VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY

HO CHI MINH CITY UNIVERSITY OF SCIENCE

INFORMATION TECHNOLOGY DEPARTMENT

Lab 01

Topic: Color Compression

Course: Applied Mathematics and Statistics

Student:
Nguyễn Công Tuấn
(22127436_22CLC03)

Lecturer:
Vũ Quốc Hoàng
Nguyễn Văn Quang Huy
Nguyễn Ngọc Toàn
Phan Thị Phương Uyên

19th June 2024

Contents

1 Information

STUDENT INFORMATION TABLE

Student	ID	Class
Nguyễn Công Tuấn	22127436	22CLC03

CHECK LIST

No.	Feature	Completion
1	Personal information: Full name, Student ID, class	Done
2	Implementation idea, description of functions	Done
3	Result images for each number of colors, $\mathbf{k = \{3, 5, 7\}}$ values	Done
4	Comments on the results.	Done
5	The report should include page numbers and references.	Done

2 Requirements

2.1 Description

In this project, you must implement a program to reduce the number of colours in an image using the K-Means algorithm.

2.2 Assignment requirements

- Use the Python programming language.
- Implement the required functions in a Jupyter Notebook file named MSSV.ipynb (replace MSSV with your student ID):
  - Read an image
  - Display an image
  - Save an image
  - Convert the image from 2D size (height, width, channels) to 1D size (height × width, channels)
  - Perform colour clustering using K-Means
  - Create a new image from the cluster centres obtained from K-Means
  - Main function:
    - Allow input of the image filename each time the program runs (use input() in Python)
    - Allow saving the image in at least two formats: pdf and png
- Allowed libraries:
  - NumPy (for matrix operations)
  - PIL (for image reading and writing)
  - matplotlib (for displaying images)

3 Idea for Implementation and Function Description

Idea for Implementation

Step 1: Reading the image to the 2D Numpy array then converting that array to a 1D array.

Step 2: Implementing the K-means algorithm:

Initialization:

- Start by selecting K initial colours (centroids) from the image pixels. This can be done randomly or by choosing K random pixels from the image.
- See function **initialize_centroids(img_1d, k_clusters, init_method)** in the source code.

Assignment:

- For each pixel in the image, calculate the **Euclidean distance** between the pixel's colour and each centroid.
- Assign the pixel to the cluster whose centroid is closest.
- See function **assign_labels(img_1d, centroids)** in the source code.

Update Centroids:

- Recalculate the centroids by computing the mean colour of all pixels assigned to each cluster (after doing the **Assignment** step).
- See function **update_centroids(img_1d, centroids, labels, k_clusters)** in the source code.

Repeat:

- Repeat the assignment and update steps until the centroids do not change significantly between iterations or until the maximum number of iterations is reached.
- In the source code, the loop (repeat the **Assignment** and **Update Centroids** steps) is within function **kmeans(img_1d, k_clusters, max_iter, init_centroids)**

Result:

- After convergence (the centroids do not change significantly between iterations or until the maximum number of iterations is reached), the centroids represent the reduced set of colours. The colour of its assigned centroid replaces each pixel.
- See function **generate_2d_img(img_2d_shape, centroids, labels)** in the source code.

Step 3: Generating the 2D image and then saving that image with the extension **.png** and **.pdf**.

Function Description

1. `read_img(img_path)`

   - Function to read an image from the specified path by using the **PIL** (Python Imaging Library) and then convert it into a **Numpy array** by using the Numpy library.
   - **Input**: img_path - a path to the image file.
   - **Output**: Numpy array (in numeric) representing the image.

2. `show_img(img_2d)`

   - Function to display a 2D image by using **matplotlib** library.
   - **Input**: img_2d - Numpy array (in numeric) representing the image.
   - **Output**: Displays the image by using **plt.imshow()** of the **matplotlib** library.

