ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Applied Mathematics and Statistics

Project 1: Color Compression

Student:
Đinh Nguyễn Gia Bảo
(22127027) 22CLC03

Teacher:
GV. Vũ Quốc Hoàng
GV. Phan Thị Phương Uyên
Gv.Nguyễn Ngọc Toàn
Gv.Nguyễn Văn Quang Huy

Ngày 19 tháng 6 năm 2024

Mục lục

1 Overall information

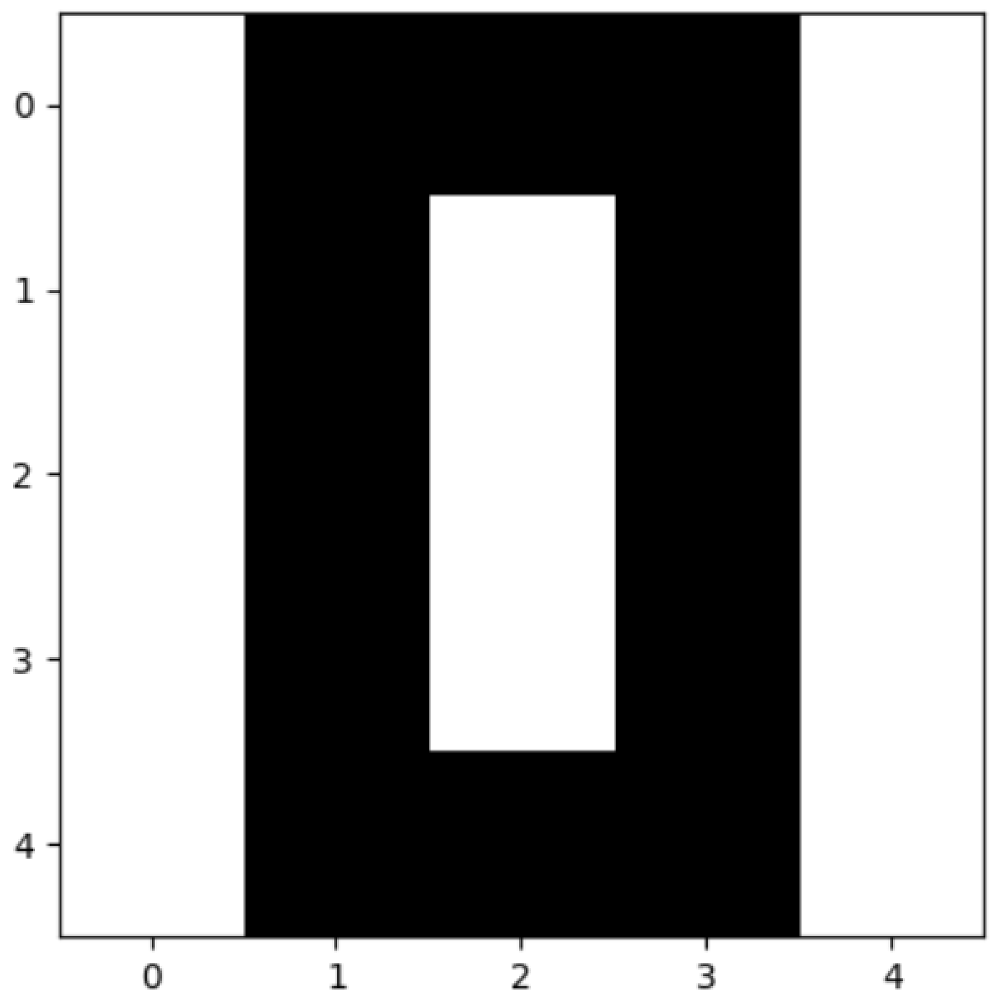
1.1 Project 1: Color Compression

1.1.1 Brief Description

Một bức ảnh có thể lưu trữ dưới ma trận của các điểm ảnh. Có nhiều loại ảnh được sử dụng trong thực tế, ví dụ: ảnh xám, ảnh màu,...
Đối với ảnh xám, một điểm ảnh sẽ là được biểu diễn bằng giá trị không âm.
Ví dụ ta có thể dùng ma trận này:

$$\begin{bmatrix} 255 & 0 & 0 & 0 & 255 \\ 255 & 0 & 255 & 0 & 255 \\ 255 & 0 & 255 & 0 & 255 \\ 255 & 0 & 255 & 0 & 255 \\ 255 & 0 & 0 & 0 & 255 \end{bmatrix}$$

có thể biểu diễn cho ảnh xám có nội dung như sau:

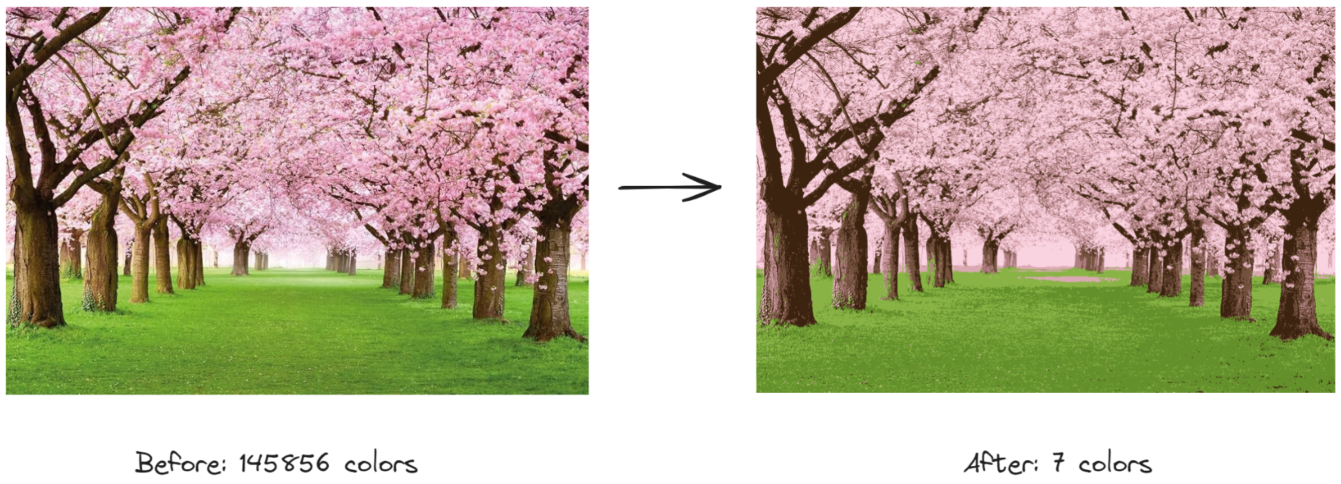


Hình 1: Ảnh xám

Ảnh màu được sử dụng phổ biến là ảnh RGB, trong đó, mỗi điểm ảnh sẽ lưu trữ 3 thông tin kênh màu (mỗi kênh màu 1 byte) là: R (red - đỏ), G (green - xanh lá), B (blue - xanh dương).

Như vậy, số màu trong ảnh RGB có thể là $256^3 \approx 1.7 \times 10^7$. Với số lượng màu khá lớn, khi lưu trữ ảnh có thể sẽ tốn chi phí lưu trữ. Do đó bài toán đặt ra là giảm số lượng màu để biểu diễn ảnh sao cho nội dung ảnh được bảo toàn nhất có thể.

Cho ảnh như sau:



Hình 2: Photo by Smileus on Getty Images

Trong ví dụ trên, số lượng màu cho ảnh ban đầu là 145856 màu. Sau khi giảm số lượng màu xuống còn 7, ảnh không còn được chi tiết nhưng cơ bản vẫn bảo toàn nội dung của ảnh ban đầu.

Để thực hiện giảm số lượng màu, ta cần tìm ra các đại diện có thể thay thế cho một nhóm màu. Cụ thể trong trường hợp ảnh RGB, ta cần thực hiện gom nhóm các pixel ($\mathbb{R}^3$) và chọn ra đại diện cho từng nhóm. Như vậy, bài toán trên trở thành gom nhóm các vec-tơ.

