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

June 15th, 2024

Applied Mathematics and Statistics
Report project 1: Color compression

Lecturers and Teaching assistants :

1. Mr.Vũ Quốc Hoàng
2. Mr.Nguyễn Văn Quang Huy
3. Ms.Phan Thị Phương Uyên
4. Mr.Nguyễn Ngọc Toàn

Student:

Name : Nguyễn Hoàng Trung Kiên

ID: 22127478

Class: 22CLC08

Table of content

I. Introduction to the project:

Project 1 - Color compression is the project belongs to Applied Mathematics and Statistics subject. It helps to gain knowledge on matrix calculation and matrix manipulation. The project will illustrate how a 2D image will be compressed using matrix. The image is treated as a matrix, in this project it is a Numpy array. It is converted to Numpy array for easy calculation. The project also requires to perform the K-means algorithm to handle the image.

Coding language used : python 3.10.6
Performing on : Jupyter Notebook
IDE used : Visual Studio Code

II. Project's idea:

The project, which is demonstrated on Jupyter Notebook platform, will take the input of an image. This image may be .jpg, .png… extension. The program handles the task to compress the image by read the input image's directory, then it will convert the image from 2D (height, width, channels) to 1D (height x width, channels). Then the program uses the K-means algorithm to compress the image based on the project's requirements. About K-means:

- K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into a pre-defined number of clusters. The goal is to group similar data points together and discover underlying patterns or structures within the data. [1]
- How it works:
  Initialization: Start by randomly selecting K points from the dataset. These points will act as the initial cluster centroids.
  Assignment: For each data point in the dataset, calculate the distance between that point and each of the K centroids. Assign the data point to the cluster whose centroid is closest to it. This step effectively forms K clusters.
  Update centroids: Once all data points have been assigned to clusters, recalculate the centroids of the clusters by taking the mean of all data points assigned to each cluster.
  Repeat: Repeat steps 2 and 3 until convergence. Convergence occurs when the centroids no longer change significantly or when a specified number of iterations is reached.
  Final Result: Once convergence is achieved, the algorithm outputs the final cluster centroids and the assignment of each data point to a cluster.
  [1]

This project uses the kmean algorithm to gather the similar color points of the image into groups. It is how the image will be compressed in this project. The project takes 4 inputs: input image's directory, output image's directory following its name and extension, centroids initialization type (random or in pixel) and number of clusters. The number of clusters is how many areas the color will be gathered in the program using K-means algorithm, also known as the number of colors in the compress output picture.

III. Library used:

- PIL : to read and write image.
- numpy : to implement calculations by matrix.
- matplotlib: to present the image on output.

IV. Code and functions explanation:

In this section I will explain the code following the order of the executing lines:

1. **Import libraries:** PIL, numpy and matplotlib.

2. **read_img(img_path):**

- **Purpose:** to read image input by the input path.
- **Code explanation:**
  The function takes the file path as an input with the string datatype, then opens the image by Image.open function which is belonged to PIL library. The opened image is then converted into Numpy array for later matrix calculation with the image's data. Finally the function returns that 2D image as numpy array.

3. **show_img(img_2d):**

- **Purpose:** to show the 2D image on screen in its original form.
- **Code explanation**:
  The function takes the 2D numpy array of the original image as an input, then it is displayed by using plt.show function from Matplotlib library to render the image for displaying on screen, since the input takes the numpy array image. Finally the function ends with plt.close function used for closing plot window.

4. **save_img(img_2d, img_path):**

- **Purpose:** to save the 2D image following a directory where you want to save it. It can be saved with many extensions such as .png, .jpg, and .pdf.
- **Code explanation:**
  The function takes the 2D numpy array of the original image and the saving directory as inputs. First the function converts the numpy array back to PIL Image type. Since the input image may have RGBA attribute (Red Green Blue Alpha) so the functions will convert the RGBA image into RGB one. Then the function checks the image's output path by converting all the characters of the output path to lowercase and using string method endswith() to check if the string ends with a specified suffix. After checking,

the function then saves the image with the corresponding extension by using img.save().

