VNU HCMC-UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

PROJECT REPORT

APPLIED MATHEMATICS AND STATISTICS

COLOR COMPRESSION

Lecturers: Vu Quoc Hoang

Practical Instructor: Phan Thi Phuong Uyen

Nguyen Van Quang Huy

Le Thanh Tung

Student: Bui Quang Thanh - 20127329

Contents

ABSTRACT

The K-Means clustering algorithm is proposed by Mac Queen in 1967 which is a partition-based cluster analysis method. It is used widely in cluster analysis for that the K-means algorithm has higher efficiency and scalability and converges fast when dealing with large data sets. However, it also has many deficiencies: the number of clusters K needs to be initialized, the initial cluster centers are arbitrarily selected, and the algorithm is influenced by the noise points. In view of the shortcomings of the traditional K-Means clustering algorithm, this paper presents an improved K-means algorithm using noise data filter. The algorithm developed density-based detection methods based on characteristics of noise data where the discovery and processing steps of the noise data are added to the original algorithm. By preprocessing the data to exclude these noise data before clustering data set the cluster cohesion of the clustering results is improved significantly and the impact of noise data on K-means algorithm is decreased effectively and the clustering results are more accurate.

This report discusses a traditional clustering method called KMeans algorithm from mathematical perspective. Additionally, an experiment is provided to examine the algorithm in two-dimensional space then an application in image compressing.

INTRODUCTION

Clustering is an important means of data mining and of algorithms that separate data of similar nature. Unlike the classification algorithm, clustering belongs to the unsupervised type of algorithms. Color compression in an image is a process of separating colors in the image. K-means algorithms identify *k* number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The *'means'* in the K-means refers to averaging of the data; that is, finding the centroid. We will be clustering the pixel intensities of an RGB image. Given an *MXN* size image, we thus have *MxN* pixels, each consisting of three components: Red, Green, and Blue respectively.

In this lab, we will simultaneously use list and NumPy to perform math operations on matrices.

In it, use list to perform math operations by manual code; use NumPy to call functions available in the library.

THE IDEA OF IMPLEMENTING THE PROJECT

Build algorithms based on input data

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids='random'):
    '''
    K-Means algorithm

    Inputs:
        img_1d : np.ndarray with shape=(height * width, num_channels)
            Original image in 1d array

        k_clusters : int
            Number of clusters

        max_iter : int
            Max iterator

        init_cluster : str
            The way which use to init centroids
            'random' --> centroid has `c` channels, with `c` is initial random in [0,255]
            'in_pixels' --> centroid is a random pixels of original image

    Outputs:
        centroids : np.ndarray with shape=(k_clusters, num_channels)
            Store color centroids

        labels : np.ndarray with shape=(height * width, )
            Store label for pixels (cluster's index on which the pixel belongs)

    '''
```

The idea is to use **plt** (import matplotlib. pyplot as plt) to read the image, then reshape the image from a three-dimensional scalar into a 2-array dimension of the form [**height*width, n_channel**], here **n_channel = 3**.
Corresponding to each type of init centroids is random in the range [0.255] or random pick from the color of the image, we do an init for centroids of **n_clusters** color points where **n_clusters** is the number of clusters (**3,5 or 7).** Then we run exactly **max_iter** times, each time, for a pixel taken out, we recalculate the distance (L2 distance) and get the mean of the extracted image clusters, then reassign it to the new label and update it again centroids.
In the current installation, all the functions of the NumPy library are used and are limited. Use loops if not needed. From there, it is possible to improve the speed of the algorithm.

DESCRIPTION OF FUNCTIONS

1. Essential functions

```
  #random initialization centroid -->
def random_centroids(img,k_clusters,input):
    try:
        if input =="random":
            centroids = np.random.randint(low=0,high=255,size=(k_clusters,img.shape[1]))
        elif input =="in_pixels":
            centroids=img[np.random.choice(img.shape[0] , size = k_clusters , replace = False)]
    except:
        print("Please check again input !")
    return centroids
```

This function to generate random centroids that can be seen at random from any other color or can randomly choose the colors in the image that the user passes in. If the user enters the wrong input, the message will be returned.