Trong đồ án này, bạn được yêu cầu cài đặt chương trình giảm số lượng màu cho ảnh sử dụng thuật toán K-Means.

1.1.2 Programming language and environment

- Python (.ipynb)
- Visual Studio Code

1.1.3 Visualization

All the visualization are display on the IDE console and report.

1.1.4 Assignment Requirements Assessment

No.	Specifications	Completion
1	Read image	100%
2	Show image	100%
3	Save image	100%
4	Convert image from 2D to 1D	100%
5	Kmeans algorithm	100%
6	Generate 2D image	100%
7	Analysis and experimentation	100%
	Report	100%

1.1.5 Code Library and install instruction

To run code properly, you need to turn your commandline and have these Library installed first:

- **Numpy**: pip install numpy
- **PIL**: pip install Pillow
- **matplotlib**: pip install matplotlib

2 Solution description

Color compression is a technique used to reduce the number of distinct colors in an image while preserving its visual appearance as much as possible. One effective method for achieving this is through the use of the K-means clustering algorithm

2.1 How to apply Kmeans to this Problem

Steps Involved in Color Compression Using K-means:

Image Preprocessing:

- Read and load the image into a format suitable for processing.
- Convert the image into a two-dimensional array where each pixel is represented by its color values (e.g., RGB).

Flattening the Image: Reshape the 2D array of pixels into a 1D array where each entry represents a pixel's color values. This transformation makes it easier to apply the K-means algorithm.

K-means Clustering:

- **Initialization**: Select `k` initial centroids, which can be done randomly or by choosing `k` random pixels from the image.
- **Assignment Step**: For each pixel, calculate the distance to each centroid and assign the pixel to the nearest centroid.

- **Update Step**: Recalculate the centroids as the mean of all pixels assigned to each centroid.
- **Iteration**: Repeat the assignment and update steps until the centroids no longer change significantly (convergence).

Reconstructing the Image:

- After the K-means algorithm converges, each pixel is replaced by the centroid of the cluster to which it was assigned.
- Reshape the 1D array back into the 2D image format.

Output: The resulting image has reduced colors, represented by the `k` centroids, effectively compressing the color palette of the image.

2.2 Benefits

Reduced Storage: The compressed image requires less storage space due to the limited number of colors.

Faster Processing: Subsequent image processing tasks can be faster on the compressed image due to fewer colors.

Visual Quality: Maintains visual quality by approximating the original colors with cluster centroids.

Example Applications:

- **Image Compression**: Reducing file size for storage and transmission.
- **Graphics and Visualization**: Simplifying color schemes for visualizations and graphics.
- **Pattern Recognition**: Preprocessing step for computer vision tasks to reduce computational complexity.

3 Implementation description

3.1 Pre-processing image

In this report, we explore a set of Python functions designed for basic image processing tasks using the `PIL` (Pillow) and `matplotlib` libraries. These functions facilitate operations such as reading an image from a file, displaying it, saving it in different formats, and converting it between 2D and 1D arrays for further analysis.

3.1.1 Read image

- **Purpose**: This function reads an image file from the specified `img_path` and converts it into a 2D NumPy array.
- **Implementation**: It utilizes the `Image.open()` function from `PIL` to open the image and `np.array()` to convert it into a NumPy array.
- **Returns**: The function returns the 2D array representation of the image.

3.1.2 Show image

- **Purpose**: Displays the image represented by the 2D NumPy array `img_2d`.
- **Implementation**: It uses `matplotlib.pyplot.imshow()` to display the image and `plt.axis('off')` to turn off the axis for a cleaner display.
- **Visualization**: The image is shown immediately after calling this function.

3.1.3 Save image

- **Purpose**: Converts the 2D NumPy array `img_2d` back into an image format and saves it to the specified `img_path` in both PNG and PDF formats.
- **Implementation**: It converts the array using `Image.fromarray()` and then saves it using `img.save()` with the appropriate file extensions.
- **Usage**: This function is essential for saving processed images for further use or distribution.