5. **convert_img_to_1d(img_2d):**

- **Purpose:** to convert 2D image (height, width, channels) into 1D one (height x width, channels).
- **Code explanation:**
  The function takes 2D numpy array of the original image as input. Then it extracts the image's shape to get its height, width and number of channels. The image is then reshaped into a 1D array where each row represents a pixel (height x width) and each column represents a color channel.

6. **kmeans(img_1d, k_clusters, max_iter, init_centroids='random'):**

- **Purpose:** to compress image by the idea of k-means algorithm which will group similar color points together.
- **Code explanation:**
  1. Initialize centroids:
  - If we want to initialize the centroids 'random', then for each channels the centroids will be initialized with random numbers between 0 and 255 using Numpy function np.random.randint . (k_cluster, img_1d.shape[1]) which is the third parameter passed in the function randint, is also the shape of the array, means the generated array has k_cluster rows and number of columns is also the number of channels in the image. k_cluster is the number of clusters and img_1d.shape[1] is the number of channels in the 1D image.

  - If we want to initialize the centroids 'in_pixel', a random pixel from the image is chosen as a centroid for each cluster using Numpy function np.random.choice, which selects random elements from the given image array. In the function, img_1d.shape[0] is the total number of pixels in the image, k_cluster is the number of cluster and replace=False makes sure that the same pixel is not selected more than once. From this step we have an array of k_clusters unique indices, each representing a randomly chosen pixel from the image, it is stored in the random_idx variable. Then I use img_1d[random_idx] to select these pixels at these indices from img_1d. We will have the result is an array of shape (k_clusters, num_channels), where each row represents the RGB values of a randomly selected pixel from the image. And finally the array is assigned to the centroids variable, which means the initial centroids are the colors of randomly chosen pixels from the image.

2. Update centroids by iteration:

- First I calculate the Euclidean distance between each pixel and each centroid, this will give the distance matrix. I'll explain how this is implemented in math notations.
  Let:
  X = $\{x_1, x_2, \dots, x_n\}$: be the set of pixels, each pixels x_i is a vector in R^3.
  Y = $\{y_1, y_2, \dots, y_m\}$: be the set of centroids, each pixels y_i is a vector in R^3.
  n is the number of pixels, k is the number of centroids.
  $x_i = (x_{i1}, x_{i2}, x_{i3})$: a vector in R^3 of pixel
  $y_j = (y_{j1}, y_{j2}, y_{j3})$: a vector in R^3 of centroid

  We will need to construct the matrix A where each elements A_{ij} is the Euclidean distance between pixel x_i and centroid y_j.
  For calculating distance A_{ij}:

  $$\boldsymbol{A_{ij} = d_{ij}{}^2 = \|x_i - y_j\|^2} \quad [2]$$

  where d_{ij} is the Euclidean distance between pixel x_i and centroid y_j :

  $$\boldsymbol{d_{ij} = \sqrt{(x_{i1} - y_{j1})^2 + (x_{i2} - y_{j2})^2 + (x_{i3} - y_{j3})^2}} \quad [3]$$

  Now back to the code explanation, I implement the calculation of the Euclidean distance of each pixels and each centroids first . It uses img_1d[:, np.newaxis] to add a new axis to img_1d, that will change its shape into (num_pixels, 1, num_channels), makes it simplier to substract the centroids with each pixels. Then we substract img_1d[:, np.newaxis] with centroids and square each elements in the distance array by using **2. The array now has the shape (num_pixels, k_clusters, num_channels). sum(axis=2) will sum the squared distance along the color channels, which collapses the shape to (num_pixels, k_clusters). Each elements in this array is the distance between each pixels and each centroids.

- Then I assign each pixels to the nearest centroid based on the minimum distance using Numpy function np.argmin.

