

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY  
UNIVERSITY OF SCIENCE  
FACULTY OF INFORMATION TECHNOLOGY



**SUBJECT:** Applied Mathematics and Statistics

CLASS 20CLC02

Phan Minh Phúc

20127063

## Project 01: Color Compression

Lecturers: Vũ Quốc Hoàng, Nguyễn Văn Quang Huy, Lê Thanh Tùng, Phan Thị Phương Uyên

Ho Chi Minh City – 2022

## CONTENTS

<b>CONTENTS</b> .....	2
<b>I. General information</b> .....	3
<b>II. Requirements</b> .....	3
1. Project's requirement: .....	3
2. Report's requirement:.....	3
<b>III. Conclusion</b> .....	3
1. Present idea and describe functions .....	3
2. Testing result .....	8
3. Comments.....	20
4. Compare with KMeans algorithm in scikit-learn library.....	21
<b>IV. References</b> .....	31

## I. General information

- Project 01: Color Compression
- Author: Phan Minh Phúc – Student ID: 20127063
- Environment: jupyter notebook
- Programing language: python

## II. Requirements

### 1. Project's requirement:

- Install a program that reduces the number of colors in an image using the K-Means algorithm.
- Provides a main () function that allows users to enter their image file's name and to select the saving format for their image's output including “png” and “pdf”.

### 2. Report's requirement:

No.	Task	Status
1	Present your idea generally, describe all your defined functions	Done
2	For each value of k (k = 3, 5, 7), return image after being compressed corresponding to k. You can try more value of k to gain insight into this algorithm	Done
3	Make your own comments on those results	Done
4	Your report must be paginated and included the reference section	Done

## III. Conclusion

### 1. Present idea and describe functions

#### - General idea:

- Color images commonly used are RGB images, where each pixel stores 3 color channel information (1 byte for each color channel): R (red), G (green), B (blue). Each component can take a value between 0 and 255, where the tuple - pixel (0, 0, 0) represents black and (255, 255, 255) represents white.
- K-means is a clustering algorithm that is used to group data points into clusters (centroids) such that data points lying in the same group are very similar to each other in characteristics. K-means algorithm can be used to find subgroups in the image and assign the image pixel to that subgroup which results in image segmentation. We are

proactive in choosing  $k$  - the number of colors the image must represent.

- So, we're going to use K-means to present the initial image in the same size but only with  $K$  colors.
- Describe functions:
  - Steps of K-means:
    - Step 1: Generate  $K$  centroids.
    - Step 2: Assign the image pixel to these centroids.
    - Step 3: Generate  $K$  new centroids which values equal the mean of that cluster  $i$ .
    - Step 4: Repeat until the results are convergent.
  - Declare libraries:

```
# Ho Chi Minh city University of Science (HCMUS)
# Author: Phan Minh Phuc
# Student ID: 20127063
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image as img
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin
from sklearn.utils import shuffle
from time import time
```

- generate\_centroid(img\_1d, k\_clusters, init\_centroids):

```
def generate_centroid(img_1d, k_clusters, init_centroids):
    if init_centroids == 'in_pixels':
        return A[np.random.randint(A.shape[0], size = k_clusters), :]
    if init_centroids == 'random':
        return np.random.randint(256, size = (k_clusters, 3))
```

- Generate  $K$  centroids base on the value of **init\_centroids**:
  - “random” → centroid has 3 channels, each channel is initial random in  $[0, 255]$ .
  - “in\_pixels” → centroid is a random pixels of original image.
- classify\_pixel(img\_1d, centroid):

```
def classify_pixel(img_1d, centroid):
    # distance between each pixel and the 1st centroid
    distance = np.linalg.norm(img_1d - centroid[0], axis = 1)
    # convert the array to shape (img_1d.shape[0], 1)
    distance = distance.reshape((img_1d.shape[0], 1))

    # distance between each pixel and the rest centroids
    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))
        # adding these distances to the initial distance array
        distance = np.concatenate((distance,temp),axis=1)

    # return index of centroid nearest to each pixel
    # return distance
    return np.argmin(distance,axis = 1)
```

