VNU-HCM University Of Science

Faculty of Information Technology

PRACTICAL REPORT 1

COLOR COMPRESSION

APPLIED MATHEMATICS AND STATISTICS

MTH00051 – Applied Mathematics and Statistics

Ho Chi Minh City, 2024

VNU-HCM University of Science

Falcuty of Information Technology

22CLC08

PROJECT REPORT

COLOR COMPRESSION

INSTRUCTORS	STUDENTS
Mr. Vũ Quốc Hoàng	22127107-Nguyễn Thế Hiển
Mr. Nguyễn Văn Quang Huy	
Mr. Nguyễn Ngọc Toàn	
Mrs. Phan Thị Phương Uyên	

MTH00051 – Applied Mathematics and Statistics

Ho Chi Minh City, 2024

Acknowledgments

MTH0051 – Applied Mathematics and Statistics has been declared as an essential part of the Information Technology major. For us, it has been quite a time to develop our skills to become better and better in this field.

We want to express our genuine gratitude towards our Lab Instructors, Mr. Nguyễn Văn Quang Huy and Mrs. Phan Thị Phương Uyên for providing us a chance to gain more experience through this assignment, in addition to supporting us in the past few weeks throughout this course. Moreover, we also want to send our uttermost appreciation to our Main Lecturer, Mr. Vũ Quốc Hoàng for conducting excellent lessons and helping us with the familiarity in this practice.

Last but not least, we want to say thank you to our wonderful classmates of 22CLC08 who went with us on this real challenging journey. It is fascinating to acknowledge that students are here not only to analyze, to compete but also willing to learn and support others.

We hope that you will appreciate our coding work, our analysis, and our report in this Color Compression project. We also hope to receive unbiased feedback and expedient evaluations from our instructors to improve much more in the future.

Abstract

Digital images play a crucial role in today's information-driven world. However, the large file sizes associated with high-resolution images pose challenges for storage and transmission. To address this issue, image compression techniques are employed to reduce file sizes while preserving essential visual information. This project focuses on color compression using the K-Means clustering algorithm, implemented in Python using Jupyter Notebook and Google Colab.

The K-Means clustering algorithm is a widely used technique for data classification, effectively grouping similar data points into clusters. In the context of color compression, K-Means can be applied to reduce the number of colors in an image, thereby reducing its file size. The algorithm iteratively assigns each pixel to the nearest cluster centroid, resulting in a reduced color palette.

The project demonstrates the effectiveness of K-Means clustering in reducing image file sizes while maintaining visual quality. The implementation in Python using Jupyter Notebook and Google Collab provides a user-friendly and accessible approach to color compression..

TABLE OF CONTENT

LIST OF FIGURE

I. Introduction

1. Personal information:

Name: Nguyễn Thế Hiển

MSSV: 22127107

Class: 22CLC08

II. Implementation idea and description

1. Implementation idea

The recipe for k-means clustering:

- Determine the number of clusters (k): Choose the desired number of clusters, denoted as k.
- Initialize the centroids: Randomly select k points from the dataset as the initial centroids of the clusters.
- Calculate distances: For each data point, compute the distance to each of the k centroids.
- Assign clusters: Assign each data point to the cluster whose centroid is the closest.
- Update centroids: Recalculate the centroid of each cluster by taking the average of all the data points assigned to that cluster.
- Iterate until convergence: Repeat steps 3 to 5 until the centroids no longer change positions significantly, indicating that the clusters have stabilized.

2. Description of Function and Algorithm Explanation

- **read_img**

```
def read_img(img_path):
    # YOUR CODE HERE
    image = Image.open(img_path)
    image_2d = np.array(image)
    return image_2d
```

Picture 1: Read_img function

- **Purpose**: Read an image from the provided path and convert it into a numpy array in 2D format.

- **Parameters**:
  + img_path (str): Path to the image.
- **Returns**: Numpy array representing the 2D image.
- **Algorithm**:
  + Use the `PIL` library to open the image from the path.
  + Convert the image into a numpy array using `np.array`.

show_img

```
def show_img(img_2d):
    # YOUR CODE HERE
    plt.imshow(img_2d)
    plt.axis('off')
    plt.show()
```

Picture 2: Show_img function

- **Purpose**: Display a 2D image.
- **Parameters**:

  + `img_2d`: The 2D image to be displayed.

- **Algorithm**:

  + Use the `matplotlib` library to display the image and turn off axis display.

save_img

```
def save_img(img_2d, img_path):
    # YOUR CODE HERE
    image = Image.fromarray(img_2d.astype('uint8'))
    image.save(img_path)
```

Picture 3: Save_img

- **Purpose**: Save an image to the provided path.
- **Parameters**:

  + `img_2d`: The 2D image to be saved.