```
def calc_inner_product(v, w): # calculate inner product
        return sum(vi*wi for vi, wi in zip(v, w))
def norm_square(v):
    return calc_inner_product(v, v)
def sub_vector(v, w):   #minus 2 vector
    return [vi - wi for vi, wi in zip(v, w)]
def euclidian_distance(vector_i, vector_j):# calculate euclidean distance
        return np.sqrt(norm_square(sub_vector(vector_i,vector_j)))
```

The three functions above are used to calculate the norm of a vector.

```
 # Specify the index of the labels
def indexMinDistance(distance,input):
    for i in range(len(distance)):
        if input==distance[i]:
            return i

# Calculate the point closest to the centroids and generate a labels
def takeClosestCentroids(img_1d, centroids):
    labels = np.zeros(len(img_1d)) # index value correspond cluster which each pixel belongs to
    distance=[]
    for i in range(len(img_1d)):
        distance_temp=10000.0;
        distance=[]
        for j in range(len(centroids)):
            result=euclidian_distance(img_1d[i],centroids[j])
            distance.append(result)
            distance_temp = min(result,distance_temp)
        labels[i]=indexMinDistance(distance,distance_temp)
    return labels # return index of closest centroids for each sample
```

The function takeClosestCentroids() takes 2 parameters:

1. One processed photo that can be reshape to the same size
2. Random random centroids values.

   This function is used to calculate the distance and consider the closest vectors to generate labels. The indexMinDistance() function is used to return the index of the vector with the closest distance to the centroids.

```
def updateCentroids(img, labels, old_centroids_shape):
    centroids = np.zeros(old_centroids_shape)
    for i in range(old_centroids_shape[0]):
        # Calculate new centroid using mean on rows with each cluster
        if  len(img[(labels == i)])!=0: # Prevent nan values
            centroids[i] = img[(labels == i)].mean(axis=0)
    return centroids
```

This function is used to update the centroids by averaging the centroids. The output is a new centroid is mean of centroids.

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids='random'):
    centroids=random_centroids(img_1d,k_clusters,init_centroids)
    labels  = np.full(img_1d.shape[0], -1)

    for i in range(max_iter):
        # labels  = takeClosestCentroids_1(img_1d, centroids) Method 1
        labels =takeClosestCentroids(img_1d, centroids) # Method 2
        old_centroids = centroids
        # print("Truoc ki update: ",old_centroids)
        centroids = updateCentroids(img_1d, labels, centroids.shape)
        if np.allclose(old_centroids, centroids, rtol=10e-3, equal_nan=False):
            break
    return centroids, labels
```

The function takes 4 parameters

- img_1d: (np.ndarray) with shape=(height * width, num_channels). Original image in 1d array
- k_clusters : (int) Number of clusters
- max_iter : (int) . Max iterator
- init_cluster : (str) The way which use to init centroids
  - 'random' --> centroid has `c` channels, with `c` is initial random in [0,255]
  - 'in_pixels' --> centroid is a random pixels of original imagE.

Outputs:

- centroids : np.ndarray with shape=(k_clusters, num_channels).Store color centroids
- labels : np.ndarray with shape=(height * width,).Store label for pixels (cluster's index on which the pixel belongs)

The above function returns centroids and labels. Then iterates through each pixel in the image for each pixel with its own labels.

```
     # Returns the final image after reshape
def convertOriginalSize(labels,centroids,flat_image,image):
    try:
        labels = labels.astype(int) # change int labels
        img_result=np.zeros_like(flat_image) # create new iamge have same size with flat_image
        for i in range(len(img_result)):
            img_result[i]=centroids[(labels[i])]
        img_result=img_result.reshape(image.shape[0],image.shape[1],image.shape[2]) # reshape new
image to original size
    except:
        print("Error")
    return img_result
```

