends with '.jpg' or path ends with '.jpeg':
    return True
  else:
    # Raise an error if the file is not .png, .jpg, or .jpeg
    raise ValueError("File is not .png, .jpg, .jpeg")
```

- Dùng để kiểm tra xem link ảnh có đuôi ảnh hay không

3. Testcases

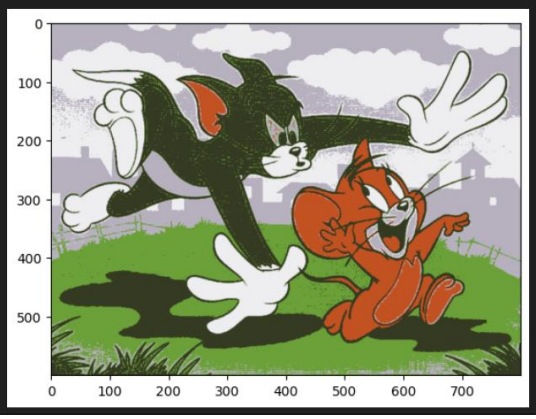
a. Testcase 1 (Using random)

- **K_clusters = 3**

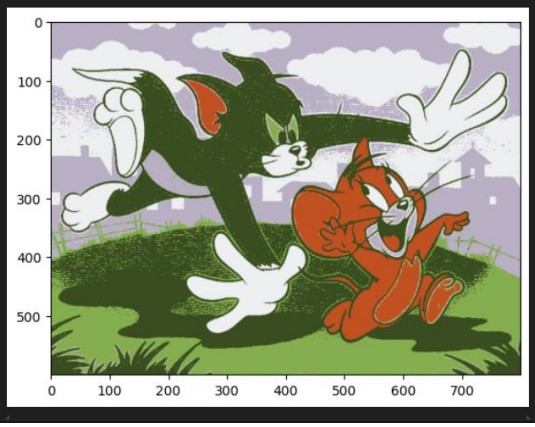


- **K_clusters = 5**

```
Using self-made KMeans algorithm
centroids:
 [[238.27118008 238.58049113 239.90721351]
 [ 53.52669335  58.85896106  34.58707372]
 [183.90133098 176.81506412 191.34408786]
 [195.16698106  82.71351524  35.14875528]
 [107.51848496 161.2140776   56.81803301]]
labels:
 [2 2 2 ... 1 1 1]
k_clusters:  5
```

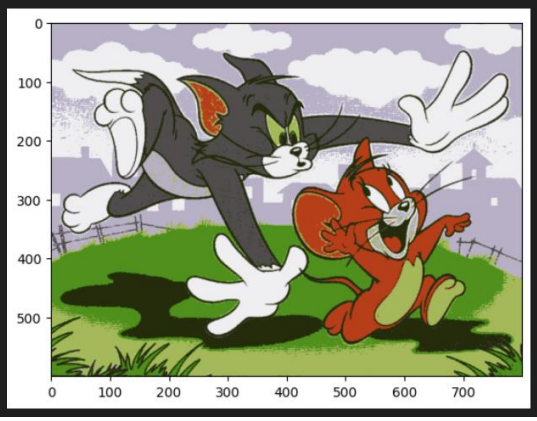



```
Using skilearn KMeans algorithm:
centroids
: [[ 58.07587464  77.85337986  32.97152077]
 [238.86159242 239.152323   240.16813084]
 [185.41281193 176.41673958 197.6829542 ]
 [194.47750101  81.13281271  34.38136529]
 [133.51555207 175.75539219  79.19391906]]
labels
: [2 2 2 ... 0 0 0]
k_clusters:  5
```



- **K_clusters = 7**

```
Using self-made KMeans algorithm
centroids:
 [[165.37390317 185.54558611  91.37870267]
 [182.2174218   63.50725377  26.26901126]
 [ 36.86598381  44.82135782  11.00333236]
 [184.95763487 176.25904834 199.77724237]
 [ 79.04538428 145.97439258  28.4326713 ]
 [239.01687922 239.29384672 240.25060626]
 [ 80.58071891  77.41895449  77.22754606]]
labels:
 [3 3 3 ... 4 4 4]
k_clusters:  7
```

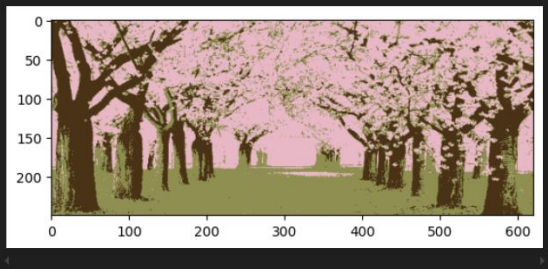