  + `img_path` (str): Path to save the image

- **Algorithms:**

  + `img_2d`: The 2D image to be saved.

+ `img_path` (str): Path to save the image.

convert_img_to_1d

```
def convert_img_to_1d(img_2d):
    # YOUR CODE HERE
    return img_2d.reshape((-1, img_2d.shape[2]))
```

Picture 4: Convert_img_to_1d function

- **Purpose**: Convert a 2D image to a 1D image.
- **Parameters**:

+ `img_2d`: The 2D image to be converted .

- **Returns**: Numpy array representing the 1D image.
- **Algorithm**:

+ Use the `reshape` method of numpy to change the shape of the image to 1D.

Kmean

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids='random'):
    # YOUR CODE HERE
    np.random.seed(42)

    if init_centroids == 'random':
        centroids = np.random.randint(0, 256, size=(k_clusters, img_1d.shape[1]))
    elif init_centroids == 'in_pixels':
        centroids = img_1d[np.random.choice(img_1d.shape[0], k_clusters, replace=False)]
    else:
        raise ValueError("init_centroids phải là 'random' hoặc 'in_pixels'")

    for _ in range(max_iter):
        distances = np.linalg.norm(img_1d[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)

        new_centroids = np.array([img_1d[labels == i].mean(axis=0) if np.any(labels == i) else centroids[i]
                                  for i in range(k_clusters)])
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return centroids, labels
```

Picture 5: Kmean function

- **Purpose**: Perform the K-means algorithm on image data to find color clusters.
- **Parameters**:

+ `img_1d` (np.ndarray): 1D image data.

+ `k_clusters` (int): Number of clusters.

+ `max_iter` (int): Maximum iterations.

+ `init_centroids` (str): Method to initialize centroids (`'random'` or `'in_pixels'`).

- **Returns**:

+ `centroids` (np.ndarray): Array of cluster centroids.

+ `labels` (np.ndarray): Array of labels for each pixel.

- **Algorithm**:
  - Initialize centroids:

    + If `init_centroids` is `'random'`, initialize centroids with random values in the range 0-255.

    + If `init_centroids` is `'in_pixels'`, randomly select pixels from the image as centroids.

  - Iterate for a maximum of `max_iter` times:

    + Compute the Euclidean distance between each pixel and centroids.

    + Assign each pixel to the nearest cluster.

    + Update the centroids by calculating the mean of the pixels in each cluster.

    + Check for convergence: if centroids do not change, break the loop

- generate_2d_img

```
def generate_2d_img(img_2d_shape, centroids, labels):
    # YOUR CODE HERE
    new_img_1d = centroids[labels].astype(np.uint8)
    # 1D (height * width, channels) -> 2D (height, width, channels)
    new_img_2d = new_img_1d.reshape(img_2d_shape)
    return new_img_2d
```

Picture 6: Generate_2d_img

- **Purpose**: Generate a 2D image from cluster centroids and pixel labels.
- **Parameters**:

+ `img_2d_shape` (tuple): Original shape of the image (height, width, number of color channels).

+ `centroids` (np.ndarray): Array of color centroids.

+ `labels` (np.ndarray): Array of pixel labels.

- **Returns**: New 2D image.
- **Algorithm**:

+ Create a new 1D array by replacing each pixel with the centroid color of its cluster.

+ Reshape the 1D array back to the original 2D shape of the image.

Run Test_function

```
def test_functions():
    img_path = "path/to/your/test/image.jpg"

    img_2d = read_img(img_path)
    assert img_2d is not None, "Failed to read image"
    assert len(img_2d.shape) == 3 and img_2d.shape[2] == 3, "Image is not a valid RGB image"

    print("Displaying original image:")
    show_img(img_2d)

    img_1d = convert_img_to_1d(img_2d)
    assert img_1d.shape[0] == img_2d.shape[0] * img_2d.shape[1], "Conversion to 1D failed"
    assert img_1d.shape[1] == 3, "Conversion to 1D failed: number of channels should be 3"

    k_clusters = 5
    max_iter = 100
    centroids_random, labels_random = kmeans(img_1d, k_clusters, max_iter, init_centroids='random')
    assert centroids_random.shape == (k_clusters, img_1d.shape[1]), "K-Means centroids shape mismatch for 'random'"
    assert labels_random.shape == (img_1d.shape[0],), "K-Means labels shape mismatch for 'random'"

    new_img_2d_random = generate_2d_img(img_2d.shape, centroids_random, labels_random)
    assert new_img_2d_random.shape == img_2d.shape, "Generated image shape mismatch for 'random'"