3.1.4 Convert 2D array to 1D

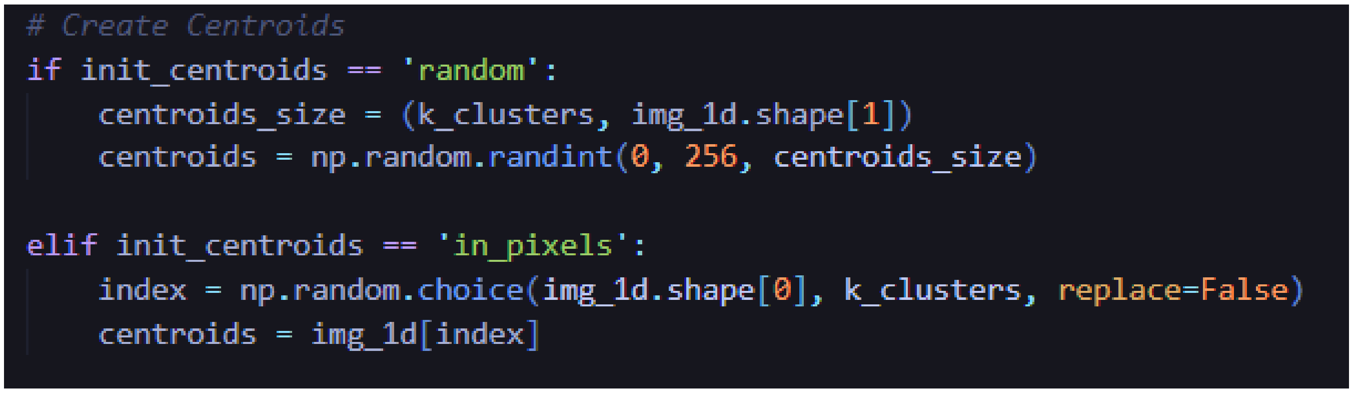
- **Purpose**: Reshapes the 2D image array `img_2d` into a 1D array.
- **Implementation**: Calculates the dimensions (height, width, channels) of `img_2d` and reshapes it using `reshape((height * width, channels))`.
- **Returns**: The function returns the flattened 1D representation of the image.

3.2 Kmeans algorithm

The functions presented here implement the K-means clustering algorithm for image segmentation. Image segmentation involves partitioning an image into distinct regions based on similarity, which in this case, is defined by color similarity. K-means clustering is used to group pixels into clusters based on their color values and assign each pixel to its nearest cluster centroid. Here's a detailed description of each function [1].

- **Purpose**: This function performs the K-means clustering algorithm on a flattened 1D representation of an image (`img_1d`) to identify `k_clusters` clusters of similar colors.
- **Parameters**:
  - `img_1d`: 1D NumPy array representing the flattened image pixels.
  - `k_clusters`: Number of clusters to identify.
  - `max_iter`: Maximum number of iterations to run the algorithm.
  - `init_centroids`: Method to initialize centroids ('random' or 'in_pixels').
- **Steps**:

1. **Initialization**: Depending on `init_centroids`, centroids are initialized randomly or selected from pixels (`in_pixels`).[2]



```python
# Create Centroids
if init_centroids == 'random':
    centroids_size = (k_clusters, img_1d.shape[1])
    centroids = np.random.randint(0, 256, centroids_size)

elif init_centroids == 'in_pixels':
    index = np.random.choice(img_1d.shape[0], k_clusters, replace=False)
    centroids = img_1d[index]
```

Hình 3: Initialization centroids random/in pixels

2. **Iteration**:
   - Iteratively assigns pixels to the nearest centroid and updates centroids based on pixel assignments.
   - Stops if centroids converge (change less than a threshold).
3. **Convergence Check**: Uses `np.allclose()` to check if centroids have converged.