```
Using skilearn KMeans algorithm:
centroids
: [[ 80.59589304  77.43835722  77.24322871]
 [238.90103148 239.17981004 240.17905402]
 [165.61334215 185.60414257  91.43754017]
 [182.13313978  63.44137447  26.19602463]
 [ 36.86432562  44.85703542  11.01212475]
 [184.81539048 176.08941611 199.68333382]
 [ 79.16025835 146.07101158  28.50810267]]
labels
: [5 5 5 ... 6 6 6]
k_clusters:  7
```

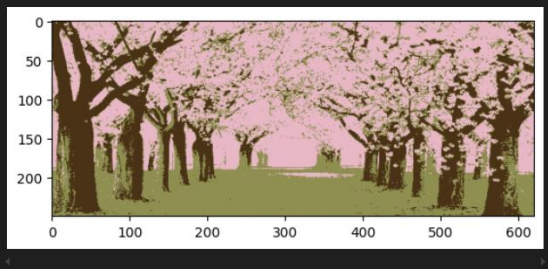
b. Testcase 2 (Using random)

- **K_clusters = 3**

```
Using self-made KMeans algorithm
centroids:
 [[142.16439575 143.05133783  80.53527004]
 [ 73.36960482  50.74406695  22.11778992]
 [232.12986837 184.27234343 198.36388088]]
labels:
 [1 1 1 ... 0 0 0]
k_clusters:  3
```

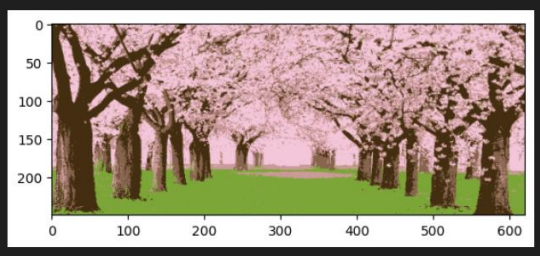



```
Using skilearn KMeans algorithm:
centroids
: [[142.46579431 143.29230496  80.87187444]
 [ 73.5673716   50.93384999  22.2786812 ]
 [232.2594206  184.36494634 198.54631918]]
labels
: [1 1 1 ... 0 0 0]
k_clusters:  3
```

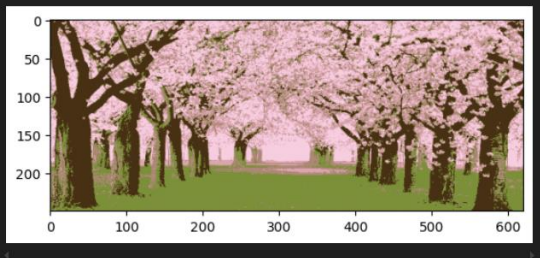


- **K_clusters = 5**

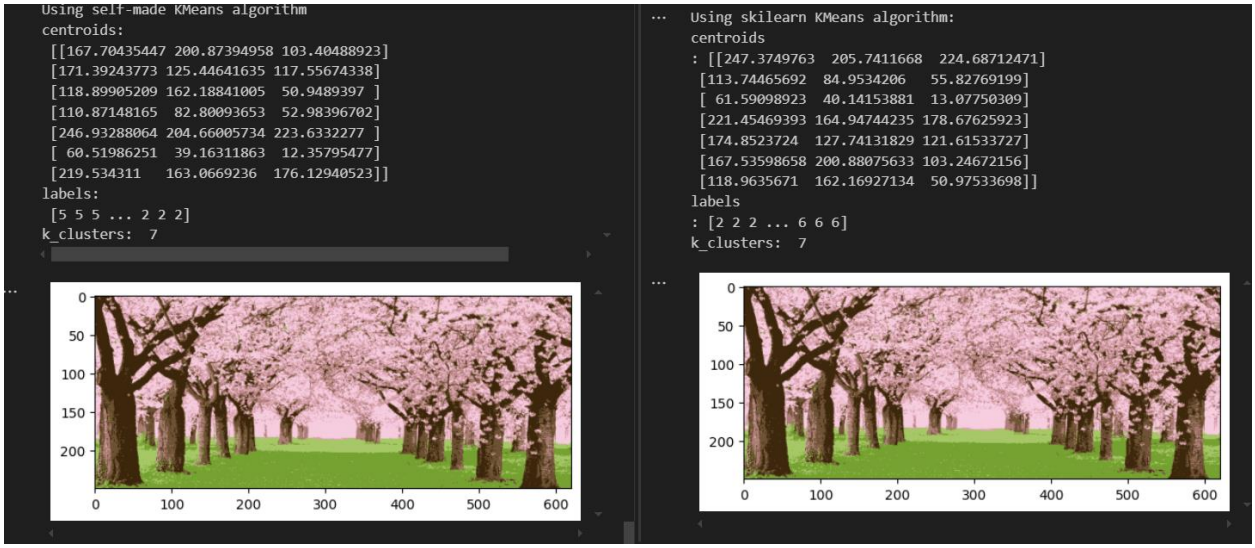
```
Using self-made KMeans algorithm
centroids:
 [[ 68.08661083  46.04339646  17.79303229]
 [243.82943136 198.4454393  217.12554041]
 [124.57706403 167.7296273   56.72526251]
 [203.185854   154.70435397 154.93124374]
 [136.33195272 101.33088235  79.01582278]]
labels:
 [0 0 0 ... 2 2 2]
k_clusters:  5
```