    print("Displaying new image with 'random' initialization:")
    show_img(new_img_2d_random)

    save_img(new_img_2d_random, "test_image_result_random.png")
    save_img(new_img_2d_random, "test_image_result_random.pdf")

    centroids_in_pixels, labels_in_pixels = kmeans(img_1d, k_clusters, max_iter, init_centroids='in_pixels')
    assert centroids_in_pixels.shape == (k_clusters, img_1d.shape[1]), "K-Means centroids shape mismatch for 'in_pixels'"
    assert labels_in_pixels.shape == (img_1d.shape[0],), "K-Means labels shape mismatch for 'in_pixels'"

    new_img_2d_in_pixels = generate_2d_img(img_2d.shape, centroids_in_pixels, labels_in_pixels)
    assert new_img_2d_in_pixels.shape == img_2d.shape, "Generated image shape mismatch for 'in_pixels'"

    print("Displaying new image with 'in_pixels' initialization:")
    show_img(new_img_2d_in_pixels)

    save_img(new_img_2d_in_pixels, "test_image_result_in_pixels.png")
    save_img(new_img_2d_in_pixels, "test_image_result_in_pixels.pdf")

    print("All tests passed.")
```

Picture 7: Test_function

When you run test_functions(), you will:

- Be prompted to enter the path to the image file (if you update the img_path in the function).

- See the original image displayed.
- Observe any output confirming each step of the process, such as successful conversion, clustering, and generation.
- See the new image displayed after K-Means clustering.
- Have the new image saved as test_image_result.png and test_image_result.pdf in your current working directory.
- Finally, see the message "All tests passed."

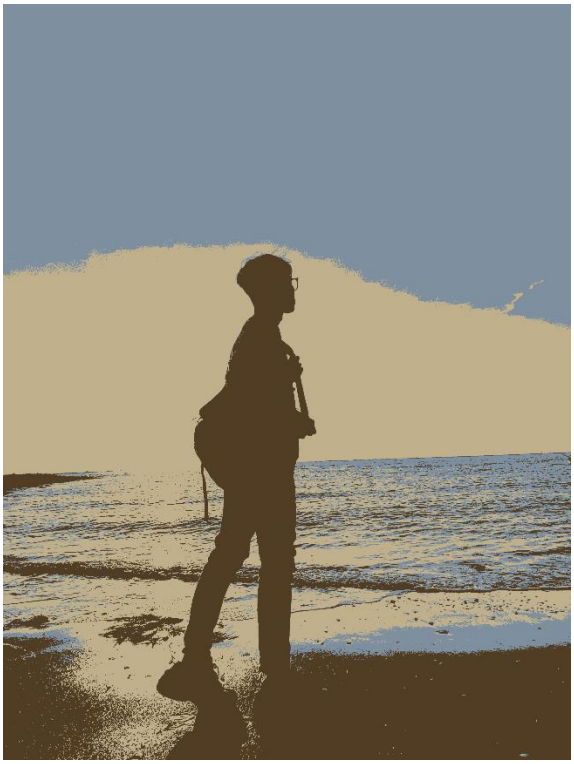
III. Demo and Comment result

➢ Test program

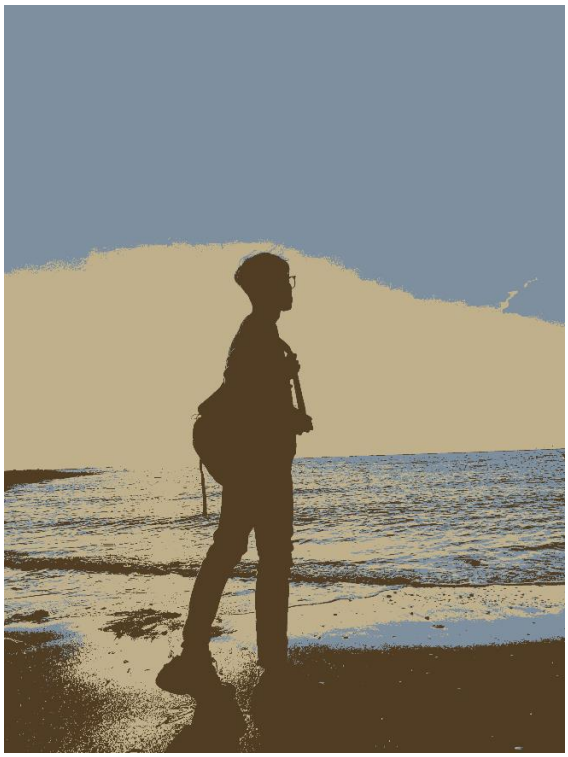
➢ 

Picture 8: Test

- K = 3:

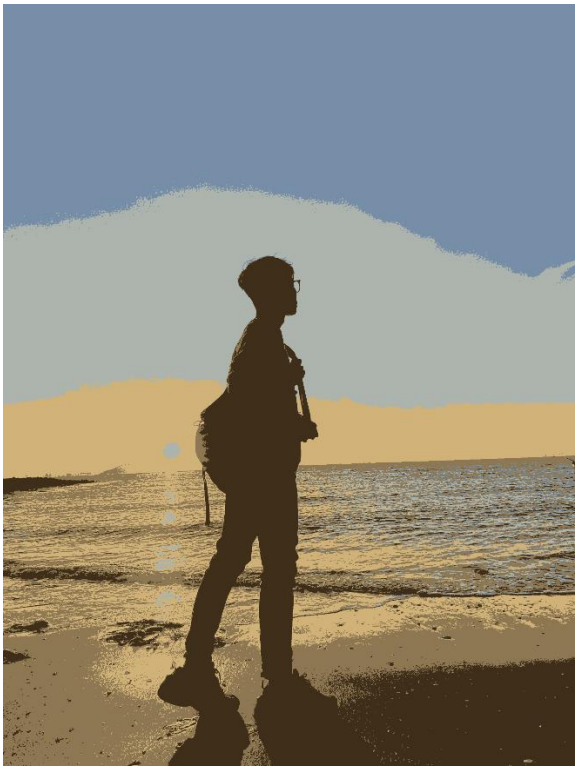