`convertOriginalSize()`: This function takes the labels of all pixels in the image. Iterate through each pixel with each label assigned the corresponding centroids.

```
 def openAndReadImage(nameImg): #Operation to open 1 photo
      image = plt.imread(nameImg)
      image = np.array(image,dtype='int64')
      return image

def iamgePreProcess(image): #pre-processing of the photo
      width=image.shape[0]
      height=image.shape[1]
      channels=image.shape[2]
      flat_image = image.reshape(width* height,channels)
      return flat_image,width, height, channels
```

The above two functions are used for image processing : read image , reshape image ,.

2.Support functions

```
def display(Image1,n_clusters01,Image2,n_clusters02,Image3,n_clusters03,Image4,n_clusters04):
    # create figure
    fig = plt.figure(figsize=(10, 7))

    # setting values to rows and column variables
    rows = 2
    columns = 2

    # reading imag

    # Adds a subplot at the 1st position
    fig.add_subplot(rows, columns, 1)

    # showing image
    plt.imshow(Image1)
    plt.axis('off')
    plt.title(n_clusters01,color='white')

    # Adds a subplot at the 2nd position
    fig.add_subplot(rows, columns, 2)

    # showing image
    plt.imshow(Image2)
    plt.axis('off')
```

```
    plt.title(n_clusters02,color='white')

    # Adds a subplot at the 3rd position
    fig.add_subplot(rows, columns, 3)

    # showing image
    plt.imshow(Image3)
    plt.axis('off')
    plt.title(n_clusters03,color='white')

    # Adds a subplot at the 4th position
    fig.add_subplot(rows, columns, 4)

    # showing image
    plt.imshow(Image4)
    plt.axis('off')
    plt.title(n_clusters04,color='white')
```

This function uses the matplotlib library to output multiple images at once

3.Using module

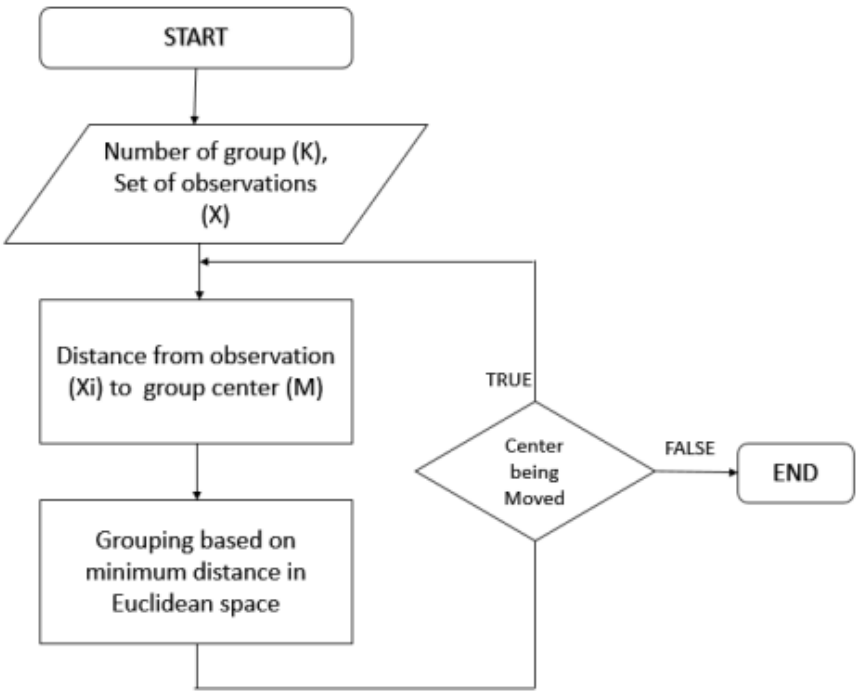
```
 from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

img_test=plt.imread('tree.jpg')
width_test=img_test.shape[0]
height_test=img_test.shape[1]
n_channel=img_test.shape[2]
img_test=img_test.reshape(width_test*height_test,n_channel)


kmeans=KMeans(n_clusters=4).fit(img_test)
labels=kmeans.predict(img_test)
clusters=kmeans.cluster_centers_

#method 1
img_test_2=np.zeros_like(img_test)
for i in range(len(img_test)):
    img_test_2[i]=clusters[labels[i]]

#method 2

img_test_3=np.zeros((width_test,height_test,n_channel),dtype=np.uint8)
index=0

for i in range(width_test):
    for j in range(height_test):
        labels_of_pixel=labels[index]
        img_test_3[i][j]=clusters[labels_of_pixel]
        index=index+1


img_test_2=img_test_2.reshape(width_test,height_test,n_channel)
plt.imshow(img_test_3)
plt.show()
```