```
Using skilearn KMeans algorithm:
centroids
: [[123.07191855 144.42194469  57.66824572]
 [219.67872368 166.32076732 176.14983776]
 [ 71.66302328  48.5692568   20.81701667]
 [247.16973806 205.0857319  224.15491525]
 [173.16609043 136.99725177 118.60527482]]
labels
: [2 2 2 ... 0 0 0]
k_clusters:  5
```



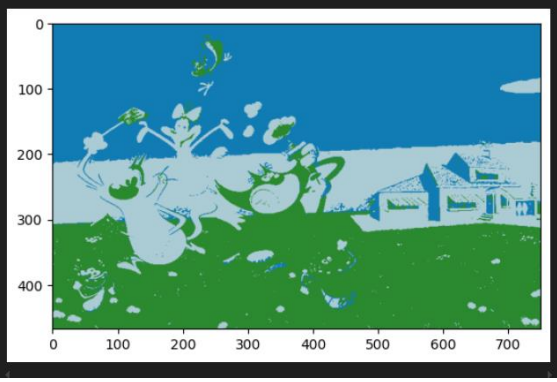
- **K_clusters = 7**



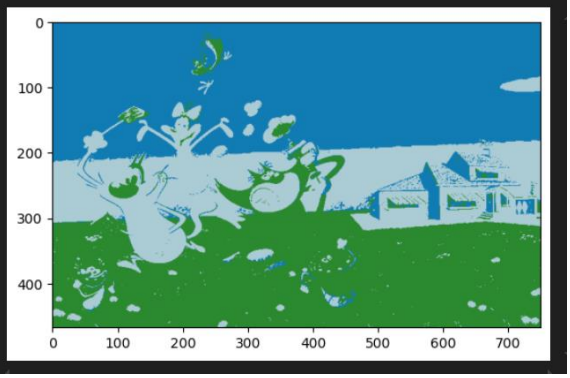
c. Testcase 3 (Using in_pixels)

- **K_clusters = 3**

```
Using self-made KMeans algorithm
centroids:
 [[171.39078327 203.06265415 213.22796389]
 [ 43.85273709 137.85573098  48.09625929]
 [ 17.68920957 124.37178726 180.94516426]]
labels:
 [2 2 2 ... 1 1 1]
k_clusters:  3
```

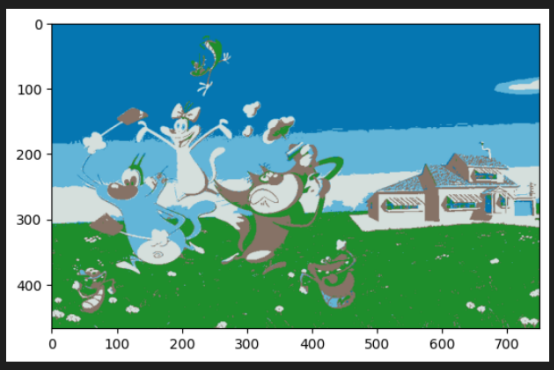