Picture 9: In_pixesl with K = 3



Picture 10: Random with K = 3

- K = 5:

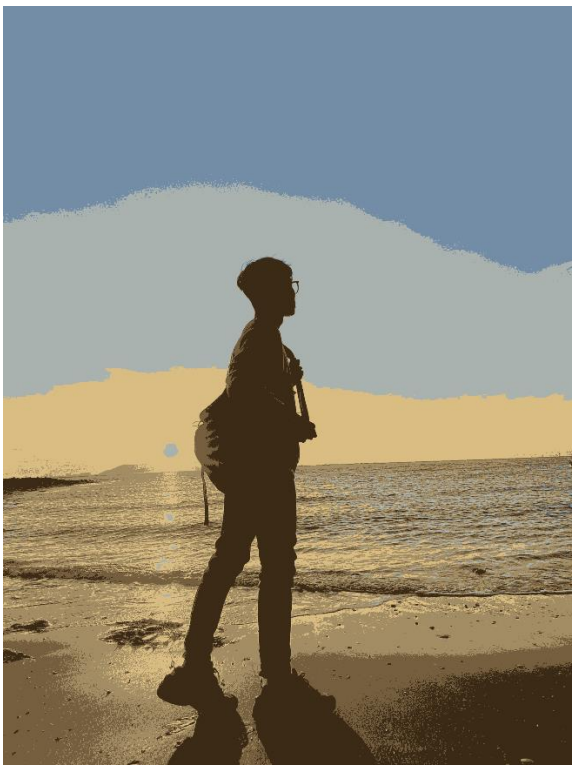


Picture 11: In_pixels with K = 5

Picture 12: Random with K = 5

- K = 7:

Picture 13: In_pixels with K = 7



Picture 14: Random with K = 7

➢ Comment:

Results Commentary:

Random Initialization:

- Centroids are randomly chosen from the entire color space (0-255), which can help the compression converge quickly but might not always be optimal.

In-Pixels Initialization:

- Centroids are chosen from actual color points in the image, which can start with centroids closer to the actual colors of the image, but may take more time to converge.

Comparison:

- Random initialization may be better for images with diverse colors: Random initialization will cover the entire color space, helping to find suitable centroids more quickly.
- In-pixels initialization may be better for images with fewer or more uniform colors: Starting with centroids from actual image colors can help converge quickly to the significant colors.

Summary:

- Random Initialization:
  - Speed: Faster for diverse color images.
  - Efficiency: Can be less optimal for compression quality in some cases.
- In-Pixels Initialization:
  - Speed: Slower for diverse color images.
  - Efficiency: Often better compression quality for images with fewer or uniform colors.

IV. References

Lab: https://github.com/NgocTien0110/Applied-Mathematics-and-Statistics/blob/main/lab02_project01/20127641.ipynb

Idea: https://en.wikipedia.org/wiki/K-means_clustering

Library:

Numpy Documentation: https://numpy.org/doc/

Matplotlib3.9.0:

https://matplotlib.org/stable/tutorials/pyplot.htmldocumentaion

Pilow_PIL: https://pillow.readthedocs.io/en/stable/