3. `save_img(img_2d, img_path)`

   - Function to save a 2D image (after converting the Numpy array to a real image by using the **Image.fromarray()** of the **PIL** library) to a specified file oath using the **PIL** library.
   - **Inputs**:
     - img_2d - The Numpy array representing the image.
     - img_path - Path where the image should be saved in.
   - **Output**: Saves the image file.

4. `convert_img_to_1d(img_2d)`

   - Function to converts a 2D image into a 1D array.
   - **Input**: img_2d - NumPy array representing the image.
   - **Output**: Flattened 1D array of shape.

5. `kmeans(img_1d, k_clusters, max_iter=100, init_centroids='random')`

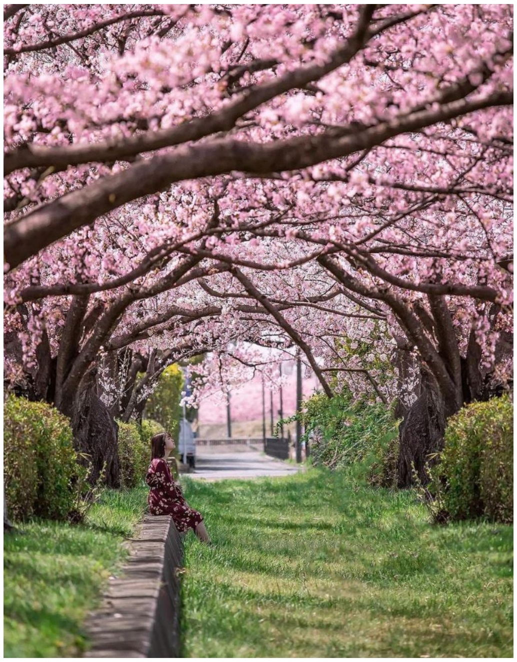
   - Function to perform the K-Means clustering algorithm on a 1D representation of image data.
   - **Inputs**:
     - img_1d - 1D NumPy array of image data.
     - k_clusters - Number of clusters (or colours) to reduce the image to.
     - max_iter - Maximum number of iterations for K-Means algorithm.
     - init_centroids - Initialization method for centroids (**'random'** or **'in_pixels'**).
   - **Outputs**:

  - centroids - Final centroids of the clusters.
  - labels - Labels (cluster assignments) for each pixel.

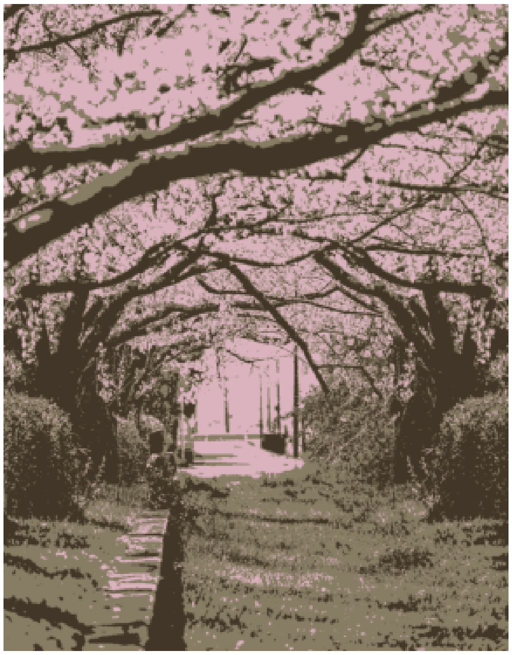
6. `generate_2d_img(img_2d_shape, centroids, labels)`
   - Function to generate a new 2D image based on the centroids and labels obtained from K-Means clustering.
   - **Inputs**:
     - img_2d_shape - Shape of the original 2D image.
     - centroids - Centroids of the clusters obtained from K-Means.
     - labels - Cluster labels assigned to each pixel.
   - **Output**: 2D NumPy array representing the compressed image.
7. `initialize_centroids(img_1d, k_clusters, init_method)`
   - Function to initialize centroids for K-Means clustering.
   - **Inputs**:
     - img_1d - 1D NumPy array of image data.
     - k_clusters - Number of clusters (or colours).
     - init_method - Method for initializing centroids ('random' or 'in_pixels').
   - **Output**: Initial centroids as a NumPy array.
8. `assign_labels(img_1d, centroids)`
   - Function to assign labels (cluster indices) to each pixel based on the closest centroid.
   - **Inputs**:
     - img_1d - 1D NumPy array of image data.
     - centroids - Current centroids of the clusters.
   - **Output**: Labels indicating each pixel's cluster assignment.
9. `update_centroids(img_1d, centroids, labels, k_clusters)`
   - Function to update centroids based on the current assignments of pixels to clusters.
   - **Inputs**:
     - img_1d - 1D NumPy array of image data.
     - centroids - Current centroids of the clusters.
     - labels - Current cluster labels for each pixel.
     - k_clusters - Number of clusters (or colours).