- After that the function recalculates the centroids in each iterations of the K-means algorithm. First it iterates over each cluster j, then selects all the pixels assigned to cluster j and computes the mean of these pixels to get the new centroid. If the pixel has no cluster assigned then it keeps the previous centroid. Finally it combines all new centroids into single Numpy array.

- Then it will check if the array of the previous centroids and new centroids together to see if they are equal to each other by using Numpy function np.array_equal. If the centroids do not change from the previous iteration, the algorithm has converged, and the loop is broken. This is added to optimize the executing speed of the program as it won't iterate when the centroids remain unchanged.

3. Return results: The kmeans function returns the final centroids and the following labels.

7. generate_2d_img(img_2d_shape, centroids, labels):

- **Purpose :** to create 2D image based on the centroids and labels after we implemented K-means algorithm.
- **Code Explanation:**
  The function takes the original 2D image shape, the centroids and labels as input. The labels are reshaped to match the 2D image dimensions. To fit the dimensions of the 2D image, the labels are altered. Every pixel is mapped to the associated centroid color to form the new image. Finally the function returns that new image.

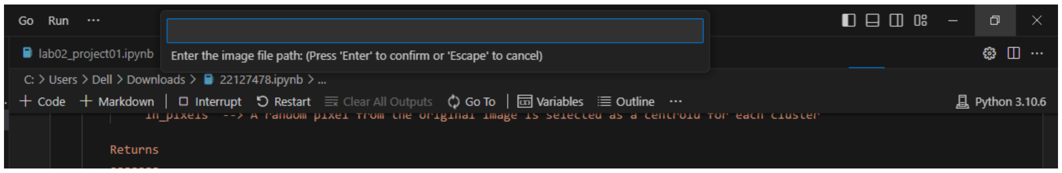
8. main() function:

- **Purpose :** to run the whole project
- **Code explanation:**
  The main function asks user to enter the 2D image file path firstly. Then it reads that 2D image and convert it into 1D using the above functions I explained earlier. Finally it reads the number of k_clusters and centroids_init by user input. Max iteration is set to 100 for K-means algorithm, then we get the centroids and labels after calling kmeans function. After that the new 2D image is generated with the following original 2D image's shape, centroids and labels. Then it outputs the original image and the image after being compressed. Finally it reads the output image's directory through user input corresponding to its extension that the user wants it to have. Finally the result image is saved to the directory and the extensions which has been inputted.

V. How to execute, Outputs and Comments:

1. Executing process:

Step 1: Run the whole jupyter notebook program (Run all/ Run by line). You can see the program asked you to enter the image file path.



Enter the image file path: (Press 'Enter' to confirm or 'Escape' to cancel)

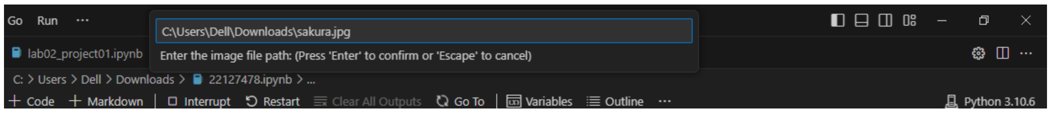
C: > Users > Dell > Downloads > 22127478.ipynb > ...

```
in_pixels --> A random pixel from the original image is selected as a centroid for each cluster

Returns
```

Figure 1 : Step 1

Step 2: Enter the input image file path and press Enter.



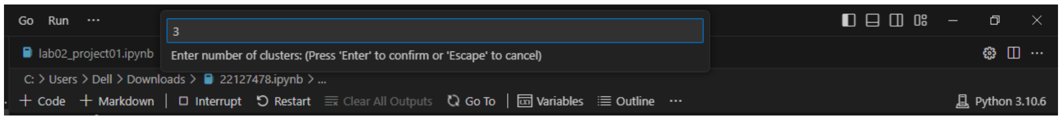
C:\Users\Dell\Downloads\sakura.jpg

Enter the image file path: (Press 'Enter' to confirm or 'Escape' to cancel)

C: > Users > Dell > Downloads > 22127478.ipynb > ...