- Calculate the distance between each pixel and the 1st centroid, save to **distance**.
  - Then, calculate the distance between each pixel and the rest centroids similarly.
  - Once again, save these results to distance by using `numpy.concatenate()` – this is the reason why you have to calculate the distance between each pixel and the 1st centroid before doing the same thing with the rest centroids.
  - Return a list contains the index of centroid nearest to each pixel aka a list contains the index of centroid that each pixel belongs to.
- `optimize_centroid(img_1d, k_cluster, centroid):`

```
def optimize_centroid(img_1d, k_cluster, centroid):
    label = classify_pixel(img_1d, centroid)
    is_changed = False
    for i in range (k_cluster):
        # choose all pixels belong to centroid[k]
        cluster = img_1d[label == i, :]
        # if no pixel belong to this centroid
        # => go on to the next centroid
        if len(cluster) == 0:
            continue
        # generate a new centroid from mean value of last chosen pixels
        new_centroid = np.mean(cluster, axis = 0)
        # if new centroid and centroid[k] is not convergent
        # => set centroid[k] with the new centroid's value
        if (not np.allclose(centroid[i], new_centroid, atol = 1, equal_nan = True)):
            centroid[i] = new_centroid
            is_changed = True

    return is_changed, centroid, label
```

- Generate **label** to save the index of centroid that each pixel belongs to.
  - Generate **is\_changed** to check whether the centroids are changed and assign the value **False** to it.
  - For each centroid, generate **cluster** contains all pixels belong to it, then
  - If there isn't any pixel belong to this centroid, the algorithm will go on to the next centroid.
  - Else, generate **new\_centroid** which value equal the mean of this **cluster**.
  - If the difference in value of this centroid and the **new\_centroid** is larger than the absolute tolerance parameter ( $atol = 1$ ), the function will update this centroid's value to new centroid's value and assign the value **True** to **is\_changed**.
  - Return whether the initial centroids are changed (**is\_changed**), the current centroid values (**centroid**), and the current label (**label**).
- `compress_pixel(img_1d, k_clusters, label, centroids):`

```
def compress_pixel(img_1d, k_clusters, label, centroids):
    for i in range(k_clusters):
        img_1d[label == i, :] = centroids[i]
    return img_1d
```

- For each centroid in K centroids, change the value of all pixels belong to this centroid to its value.
  - Return a matrix contain value of all pixels in the original image (**img\_1d**) after being compressed.
- `kmeans(img_1d, k_clusters, max_iter, init_centroids = 'random'):`

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids = 'random'):
    centroid = generate_centroid(img_1d, k_clusters, init_centroids)
    label = []
    while max_iter > 0:
        stop_condition, centroid, label = optimize_centroid(img_1d, k_clusters, centroid)
        if stop_condition == False:
            break;
        max_iter -= 1;
    # print(label.shape)

    return centroid, label
```

- Generate **centroid** contains K centroid by using `generate_centroid(img_1d, k_clusters, init_centroids)`.
- Generate **label** – an empty list contains the index of centroid that each pixel belongs to.

- Start converging centroid by using `optimize_centroid(img_1d, k_cluster, centroid)` until the `stop_condition` is satisfied or `max_iter` is  $\leq 0$ .
- Return the current `centroid` and `label`.

➤ `main()`:

```
def main():
    filename = input("Enter image's name: ")
    print("Filename: " + filename)
    output_format = input("Enter image's output format ('png' or 'pdf'): ")
    print("Image's output format: " + output_format)
    while output_format != "png" and output_format != "pdf":
        output_format = input("Enter image's output format ('png' or 'pdf'): ")
        print("Image's output format: " + output_format)

    img_1d = img.open(filename)
    # Convert the input to a matrix.
    img_1d = np.asarray(img_1d)