```
Using skilearn KMeans algorithm:
centroids
: [[ 43.85349191 137.85925662  48.09247527]
 [ 17.79342441 124.41036421 180.95762325]
 [171.54893985 203.16510074 213.27520789]]
labels
: [1 1 1 ... 0 0 0]
k_clusters:  3
```

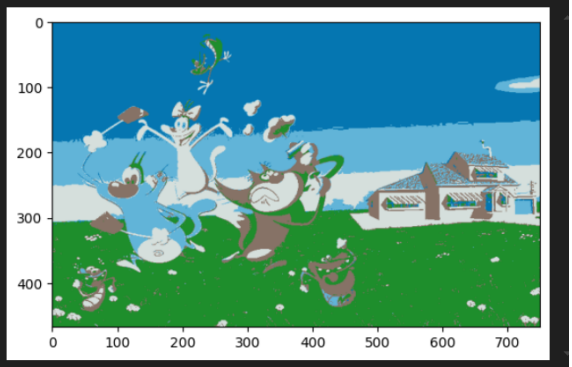


- **K_clusters = 5**

```
Using self-made KMeans algorithm
centroids:
 [[  5.46845084 118.32467163 177.66068039]
 [134.39574133 114.82400466 102.88870631]
 [213.80377102 223.6031746  220.07727529]
 [ 30.0420831  142.20871007  44.82954296]
 [ 98.31289466 181.23585614 216.19948751]]
labels:
 [0 0 0 ... 3 3 3]
k_clusters:  5
```

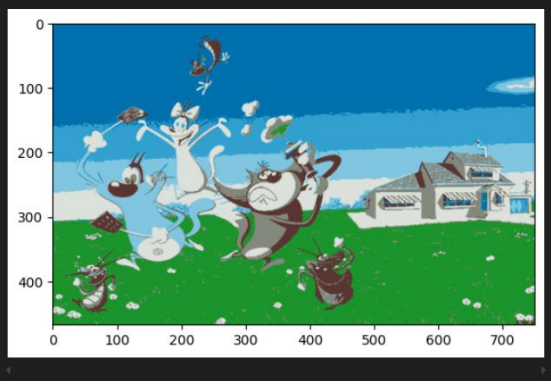



```
Using skilearn KMeans algorithm:
centroids
: [[213.47090417 223.50966748 220.13013731]
 [ 30.05050486 142.19734235  44.83577824]
 [  5.33458643 118.16185985 177.54342919]
 [ 97.60732441 180.89682062 216.02844129]
 [134.5435827  114.88867536 102.82961565]]
labels
: [2 2 2 ... 1 1 1]
k_clusters:  5
```



- **K_clusters = 7**

```
Using self-made KMeans algorithm
centroids:
 [[133.7393036  137.69010772 125.25436201]
 [221.81362736 225.89356027 219.44495161]
 [ 28.71187837 148.97958469  44.7643896 ]
 [128.41496539 198.56277132 225.81388596]
 [ 90.11053055  54.47179796  51.25301447]
 [ 50.89180843 161.10039134 207.4387296 ]
 [  1.86287894 113.90365293 174.57249931]]
labels:
 [6 6 6 ... 4 4 4]
k_clusters:  7
```

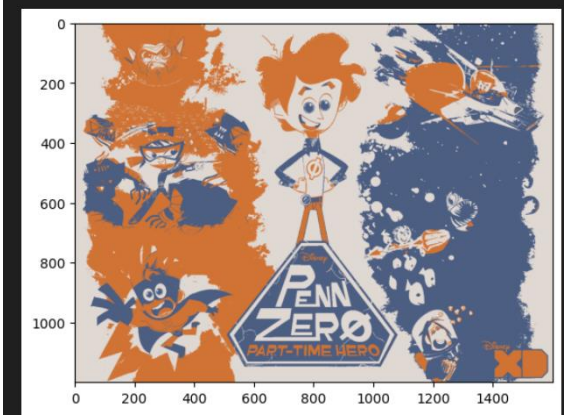