Figure 2 : Step 2

Step 3: Enter number of clusters and press Enter.



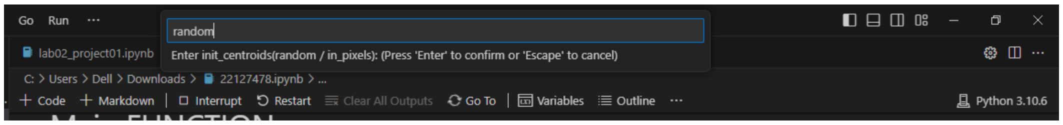
3

Enter number of clusters: (Press 'Enter' to confirm or 'Escape' to cancel)

C: > Users > Dell > Downloads > 22127478.ipynb > ...

Figure 3 : Step 3

Step 4: Enter centroids initialization type(random / in_pixel) and press Enter.



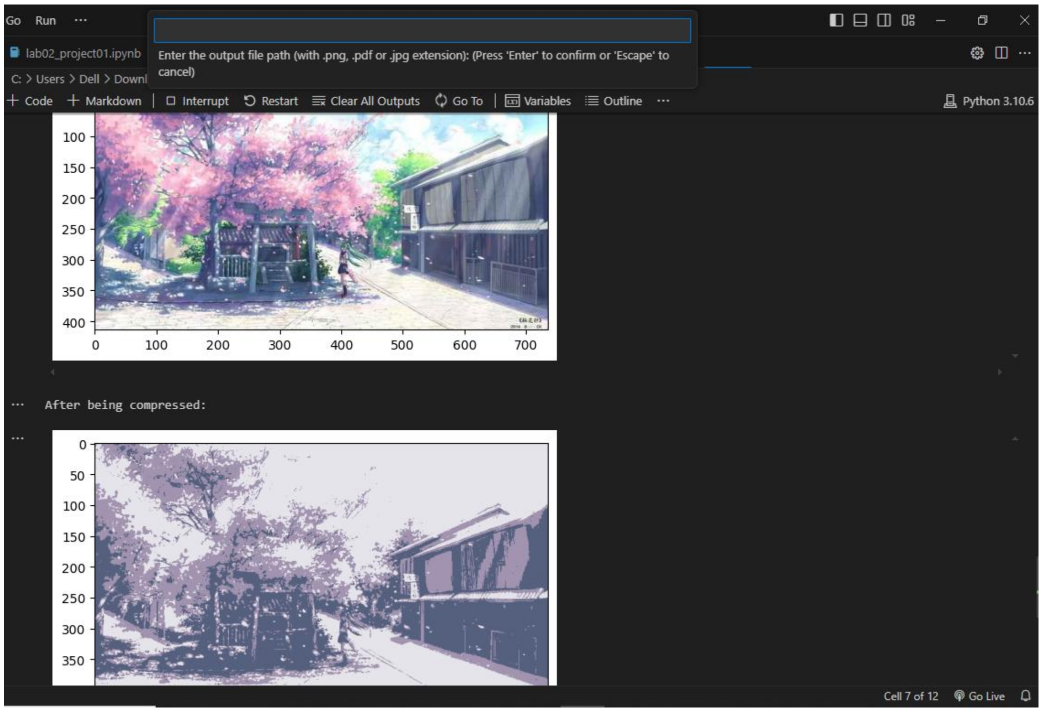
random

Enter init_centroids(random / in_pixels): (Press 'Enter' to confirm or 'Escape' to cancel)

C: > Users > Dell > Downloads > 22127478.ipynb > ...

Figure 4 : Step 4

Step 5: You now can see the compressed image and its original one.