Here is the code that uses the library to test with the same image.

ALGORITHM SUMMARY AND FLOWCHART

Step 1: Clusters the data into k groups where k is predefined.
Step 2: Select k points at random as cluster centers.
Step 3: Assign objects to their closest cluster center according to the Euclidean distance function.
Step 4: Calculate the centroid or mean of all objects in each cluster.
Step 5: Repeat steps 2

2) Flowchart: The following chart describe K-Means Algorithm



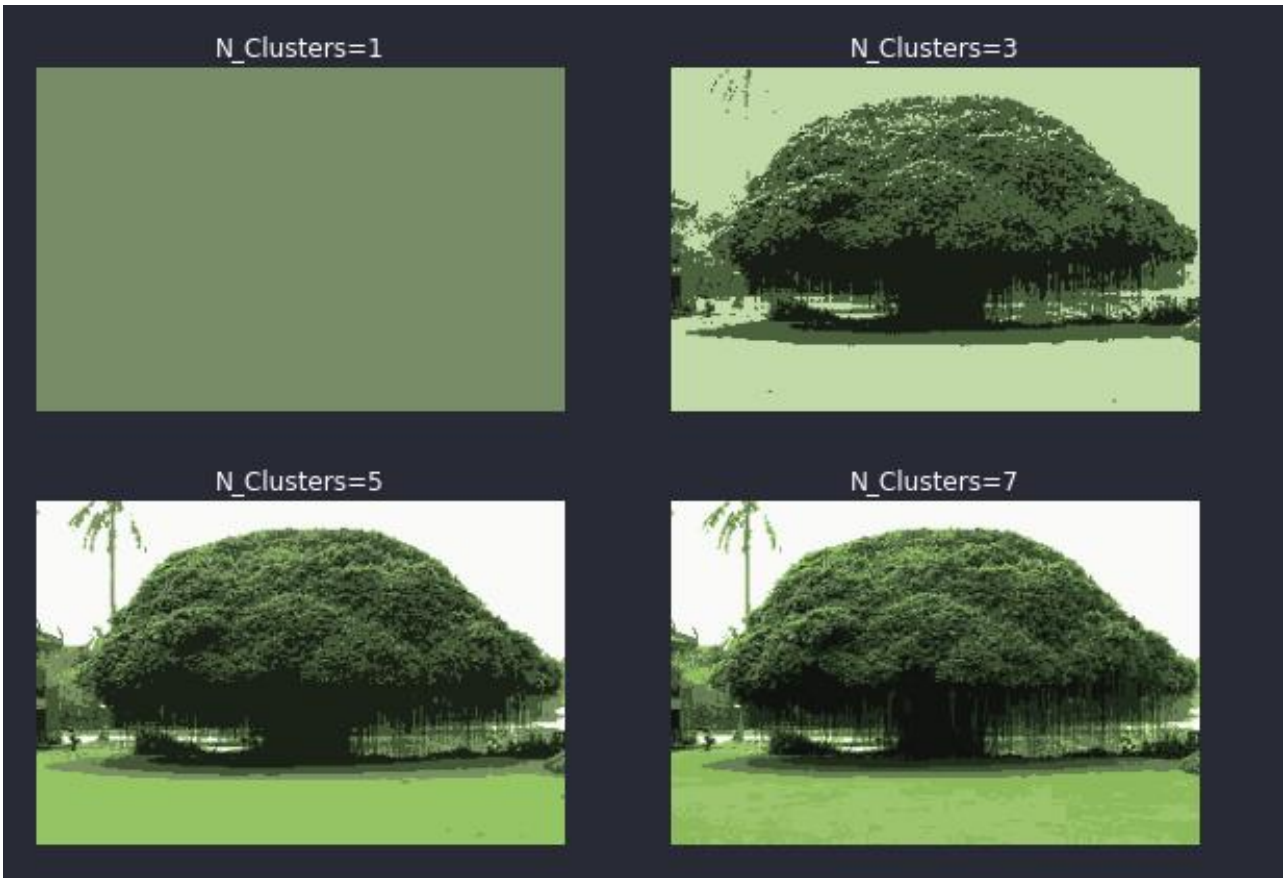
K-Means Algorithm Flowchart

RESULT IMAGE

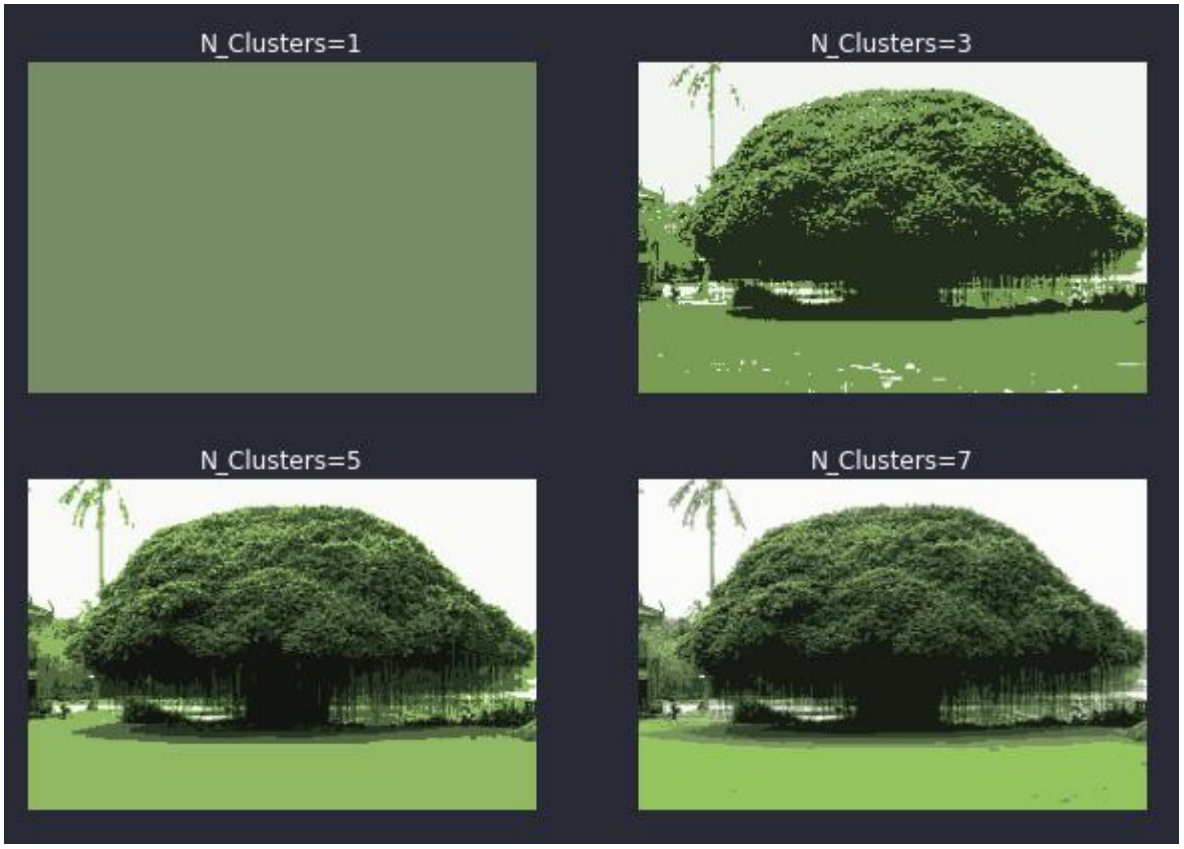
1.Test case 01

The picture below is the original image with 278*181

The image below is the image after running the program with max_iterator=3 (time: 49.6s)