```
Using skilearn KMeans algorithm:
centroids
: [[  2.0814122  114.47841429 174.97692756]
 [130.21082772 198.83572441 225.55655922]
 [ 28.76794976 149.16383959  44.83828458]
 [ 53.92678788 162.17486869 207.84142761]
 [ 77.92427902  52.76452621  53.87768149]
 [222.25879083 226.03507992 219.40366921]
 [138.34964223 134.66185891 119.59850897]]
labels
: [0 0 0 ... 4 4 4]
k_clusters:  7
```

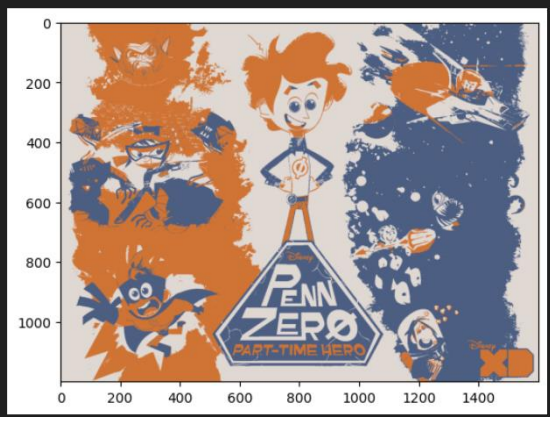
d. Testcase 4 (Using in_pixels)

- **k_clusters = 3**

```
Using self-made KMeans algorithm
centroids:
 [[208.41089076 115.78309435  53.31415628]
 [224.68042411 216.88827187 211.19680585]
 [ 76.92667065  94.54831699 130.14914024]]
labels:
 [1 1 1 ... 1 1 1]
k_clusters:  3
```

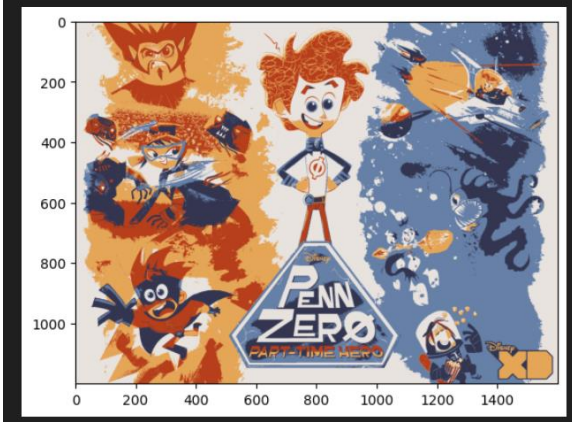



```
Using skilearn KMeans algorithm:
centroids
: [[224.66661443 216.95930117 211.3653752 ]
 [ 76.9773977   94.46880617 130.00922068]
 [208.6339752  116.09511866  53.55076705]]
labels
: [0 0 0 ... 0 0 0]
k_clusters:  3
```



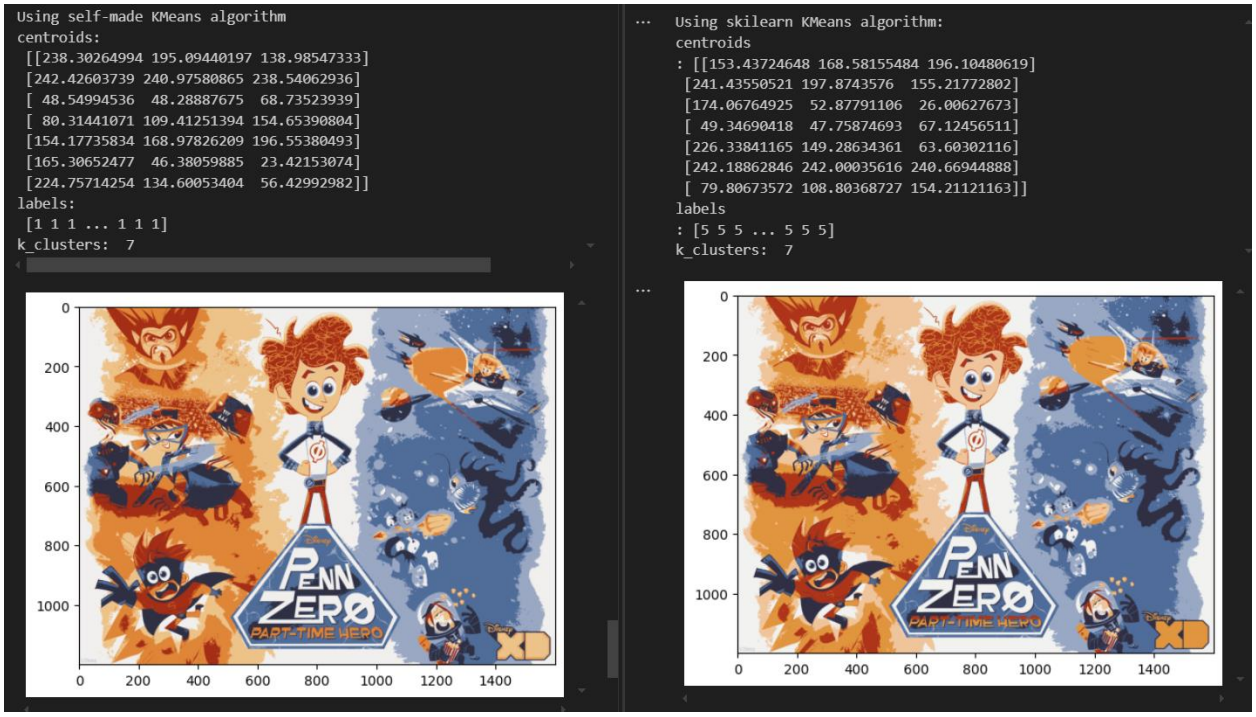
- **k_clusters = 5**