    # img_1d.shape[0]: number of rows
    # img_1d.shape[1]: number of columns
    # img_1d.shape[2]: number of channels
    row = img_1d.shape[0]
    column = img_1d.shape[1]
    # Reshape initial matrix to 1d array
    # img_1d.reshape(-1, 3) == img_1d.reshape(img_1d.shape[0] * img_1d.shape[1], img_1d.shape[2])
    img_1d = img_1d.reshape(-1, 3)

    k_clusters = 5
    max_iter = 3000
    centroid, label = kmeans(img_1d, k_clusters, max_iter, init_centroids = 'random')

    compressed_img = compress_pixel(img_1d, k_clusters, label, centroid)
    compressed_img = compressed_img.reshape(row, column, 3)
    print(f"Number of clusters: {k_clusters}")
    print(f"Centroids: \n {centroid}")
    plt.clf()
    print(f"Compressed image ({k_clusters} colors, Project 1: Color Compression)")
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(compressed_img)
    plt.savefig(f"compressed_img{k_clusters}." + output_format, dpi = 300)
```

- Generate `filename`, `output_format` and let users enter their image file's name and select the saving format for their image's output including "png" and "pdf".
- Assign `img_1d` the result of opening image `filename` by using `PIL.Image.open()`.
- Convert the `img_1d` into a matrix shape  $\text{row} \times \text{column} \times \text{channel}$ .
- Assign `row` the value of `img_1d`'s row.
- Assign `column` the value of `img_1d`'s column.
- Reshape initial matrix to 1d array by using `numpy.reshape()`.
- Assign `k_clusters` respectively the value 3, 5, 7, 13, 27 and 63.
- Assign `max_iter` (max iterator) the value 3000.



- Assign **centroid**, **label** the result after executing `kmeans(img_1d, k_clusters, max_iter, init_centroids = 'random')`.
- Assign **compressed\_img** the result after executing `compress_pixel(img_1d, k_clusters, label, centroids)`.
- Reshape **compressed\_img** to the initial matrix's shape by using `numpy.reshape()`.
- Use `matplotlib.pyplot.figure()` to create a new figure.
- Use `matplotlib.pyplot.figure.set_size_inches()` to set the figure size in inches. Row and Column are divide by 96 to convert from pixels to inches.
- Use `matplotlib.pyplot.axis("off")` to remove the axes and the plot borders completely.
- Show **compressed\_img** by using `matplotlib.pyplot.imshow()` and save it as name "`compressed_img{k_clusters}`" (`k_cluster`: number of centroids) with output format **output\_format** by using `matplotlib.pyplot.savefig()`.

## 2. Testing result

- The original iamge:



- $K = 3$ :



In [59]: `main()`

```
Enter image's name: 1.jpg
Filename: 1.jpg
Enter image's output format ('png' or 'pdf'): png
Image's output format: png
Number of clusters: 3
Centroids:
[[112  91 136]
 [199 156 167]
 [ 29  23  62]]
Compressed image (3 colors, Project 1: Color Compression)

<Figure size 432x288 with 0 Axes>
```





-  $K = 5$ :

In [57]: `main()`

```
Enter image's name: 1.jpg
Filename: 1.jpg
Enter image's output format ('png' or 'pdf'): png
Image's output format: png
Number of clusters: 5
Centroids:
[[102  92 141]
 [ 28  22  61]
 [187 105 119]
 [ 66 231 249]
 [201 174 185]]
Compressed image (5 colors, Project 1: Color Compression)

<Figure size 432x288 with 0 Axes>
```





-  $K = 7$ :

```
In [61]: main()
```

```
Enter image's name: 1.jpg  
Filename: 1.jpg  
Enter image's output format ('png' or 'pdf'): png  
Image's output format: png  
Number of clusters: 7  
Centroids:  
[[ 16  11  44]  
 [ 42 217 107]  
 [195  97 105]  
 [229 192 184]  
 [155 137 174]  
 [101  89 138]  
 [ 51  43  92]]  
Compressed image (7 colors, Project 1: Color Compression)
```