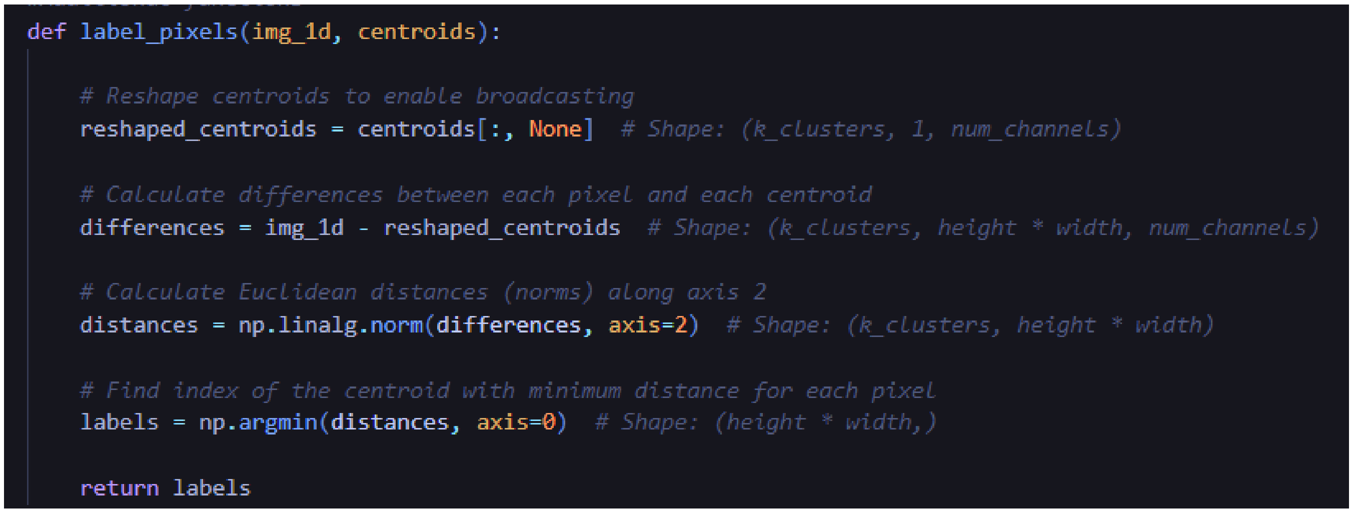
- **Returns**:
  - `centroids`: Final centroids representing the color clusters.
  - `labels`: Cluster labels assigned to each pixel.

3.3 Generate image 2D

- **Purpose**: Reconstructs the segmented image from centroids and assigned labels.
- **Parameters**:
  - `img_2d_shape`: Shape of the original image (height, width, channels).
  - `centroids`: Centroids of color clusters.
  - `labels`: Cluster labels assigned to each pixel.
- **Steps**:
  1. **Replacement**: Each pixel in the original image is replaced with its corresponding centroid based on `labels`.
  2. **Reshaping**: Reshapes the 1D array of centroids back to the 2D shape of the original image.
- **Returns**: `new_img`: Segmented image represented as a 2D NumPy array of integer values.

