HCM UNIVERSITY OF SCIENCE – FALCUTY OF INFORMATION TECHNOLOGY

Applied Mathematics and Statistics

Color Compression

Võ Duy Nhân – 20127056 – 20CLC11

6/20/2022

Teacher: Phan Thị Phương Uyên – Nguyễn Văn Quang Huy

Index

I. Project description

1. Idea

- One image may contain many pixels. And one pixel contains 3 numbers R, G, B, which are in range [0, 255]. These 3 numbers would make up a color. So we have 256^3 colors, which make the file heavy.
- This project is for the purpose of reducing the number of colors on the image down to k_cluster.
- We first pick k pixels in the image as centroids and then separate others to these clusters. And every loop, we update the new centroids and separate again. This process repeats until the centroids do not change or maximum iterators are reached.
- We finally have a new image with less colors.

2. Libraries and modules needed for project

- *Numpy*
- Module *pyplot* from *matplotlib*
- Module *Image* from *PIL*

3. Input image file

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image


# Input filename
print ("Enter the file: ")
fileName = input()

# Open image file
image = Image.open (fileName)

img1d = np.array(image)
plt.imshow(img1d)
(h,w,c) = img1d.shape
img1d = np.reshape (img1d, [h*w,c])
```

- User can enter the image file from built-in function *input()*.
- Module *Image* provide function *open(fileName)* which would load image from files and return a Image object. Here, I named it ***image***.

- I change ***image*** into an array by *np.array* and change into a matrix with shape (h*w, c). In which, h,w is height and width of the picture .correspondingly and c = 3 (RGB number present a pixel) by *numpy.reshape*
- Now we have ***img1d*** is the *numpy.array* with h*w element, each of which is a pixel [R, G, B].

4. Euclidean

```
def euclidean(point, data):
    return [np.linalg.norm(point - pixel) for pixel in data]
```

- This function takes a pixel (*point*) and numpy.array of pixels as centroids (*centroids*) as input and return a list of distance between the point and data.
- *numpy.linalg.norm ()* provide a way to calculate the norm of a vector, and *point – pixel* is also a vector created by numpy. So this actually means calculating the euclidean distance between 2 vectors.

5. kmeans (img1d, k_clusters, max_iter, init_centroids)

```
centroids = []
# Random color pixel in range [0,255]
if init_centroids == 'random':
    centroids = [[np.random.randint(255), np.random.randint(255), np.random.randint(255)] for _ in range (k_clusters)]
# Random pixel
if init_centroids == 'in_pixels':
    centroids = [ img1d[np.random.randint(len(img1d))] for _ in range(k_clusters)]
```

First we initialize our centroids. This depends on init_centroids. In case it is *random*, then we create a list of k vectors, each of which is a [R, G, B] (R, G, B are random integer in range [0,255]).
In case *init_centroids* is in_pixels, we create a list of k random pixels in *img1d*.

```
iteration = 0
prev_centroids = 0
```

We prepare variable *interation* for the loop. Here, I use a while loop because it should stop when centroids stop moving or the maximum iterator is reached. *prev_centroids* are used to store the old centroids every time updating.

```
while np.not_equal (centroids, prev_centroids).any() and iteration < max_iter:
    sorted_points = [[] for _ in range (k_clusters)]
    for pixel in img1d:
        dist = euclidean (pixel, centroids)
        centroid_idx = np.argmin (dist)
        sorted_points[centroid_idx].append(pixel)

    prev_centroids = centroids
    centroids = [np.mean(cluster, axis = 0) for cluster in sorted_points]

    for i, centroid in enumerate (centroids):
        if np.isnan(centroid).any(): # Check if any centroid is not a number (nan) then we assign the previous centroid
            centroids[i] = prev_centroids[i] # to it
    iteration += 1
```

In the while loop, we check the condition if new centroids equal to old centroids by *numpy.not_equal* and if max iterator is reached.

- We prepare a list of k empty lists, which is actually k clusters.
- Then, for each pixel in img1d, we calculate the euclidean distance between it with the centroids, and return the result to dist.
- Then *numpy.argmin* helps to find the index of minimum result in list dist and store it to variable *centroid_idx*.
- Next, we append each pixel to its cluster.

Next step, we updated the new centroids in k cluster by *numpy.mean*. To prevent an element is nan (not a number), we have to check the situation (*numpy.isnan(centroid).any()*) and assign the previous centroid to the new centroid if that happens.

When the loop ends, we now have the centroids.

```
centroids = np.array(centroids).astype(int)
```

Remember the RGB value must be int. We should change all centroid to int value by function *astype(int)*

```
labels = []
for i,pixel in enumerate (img1d):
    dist = euclidean (pixel, centroids)
    centroid_idx = np.argmin (dist)
    labels.append(centroid_idx)
    img1d[i] = centroids[centroid_idx]

return centroids, labels, img1d
```

As we have the final centroids, we should update every pixel to its own cluster color. The kmeans function returns 3 things: centroid, labels, img1d (after updated)

6. Show image after being compressed

```
(centroids, labels, img1d) = kmeans(img1d, 3, 5, 'random')
```

```
image = np.reshape (img1d, [h,w,c])
plt.imshow(image)
format = input ("Please enter format to export:")
if format == "png":
    plt.imsave('result.png',image)
if format == "pdf":
    plt.imsave('result.pdf',image)
if format == "jpg":
    plt.imsave('result.jpg',image)
```

After kmeans returning, we have a new img1d, so we reshape it into its initial shape (h,w,c). Then I show it, as well as save it to different format (pdf, jpg, png)

II. Result image (k = 3, 5, 7)

The origin image is *game.png* like the picture below

Let us define *max_iterato*r = 5. Below is the picture after kmeans algorithm with *k_clusters* = 3

Below is the picture after being color_compressed with *k_clusters* = 5

Below is the picture after being color_compressed with *k_clusters* = 7

III. Evaluation

- The image looks less clearer than origin after being compressed
- The higher k_cluster is the more beautiful the image is
- Though being compressed, we still see the structure of the origin image.

IV. References

https://www.geeksforgeeks.org/python-pil-image-open-method/

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imsave.html

https://ai538393399.wordpress.com/2020/09/29/k-means-clustering-algorithm-without-libraries

https://towardsdatascience.com/create-your-own-k-means-clustering-algorithm-in-python-d7d4c9077670 (if you can not access this link by ctrl + left click, then copy this and paste it on Google search tab, you will find it easily.)