49.6s: is the running time of 4 kmeans algorithms with different n_cluster in the same time.
init_centroids='in_pixels'



The image below is the image after running the program with max_iterator=8 (time: 1m54.9s)
1m54.9s: is the running time of 4 kmeans algorithms with different n_cluster in the same time.
init_centroids='in_pixels'

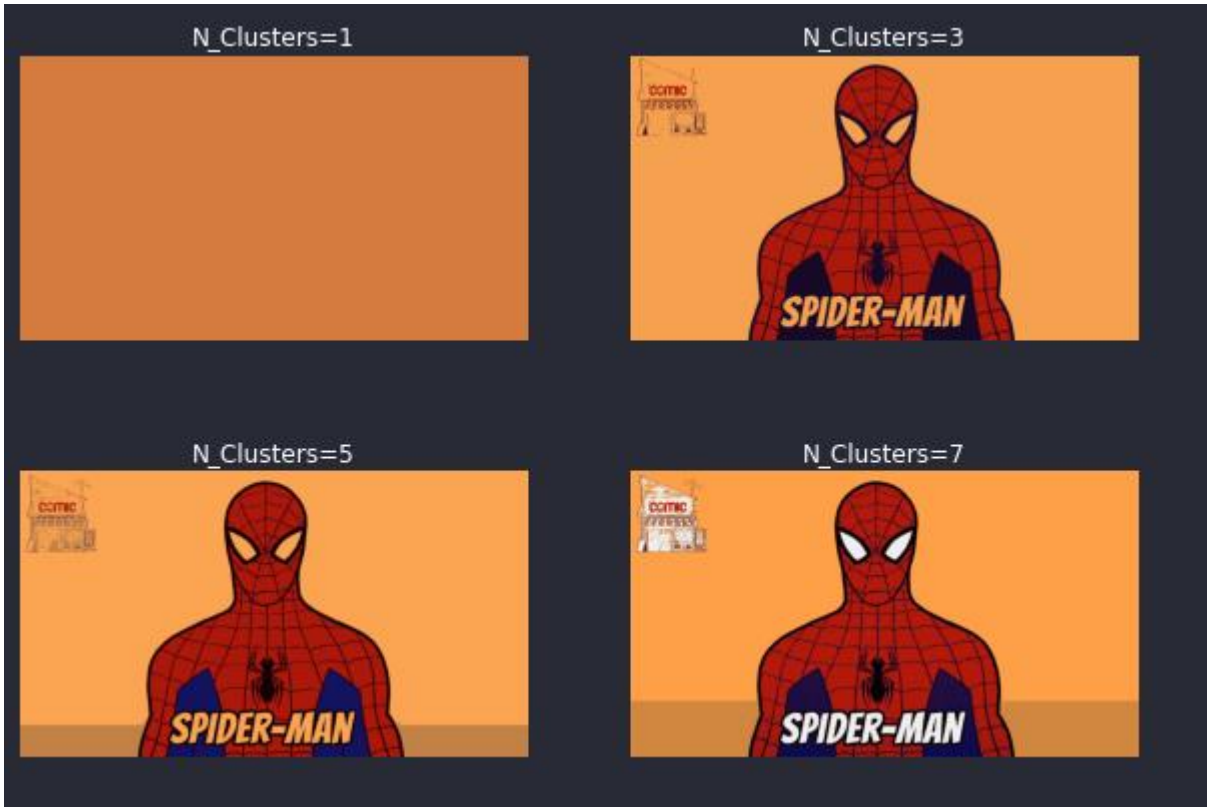


This is the image after using the Kmeans library (n_cluster=4)