3.4 Additional Functions

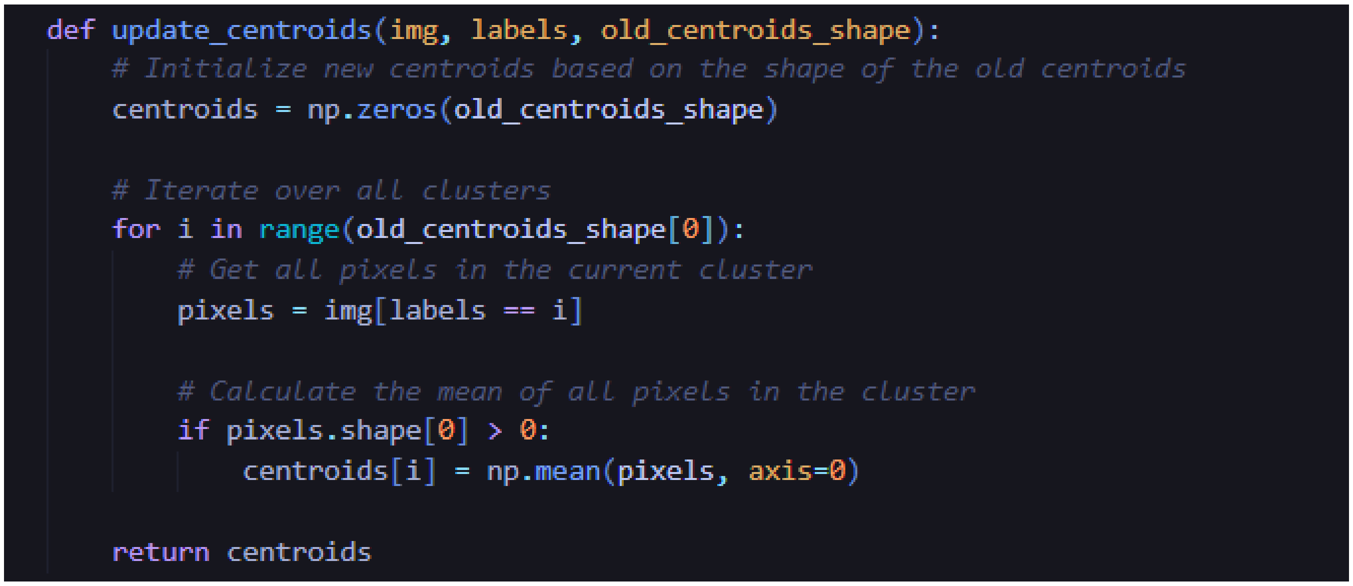
- Label pixels: Computes the closest centroid for each pixel based on Euclidean distance.[1]



```python
def label_pixels(img_1d, centroids):

    # Reshape centroids to enable broadcasting
    reshaped_centroids = centroids[:, None]  # Shape: (k_clusters, 1, num_channels)

    # Calculate differences between each pixel and each centroid
    differences = img_1d - reshaped_centroids  # Shape: (k_clusters, height * width, num_channels)

    # Calculate Euclidean distances (norms) along axis 2
    distances = np.linalg.norm(differences, axis=2)  # Shape: (k_clusters, height * width)

    # Find index of the centroid with minimum distance for each pixel
    labels = np.argmin(distances, axis=0)  # Shape: (height * width,)

    return labels
```

Hình 4: Label pixels

- Update centroids: Updates centroids based on the mean color value of pixels assigned to each cluster.



```python
def update_centroids(img, labels, old_centroids_shape):
    # Initialize new centroids based on the shape of the old centroids
    centroids = np.zeros(old_centroids_shape)

    # Iterate over all clusters
    for i in range(old_centroids_shape[0]):
        # Get all pixels in the current cluster
        pixels = img[labels == i]

        # Calculate the mean of all pixels in the cluster
        if pixels.shape[0] > 0:
            centroids[i] = np.mean(pixels, axis=0)

    return centroids
```

Hình 5: update centroids

3.5 Main function

- **Input Gathering**: Collects user input for image path, number of clusters (`k_clusters`), maximum iterations (`max_iter`), and centroid initialization type (`init_centroids`).