```
<Figure size 432x288 with 0 Axes>
```





-  $K = 13$ :

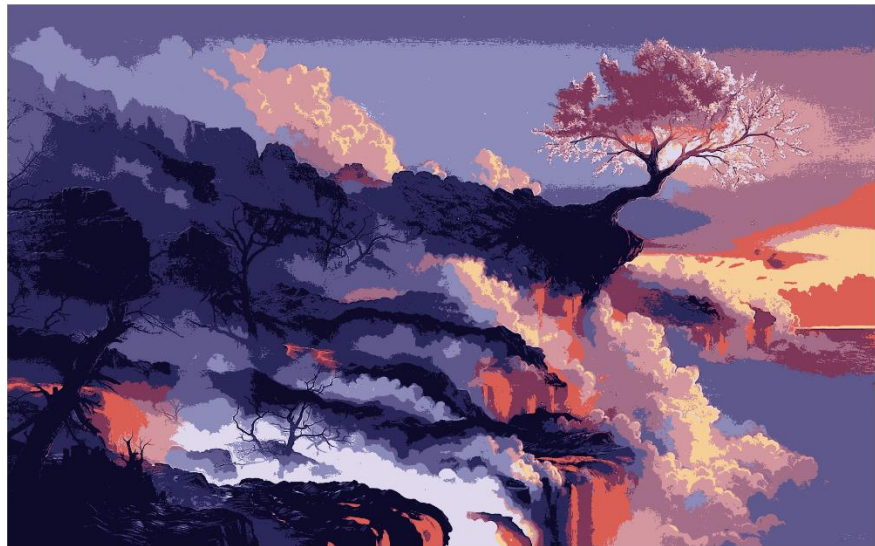
In [63]: `main()`

```
Enter image's name: 1.jpg
Filename: 1.jpg
Enter image's output format ('png' or 'pdf'): png
Image's output format: png
Number of clusters: 13
Centroids:
[[ 42  40  90]
 [ 14  10  41]
 [246 206 152]
 [140 139 185]
 [224 217 237]
 [165 110 136]
 [  6  96 217]
 [ 22   6 235]
 [ 83 231  79]
 [212 151 160]
 [219  94  85]
 [ 94  88 140]
 [126  55  88]]
Compressed image (13 colors, Project 1: Color Compression)

<Figure size 432x288 with 0 Axes>
```







- K = 27:

In [65]: `main()`

```
Enter image's name: 1.jpg
Filename: 1.jpg
Enter image's output format ('png' or 'pdf'): png
Image's output format: png
Number of clusters: 27
Centroids:
[[ 25 215 18]
 [217 42 26]
 [178 99 117]
 [135 71 107]
 [ 8 5 30]
 [ 98 60 103]
 [ 91 89 143]
 [ 63 28 71]
 [167 68 250]
 [225 129 127]
 [151 51 70]
 [ 71 7 21]
 [ 68 194 83]
 [133 100 138]
 [ 58 191 227]
 [174 255 107]
 [154 159 202]
 [ 23 25 72]
 [224 91 73]
 [ 50 57 110]
 [126 125 174]
 [180 132 155]
 [251 224 142]
 [225 169 169]
 [ 51 167 255]
 [229 223 241]
 [ 73 196 75]]
Compressed image (27 colors, Project 1: Color Compression)
```



-  $K = 63$ :

```
In [67]: main()
```

```
Enter image's name: 1.jpg  
Filename: 1.jpg  
Enter image's output format ('png' or 'pdf'): png  
Image's output format: png  
Number of clusters: 63  
Centroids:  
[[108  67 254]  
 [ 30 249   4]  
 [186 149 169]  
 [ 42   6  21]  
 [230  80  94]  
 [179  97 116]  
 [216 121 103]  
 [140  75 108]  
 [ 65 173 191]  
 [ 39  43  96]  
 [219 158 162]  
 [ 36 220 191]  
 [ 18  98 179]]
```

In [67]: main()