```
Using self-made KMeans algorithm
centroids:
 [[102.49524052 128.74539274 168.96634636]
 [230.24782404 166.25635047  88.11038743]
 [233.3489816  227.76099737 223.22301858]
 [183.51031882  63.86913986  28.86279077]
 [ 52.40324094  54.75140919  81.24393356]]
labels:
 [2 2 2 ... 2 2 2]
k_clusters:  5
```




```
Using skilearn KMeans algorithm:
centroids
: [[ 52.52840704  54.17220001  80.00989359]
 [230.37171919 167.24978431  89.31866713]
 [101.98796894 128.26927017 168.67550751]
 [233.23169298 227.82475954 223.49430722]
 [184.79156776  65.22149622  29.26557032]]
labels
: [3 3 3 ... 3 3 3]
k_clusters:  5
```

- **K_clusters = 7**

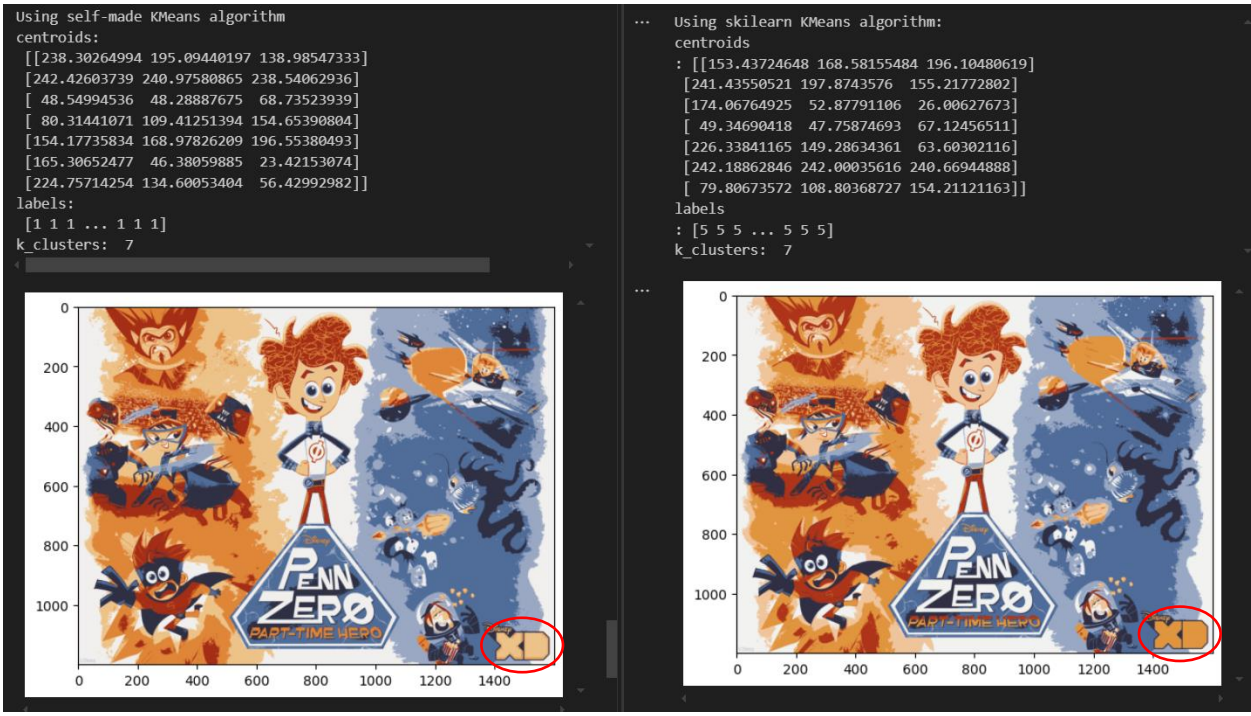


4. Nhận xét về kết quả của test case

- Gần như so với hàm sẵn của thư viện scikit-learn của test case trên thì output cho ra là gần như như nhau, tuy nhiên thời gian chạy của hàm tự làm thì lại có thời gian chạy gần như gấp 2 hay 3 lần hoặc lớn hơn rất nhiều đối với k_cluster nhiều cũng như hình có nhiều kênh màu so với hàm sẵn của scikit-learn

	Thời gian chạy trung bình Của hàm tự làm (1000 iterations)	Thời gian chạy trung bình Của hàm scikit-learn (1000 iterations)
Testcase 1 (Random)	19.4 / 3 = 6.47s	5.4 / 3 = 1.8s
Testcase 2 (Random)	15.7 / 3 = 5.23s	2.5 / 3 = 0.833s
Testcase 3 (In_pixels)	9.1 / 3 = 3.0333s	2 / 3 = 0.6667s
Testcase 4 (In_pixels)	85.2 / 3 = 28.4s	9.8 / 3 = 3.27s

- Tuy nhiên chúng ta vẫn thấy có sự khác nhau về điểm màu ở test case sau



- Đồng thời theo số lượng cụm k_clusters tăng, trọng tâm trở nên cụ thể và chi tiết hơn, thu được nhiều chi tiết hơn về cấu trúc dữ liệu.