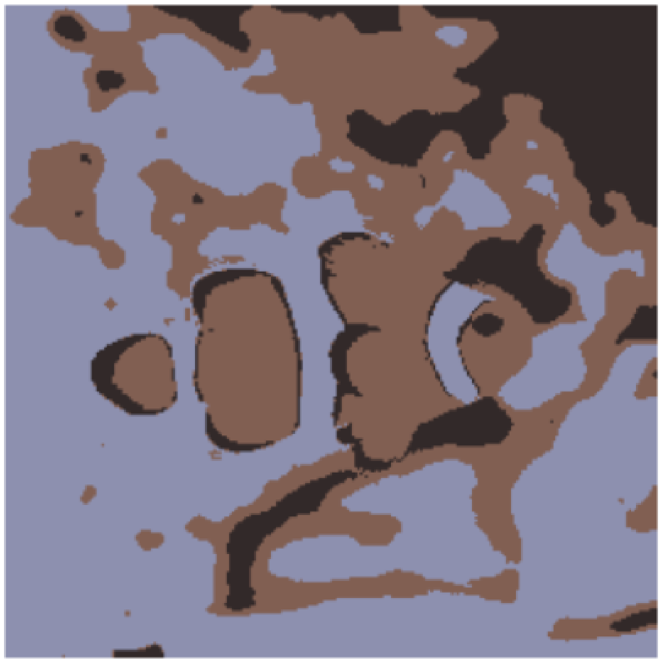
- **Read and Display Original Image**: Reads the image from `img_path` and displays it using `show_img(img_2d)`.
- **Convert Image to 1D**: Converts the 2D image array (`img_2d`) into a flattened 1D array (`img_1d`) using `convert_img_to_1d(img_2d)`.
- **Perform K-means Clustering**: Executes K-means clustering on `img_1d` to identify `k_clusters` clusters of similar colors. It initializes centroids based on `init_centroids`, iterates to optimize centroids, and checks for convergence using `kmeans(img_1d, k_clusters, max_iter, init_centroids`
- **Generate Clustered Image**: Reconstructs the segmented image using `centroids` and `labels` with `generate_2d_img(img_2d.shape, centroids, labels)`.
- **Display and Save Clustered Image**: Displays the segmented image using `show_img(new_img_2d)` and saves it to 'Result' in both PNG and PDF formats using `save_img(new_img_2d, 'Result')`.

4 Statistics and performance assessment

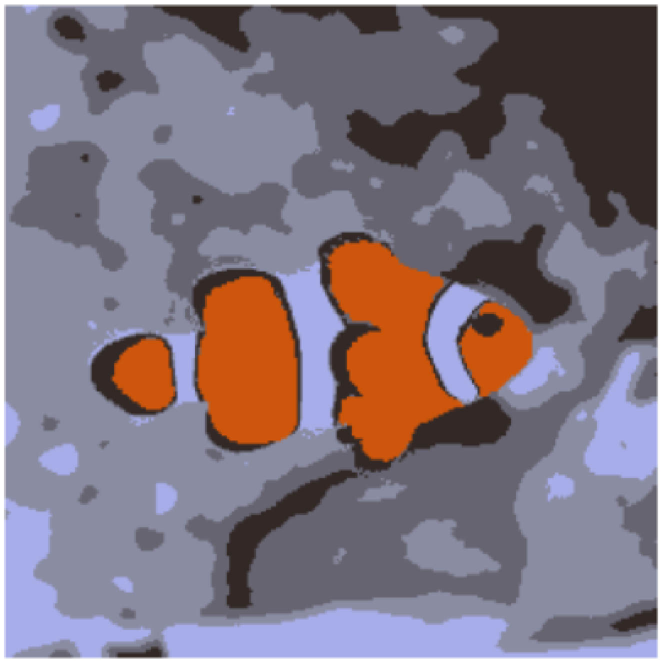
This performance assessment evaluates the execution time of the K-means image segmentation algorithm as implemented in the provided test code. The goal is to analyze how the number of clusters (`k_clusters`) affects the computation time.

Test Conditions

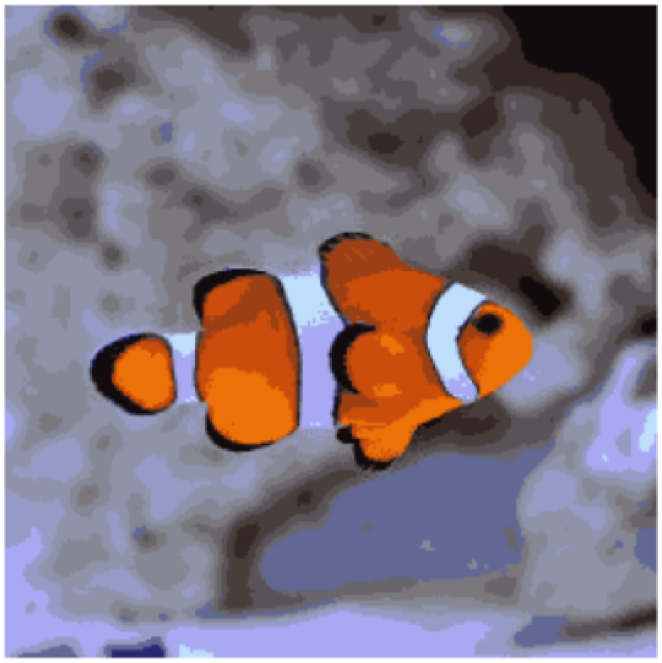
- **Image**: `clownfish.jpg`
- **Maximum Iterations**: 100
- **Centroid Initialization Method**: `in_pixels`
- **Tested Cluster Numbers**: 3, 5, 7, 16