```
[ 50 220 191]
[ 18 98 179]
[100 79 125]
[235 181 170]
[154 10 170]
[217 38 31]
[ 64 71 123]
[ 79 162 171]
[146 154 201]
[151 53 217]
[246 206 199]
[168 127 156]
[ 41 189 214]
[ 68 228 54]
[ 68 116 254]
[161 60 78]
[232 119 136]
[138 211 27]
[ 19 92 236]
[252 238 138]
```

In [67]: main()

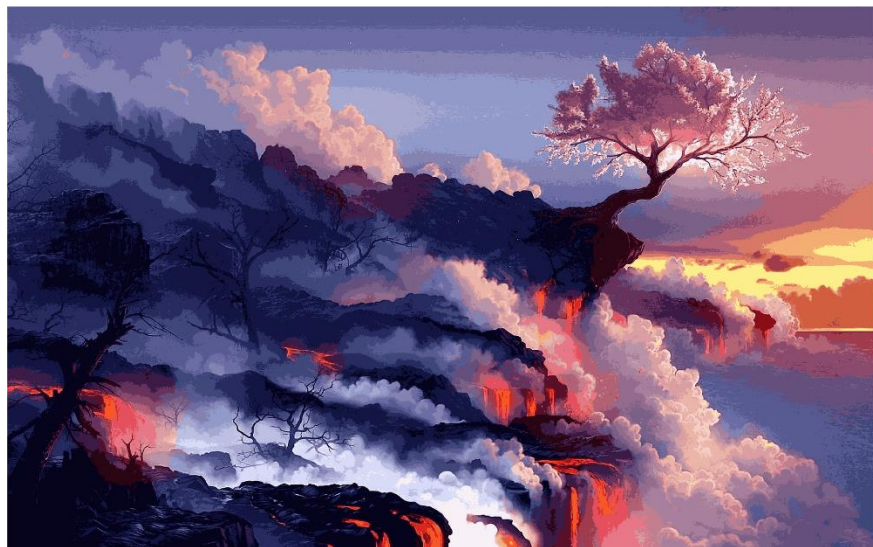
```
[ 19 92 236]
[252 238 138]
[ 88 90 146]
[ 2 234 142]
[168 169 208]
[248 240 247]
[ 51 178 0]
[ 3 96 183]
[216 219 248]
[123 96 136]
[ 44 141 9]
[139 13 21]
[ 99 194 167]
[ 90 16 40]
[108 109 164]
[196 127 143]
[ 17 195 61]
[190 192 227]
[ 55 160 179]
[ 26 190 160]
```

In [67]: main()

```
[ 26 190 160]
[  5   4  28]
[ 49 143 250]
[ 67  49 187]
[ 19  21  67]
[133 131 178]
[247  88  41]
[103  51  92]
[131 250  74]
[148 108 141]
[ 49  65 212]
[210  94  64]
[ 60  27  71]
[ 25 119 234]
[244 174 116]]
```

Compressed image (63 colors, Project 1: Color Compression)

<Figure size 432x288 with 0 Axes>



### 3. Comments

- When we increase the value of K, there's a noticeable improvement in showing the image.
- The larger the value of K is, the clearer and sharper the image is shown and the longer it takes to finish the program.

#### 4. Compare with KMeans algorithm in scikit-learn library

- Executing KMeans algorithm in scikit-learn library:

```
# compare results with KMeans in scikit-learn library
n_colors = 3

# Load the Summer Palace photo
pic_name = input("Enter image's name: ")
print("Filename: " + pic_name)

image = img.open(pic_name)
# Convert to floats instead of the default 8 bits integer coding. Dividing by
# 255 is important so that plt.imshow behaves works well on float data (need to
# be in the range [0-1])
image = np.array(image, dtype=np.float64) / 255

# Load Image and transform to a 2D numpy array.
w, h, d = original_shape = tuple(image.shape)
assert d == 3
image_array = np.reshape(image, (w * h, d))

print("Fitting model on a small sub-sample of the data")
t0 = time()
image_array_sample = shuffle(image_array, random_state=0, n_samples=1_000)
km = KMeans(n_clusters=n_colors, random_state=0).fit(image_array_sample)
print(f"done in {time() - t0:0.3f}s.")