- **Ưu và nhược điểm của k-means:**

Ưu điểm	Nhược điểm
Phương pháp dễ thực hiện, tổng quát và linh động	Có thể không chính xác và chi phí thời gian chạy rất tốn kém
Thuật toán tối ưu nhất trong việc phân tách ảnh và màu	Vì k-means là thuật toán dựa trên khoảng cách nên không phù hợp với việc phân cụm các cụm không lồi

- **Sự khác nhau giữa hai cách random và in_pixels:**

	Random	In_pixels
Ưu điểm	Dễ thực hiện Tạo các centroid nhanh	Có thể hiệu quả hơn do chọn random từ các cụm điểm từ ảnh trước Hiệu suất cải thiện do khả năng hội tụ được nhanh hơn nếu centroids ban đầu được chọn tốt
Nhược điểm	Kết quả có thể khác nhau ở mỗi lần chạy do random từ đó dẫn đến khả năng hội tụ khác nhau	Có thể phức tạp hơn so với random Cần có ma trận 1D ban đầu để lấy random ra

	Hội tụ kém do có thể centroids ban đầu không được chọn tốt	
Kết luận: + Random: do centroids được khởi tạo đơn giản nên khả năng hội tụ dưới mức tối ưu + In_pixels: do centroids được khởi tạo bằng cách chọn random từ ma trận của hình nên khả năng hội tụ được tối ưu ⇨ **In_pixels sẽ cho ra chất lượng hình ảnh cao hơn random và thời gian có thể chạy nhanh hơn**		

5. Nguồn tham khảo

- https://machinelearningcoban.com/2017/01/01/kmeans/
- https://www.geeksforgeeks.org/image-compression-using-k-means-clustering/
- https://www.youtube.com/watch?v=5w5iUbTlpMQ&t=1681s
- https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/?utm_source=blog&utm_medium=comprehensive-guide-k-means-clustering