2.Test case 02

The picture below is the original image with 1280*720

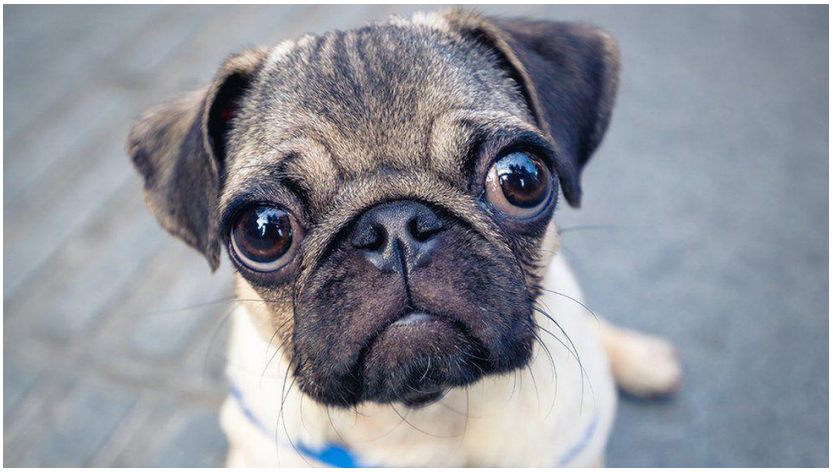
The image below is the image after running the program with max_iterator=3 (time: 9m54s)
9m54s: is the running time of 4 kmeans algorithms with different n_cluster in the same time.
init_centroids='in_pixels'



This is the image after using the Kmeans library (n_cluster=4)

3.Test case 03

The picture below is the original image with 976*549



The image below is the image after running the program with max_iterator=10 and n_cluster=5 (time: 8m01.1s), init_centroids='in_pixels'

The image below is the image after running the program with max_iterator=10 and n_cluster=5
(time: 6m58.1s) init_centroids='random'

COMMENT

1. Algorithm

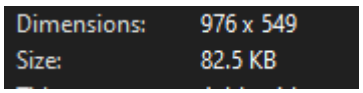
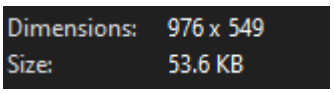
The algorithm is simple to implement, however, the sensitivity to initial centroids, and strict structure of dataset, etc are those drawbacks of the algorithm which are undeniable. Further research could be made to improve the value of initial centroids. The traditional algorithm is depended on randomness, research could also be made to discover a way to make a fixed initial centroid. Furthermore, on a large dataset, the algorithm could be very slow to converge. Research could be spent to make the iteration stops earlier.

2.Iamge quality

The above result is acceptable, if compared with scikit-learn's Kmean, it is effective are also approximately equivalent. However, the image compression performance for small image cases (images already compressed - color reduction using other software) or monochrome images often produce poor results with low max_iter).

3.Size of image

In addition, when color compression also shows us the size of the image is significantly reduced.

4.Time

If the image is large or we set the max_iterator too high, the possibility of a long wait is possible.

In addition, the speed when running the kmean algorithm depends on whether your computer is strong, high processing speed or not. In general, this processing speed is acceptable.

REFERENCES

https://machinelearningcoban.com/2017/01/01/kmeans/
Documentation on moodle: Lab01, Lab02
https://github.com/efedoganay/kmeans-image-compression-from-scratch/blob/main/kmeans_image_compression.ipynb
https://nguyenvanhieu.vn/thuat-toan-phan-cum-k-means/
https://stackoverflow.com/questions/46615554/how-to-display-multiple-images-in-one-figure-correctly