4 Image Results with Different Values of k

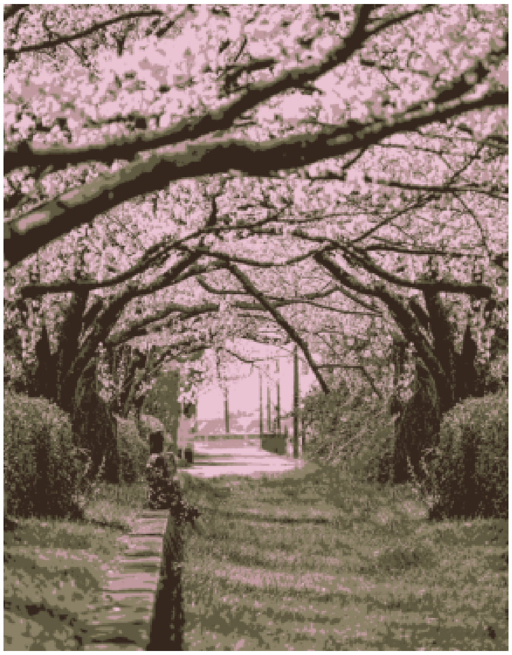
Here is the original image:



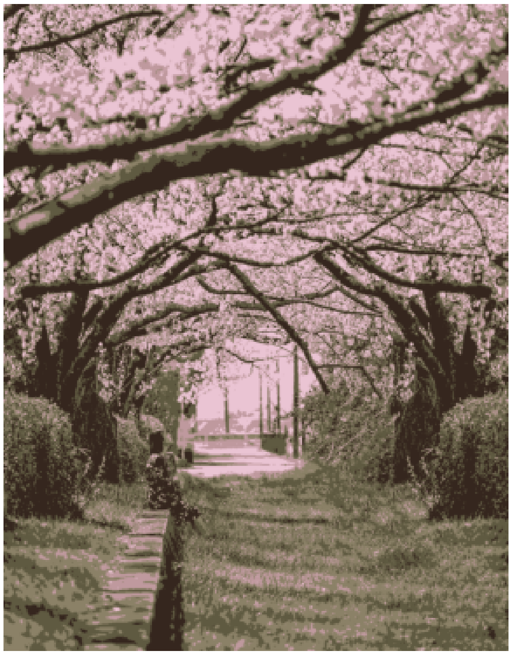
With k = 3:



With k = 5:



With k = 7:



5 Comments on the results

- As the k values increase, the image will have more colours and will look better. It will also become easier for the centroid to have no points assigned to it.

- The time required to run the k-means algorithm will increase as the k values or the max_iter variable increases.

6 Reference

[01] Introduction to K-Means Clustering Algorithm - Pulkit Sharma
[02] Understanding K-means Clustering in Machine Learning(With Examples) - Prashant Sharma
[03] k-means clustering - Wikipedia