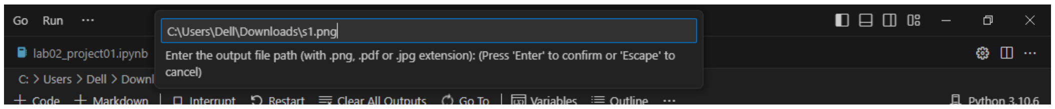


Enter the output file path (with .png, .pdf or .jpg extension): (Press 'Enter' to confirm or 'Escape' to cancel)

```
After being compressed:
```

Figure 5 : Step 5

Step 6 : Enter the output path with the following file name and extension that you want the output image to have. The program will save the image and exit from here.



C:\Users\Dell\Downloads\s1.png

Enter the output file path (with .png, .pdf or .jpg extension): (Press 'Enter' to confirm or 'Escape' to cancel)

Figure 6 : Step 6

2. Outputs:

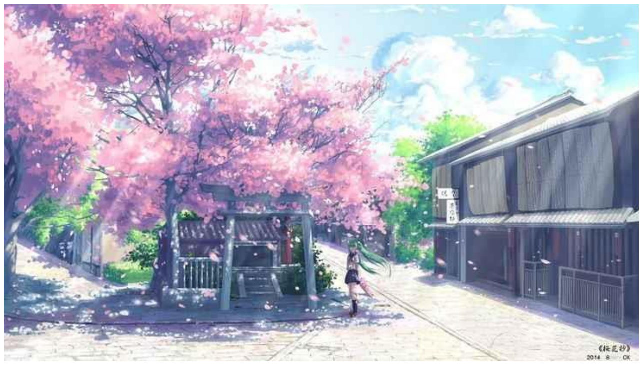
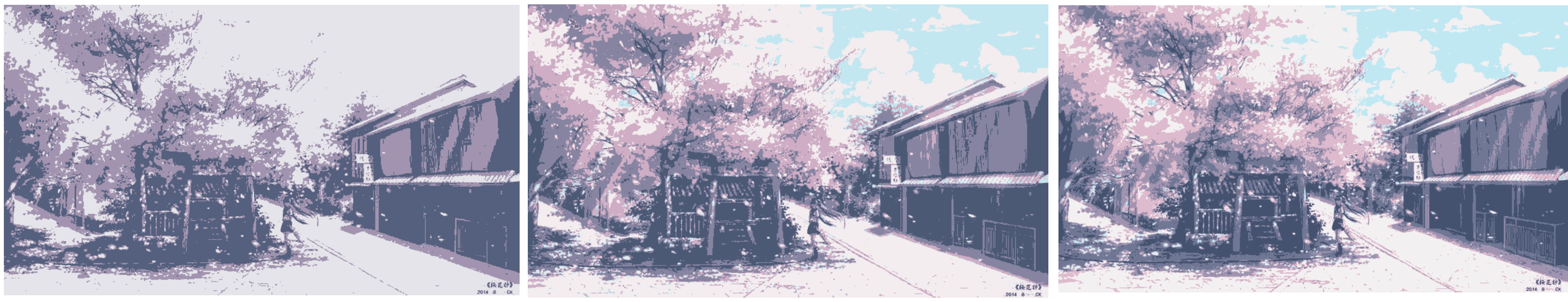