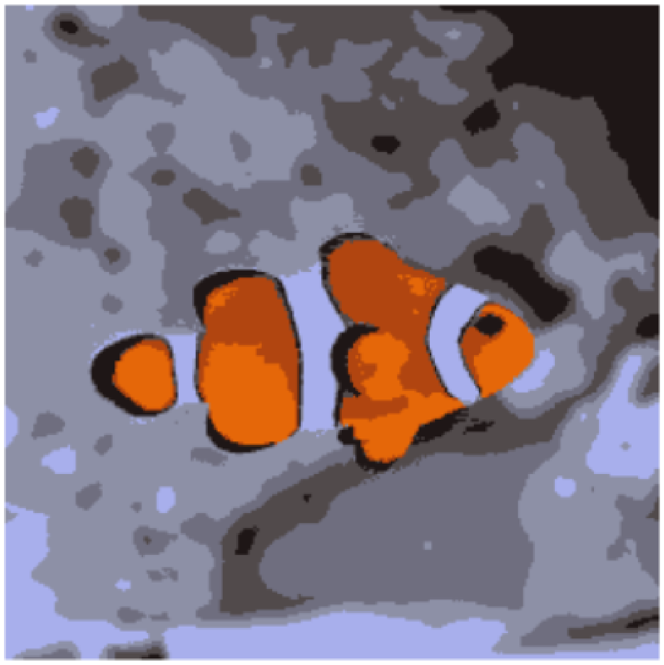
Hình 6: k = 3, type = random



Hình 7: k = 5, type = random



Hình 9: k = 16, type = random



Hình 8: k = 7, random

Hình 10: k = 16, type = in pixels

Execution Time Results

- **k = 3**: 0.3 seconds
- **k = 5**: 0.4 seconds
- **k = 7**: 1.1 seconds
- **k = 16**: 3.2 seconds

Analysis The computation time increases with the number of clusters (`k_clusters`). This is expected because:

1. **Centroid Initialization**: Initializing more centroids requires more operations.
2. **Distance Calculations**: Each iteration involves computing the distance between each pixel and all centroids. More clusters mean more distance calculations per pixel.
3. **Centroid Updates**: Updating centroids involves computing the mean of all pixels assigned to each centroid, which becomes more complex with more clusters.

Performance assessment **Small Cluster Numbers (3.1)**:

- When the number of clusters is small (5 or fewer), 'random' initialization does not produce as good a display quality as 'in_pixels.' The image tends to lean towards a certain color, usually darker tones.

- For larger cluster numbers, the difference is not too significant, but 'in_pixels' still shows a slight advantage in displaying colors that appear less frequently in the image in some cases.

- **Scalability**: The algorithm's performance degrades non-linearly as the number of clusters increases. The relationship between the number of clusters and execution time is more than linear, indicating significant overhead in distance calculation and centroid updating.

- **Practical Considerations**: For real-time or large-scale applications, the number of clusters should be chosen carefully to balance between segmentation quality and computation time.

Recommendations

- **Optimization**: Investigate optimization techniques such as using more efficient data structures, parallel processing, or optimized libraries.

- **Parameter Tuning**: Perform a parameter tuning exercise to find the optimal balance between the number of clusters and acceptable computation time for the specific application context.

4.1 Video demo

Further demonstration, you can watch my demo video right here: Google Drive

5 Reference

[1] AppMath-Project-KmeansClustering Github, Aug 18 2020

[2] numpy.random() Trong Python: Hướng Dẫn Chi Tiết, Website Online

[3] Github Color Compression, 2023