# Get labels for all points
print("Predicting color indices on the full image (k-means)")
t0 = time()
labels = km.predict(image_array)
print(f"done in {time() - t0:0.3f}s.")

def recreate_image(codebook, labels, w, h):
    """Recreate the (compressed) image from the code book & labels"""
    return codebook[labels].reshape(w, h, -1)

print(f'Centroids: \n {km.cluster_centers_ * 255}')
plt.figure(2)
plt.clf()
plt.axis("off")
plt.title(f"Compressed image ({n_colors} colors, K-Means)")
plt.imshow(recreate_image(km.cluster_centers_, labels, w, h))
```

- K = 3:

```
Enter image's name: 1.jpg
Filename: 1.jpg
Fitting model on a small sub-sample of the data
done in 0.143s.
Predicting color indices on the full image (k-means)
done in 0.207s.
Centroids:
[[114.87359551  90.10674157 133.38483146]
 [ 29.00877193  24.1754386   64.66666667]
 [196.92384106 154.59933775 168.01655629]]
```

```
Out[41]: <matplotlib.image.AxesImage at 0x2540d47df0>
```

Compressed image (3 colors, K-Means)



- K = 5:



```
Enter image's name: 1.jpg
Filename: 1.jpg
Fitting model on a small sub-sample of the data
done in 0.243s.
Predicting color indices on the full image (k-means)
done in 0.082s.
Centroids:
[[ 91.80811808  79.29151292 128.6199262 ]
 [ 25.7980456   20.80130293  60.09120521]
 [154.68269231 134.79326923 172.19711538]
 [227.33018868 192.20754717 190.0754717 ]
 [196.44444444  99.53703704 102.52777778]]
```

```
Out[44]: <matplotlib.image.AxesImage at 0x25405be2d40>
```

Compressed image (5 colors, K-Means)



-  $K = 7$ :

```
Enter image's name: 1.jpg
Filename: 1.jpg
Fitting model on a small sub-sample of the data
done in 0.163s.
Predicting color indices on the full image (k-means)
done in 0.191s.
Centroids:
[[ 13.81987578   9.80745342  41.30434783]
 [150.40935673 137.00584795 176.64912281]
 [135.42372881  56.52542373  84.61016949]
 [203.54621849 114.67226891 119.04201681]
 [227.22       194.82       192.61       ]
 [ 96.64903846  88.54807692 138.80288462]
 [ 40.13736264  39.26923077  88.98351648]]
```

```
Out[34]: <matplotlib.image.AxesImage at 0x1198b5dbb80>
```

Compressed image (7 colors, K-Means)



- K = 13:

```
Enter image's name: 1.jpg
Filename: 1.jpg
Fitting model on a small sub-sample of the data
done in 0.184s.
Predicting color indices on the full image (k-means)
done in 0.085s.
Centroids:
[[191.08641975 134.82716049 155.43209877]
 [ 87.07142857 26.96428571 61.35714286]
 [ 90.64473684 87.07894737 139.34868421]
 [ 22.46527778 23.73611111 70.90972222]
 [157.25862069 159.86206897 201.93103448]
 [131.91954023 126.01149425 172.01149425]
 [139.22916667 62.60416667 91.33333333]
 [ 11.70408163  4.95918367 28.51020408]
 [237.46031746 186.3015873 166.77777778]
 [ 48.12765957 53.85106383 106.03191489]
 [218.51923077 105.59615385 93.42307692]
 [224.07407407 219.62962963 240.40740741]
 [147.54411765 99.39705882 131.82352941]]
```

Out[36]: <matplotlib.image.AxesImage at 0x1198c220b20>

Compressed image (13 colors, K-Means)



- K = 27:

```

Enter image's name: 1.jpg
Filename: 1.jpg
Fitting model on a small sub-sample of the data
done in 0.218s.
Predicting color indices on the full image (k-means)
done in 0.100s.
Centroids:
[[ 67.58064516  36.5483871  79.80645161]
 [155.93023256 131.65116279 164.76744186]
 [192.38888889 195.72222222 231.88888889]
 [ 71.24615385  77.18461538 130.         ]
 [ 10.93939394  12.39393939  51.31818182]
 [179.4375      104.3125      120.09375   ]
 [229.09090909 166.         161.15151515]
 [ 98.0483871   99.19354839 155.80645161]
 [150.3902439   156.80487805 203.97560976]
 [124.48484848  61.78787879  99.54545455]
 [226.8125      91.9375      58.875      ]
 [193.63333333 153.56666667 173.16666667]
 [ 41.25        50.57352941 103.94117647]
 [109.72727273  18.90909091  43.27272727]
 [ 24.03846154  26.20192308  74.59615385]
 [ 43.17647059   5.41176471  18.76470588]
 [253.         232.33333333 135.55555556]
 [231.6         139.9         101.4         ]
 [137.10869565  97.89130435 135.34782609]
 [126.78846154 126.98076923 176.51923077]
 [240.30769231 234.53846154 247.84615385]
 [191.23809524 127.78571429 148.47619048]
 [218.72222222 101.27777778 112.22222222]
 [166.6875      63.75        75.         ]
 [  3.12244898   1.7755102   21.36734694]
 [ 99.62745098  81.88235294 129.21568627]
 [243.54166667 199.54166667 191.29166667]]

```

```
Out[39]: <matplotlib.image.AxesImage at 0x1198e74dba0>
```

Compressed image (27 colors, K-Means)



- $K = 63$ :



```

Enter image's name: 1.jpg
Filename: 1.jpg
Fitting model on a small sub-sample of the data
done in 0.375s.
Predicting color indices on the full image (k-means)
done in 0.149s.
Centroids:
[[141.6      146.9      193.35      ]
 [ 60.11764706 29.47058824 73.58823529]
 [205.46666667 139.26666667 155.4      ]
 [ 96.         93.875     153.29166667]
 [  3.46153846  1.26923077 12.38461538]
 [ 26.73333333 28.46666667 76.93333333]
 [167.2        84.         81.2        ]
 [249.44444444 240.         246.77777778]
 [ 49.         59.43478261 114.47826087]
 [241.41176471 198.88235294 198.88235294]
 [195.64705882 109.70588235 117.94117647]
 [138.66666667 99.55555556 136.33333333]
 [117.05555556 62.94444444 103.22222222]
 [229.5         64.25       71.         ]
 [ 15.81818182 13.13636364 44.72727273]
 [253.25        243.25       128.5        ]
 [247.28571429 193.         171.57142857]
 [ 73.75         79.25       133.20833333]
 [129.           129.48       181.68        ]
 [212.63636364 212.36363636 244.45454545]
 [166.52         141.64       170.36        ]
 [ 45.           3.27272727  7.54545455]
 [ 93.375        75.04166667 123.29166667]
 [172.7          173.5        199.1         ]
 [231.8          116.4        133.         ]
 [188.63636364 130.68181818 150.72727273]
 [ 94.11111111 22.44444444 52.77777778]
 [  2.93939394  3.42424242 35.         ]
 [126.52631579 89.36842105 128.42105263]
 [ 76.38461538 47.76923077 90.23076923]

```