Figure 7 : Original Image

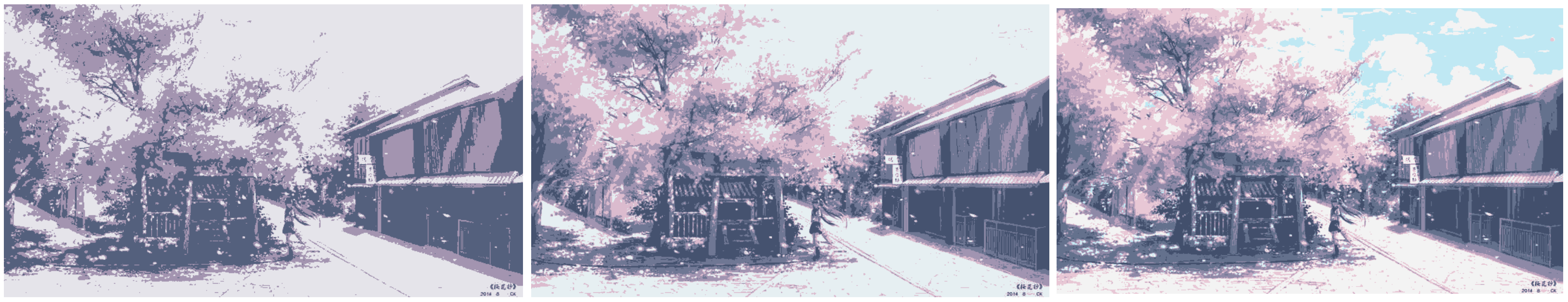
- If init_centroids = 'random'(k = {3, 5, 7}) , max_iter = 100 :



k = 3 k = 5 k = 7

Figure 8 : Compressed images if init_centroids = 'random'

- If init_centroids = 'in_pixels'(k = {3, 5, 7}) , max_iter = 100 :



k = 3 k = 5 k = 7

Figure 9 : Compressed images if init_centroids = 'in_pixels'

- Average runtime of 3 testing times :

Init_centroids / Clusters	random	in_pixels
3	(2.9s + 2.8s + 3.1s)/3 = 2.93s	(2.6s + 2.69s + 2.56s)/3 = 2.62s
5	(7.43s + 6.54s + 8.2s)/3 = 7.39s	(4.14s + 5.2s + 9.7s)/3 = 6.34s
7	(12.2s + 14.5s + 11.6s)/3 = 12.76s	(9.8s + 12.26s + 9.5s)/3 = 10.52s

Table 1 : Average running time

3. Comments:

About the result images:

As can see from the results image above, for both method of initializing centroids 'random' and 'in_pixels', we can see that the images have more color when the number of k_clusters increases. For k = 3, both of the images from 2 initialize method are almost the same as black and white picture, or more like dark and bright color, and the color which is more like the mixture between both of the color above. For k = 5 they are not much different from the k = 3 ones, but they seem to have more color and brighter. For k = 7 they are more colorful but the colors are still limited. So when we take a look at it, they are almost close to the original one but the limitation of colors still make the images have a bit difference from it.

Comparing both sets of images using 2 initialize centroids methods 'random' and 'in_pixels', they don't make so many differences. Because of the random nature of this program, the results are not completely the same to each other, but the numbers of colors for each each k_clusters cases still remains the same(same to the k_cluster number). For example, as can see from Figure 8 and 9 above:

- In the case k = 5, for 'random' method, the image has the blue color on it, while for the 'in_pixels' method the image doesn't have the blue color, but it has more tunes of pink color than the 'random' method. So both images still have the number of color is 5.

About the running time:

- Comparing the result based on the k_clusters, the running time of each k_cluster is increasing with the k_clusters.
- Comparing the result based on the init_centroids method, in this case, the 'in_pixel' method seems to run faster than the 'random' one. But still it is not a big difference.

VI. References:

[1] Pulkit Sharma, Introduction to K-Means Clustering Algorithm, https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/ , Access date: June 17th 2024.
[2] Wikipedia, Euclidean distance matrix, https://en.wikipedia.org/wiki/Euclidean_distance_matrix , Access date : June 17th 2024.
[3] Wikipedia, Euclidean_distance, https://en.wikipedia.org/wiki/Euclidean_distance , Access date : June 17th 2024.
[4] NumPy, https://numpy.org/ , Access date: June 14th 2024.
[5] Pillow, Pillow (PIL Fork) 10.3.0 documentation, https://pillow.readthedocs.io/en/stable/, Access date: June 14th 2024
[6] Matplotlib, Matplotlib: Visualization with Python, https://matplotlib.org/ , Access date : June 14th 2024.