```
[ 94.11111111 22.44444444 52.77777778 ]
[ 2.93939394 3.42424242 35. ]
[126.52631579 89.36842105 128.42105263]
[ 76.38461538 47.76923077 90.23076923]
[244. 159.33333333 107.66666667]
[143.63157895 130.68421053 168.21052632]
[248. 132. 9. ]
[221.63636364 124. 92.81818182]
[157.5 42.5 51.5 ]
[219.375 89.5 109.625 ]
[ 41.75 49.53571429 102.53571429]
[131.25 12.5 32.5 ]
[150.68421053 159.10526316 210.57894737]
[198.57142857 155.92857143 179.92857143]
[181. 189.9 228. ]
[ 9.75757576 13.81818182 57.84848485]
[226.39130435 169.82608696 166.69565217]
[212.71428571 100.57142857 63.42857143]
[102.89285714 105.21428571 158.92857143]
[ 41. 10.125 43.375 ]
[234.44444444 154.55555556 146. ]
[153.72222222 114.33333333 148.88888889]
[169.78571429 94. 119.92857143]
[ 17.82222222 21.77777778 70.02222222]
[252. 205.33333333 126. ]
[151.09090909 69.18181818 101.54545455]
[247. 86.66666667 44. ]
[106.31578947 87.47368421 131.94736842]
[ 32.09677419 39.4516129 90.41935484]
[253.66666667 246.66666667 169. ]
[ 83.33333333 88.13333333 142.2 ]
[184. 58.66666667 65. ]
[116.85 117.55 167.7 ]
[ 63.04347826 70.52173913 122.82608696]
[125.625 52.375 89.5 ]
[169.4 114.8 125. ]
[177.30769231 115.53846154 144.53846154]]
```



```
Out[40]: <matplotlib.image.AxesImage at 0x1198c132fe0>
```

Compressed image (63 colors, K-Means)



#### - Comment:

- Like our `kmeans` function, `KMeans` algorithm in `scikit-learn` library improve the quality of the image when we keep increasing the value of `K` aka the number of clusters.
- Using only `numpy` library take us more time to handle the large matrices instead of using the others libraries (such as `Scipy`, `OpenCV`...).
- The result (the value of centroids) of our `kmeans` is quite similar to `scikit-learn` one.
- The larger the value of `K` is, the more difference in the two results of those algorithms you get.
- As you can see increasing the value of `K` causes the value differential between the centroids return by our `kmeans` function and `scikit-learn` library's `KMeans`.
- This happens because of rounding the result returned by `numpy.mean()` in `optimize_centroid` function and the value of `max_iter` in `kmeans` function - it limits the number of times that `optimize_centroid` is executed which means when `K` is larger, executing `optimize_centroid` `max_iter` times is not enough to converge or to optimize centroids.

## IV. References

<https://numpy.org/doc/stable/index.html>

<https://pillow.readthedocs.io/en/stable/reference/Image.html?highlight=Image.open#PIL.Image.open>

[https://scikit-learn.org/stable/modules/generated/sklearn.cluster.k\\_means.html?highlight=k%20means#sklearn.cluster.k\\_means](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.k_means.html?highlight=k%20means#sklearn.cluster.k_means)

[https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.savefig.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.savefig.html)

[https://www.w3schools.com/python/ref\\_func\\_input.asp](https://www.w3schools.com/python/ref_func_input.asp)

<https://www.geeksforgeeks.org/how-to-create-a-matrix-of-random-integers-in-python/>

<https://stackoverflow.com/questions/14262654/numpy-get-random-set-of-rows-from-2d-array>

<https://stackoverflow.com/questions/1401712/how-can-the-euclidean-distance-be-calculated-with-numpy>

<https://stackoverflow.com/questions/28952946/how-to-replace-one-column-by-a-value-in-a-numpy-array>

<https://www.codingem.com/numpy-how-to-create-an-empty-array/>

[https://www.w3schools.com/python/python\\_while\\_loops.asp](https://www.w3schools.com/python/python_while_loops.asp)

[https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_color\\_quantization.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html)

[https://www.youtube.com/watch?v=00V\\_3Ow29BE](https://www.youtube.com/watch?v=00V_3Ow29BE)

<https://stackoverflow.com/questions/66121087/why-plt-savefig-reduced-the-original-size-of-an-image>

<https://www.adamsmith.haus/python/answers/how-to-save-a-matplotlib-figure-as-a-certain-size-in-python>

<https://www.statology.org/matplotlib-hide-axis/>

<https://blog.finxter.com/how-to-change-the-size-of-figures-drawn-with-matplotlib/>