Image Quantization using K-Means Clustering

Introduction

This project implements image quantization using the K-Means clustering algorithm in Python. Image quantization reduces the number of distinct colors in an image, which can be useful for image compression and stylization. The implementation includes functions for initializing centroids, assigning pixels to centroids, updating centroids, and creating the quantized image. Additionally, the compression ratio between the original and quantized images is calculated.

Prerequisites

The following libraries are required for this project:

- numpy: For numerical operations.
- opencv-python: For image reading and writing.
- matplotlib: For image display.

Code Overview

1. Import Libraries

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

2. Initialize Centroids

Image Quantization using K-Means Clustering

This function initializes the centroids randomly from the image pixels.

```
def initialize_centroids(pixels, k):
    np.random.seed(42)
    random_indices = np.random.choice(pixels.shape[0], k, replace=False)
    centroids = pixels[random_indices]
    return centroids
```

3. Assign Pixels to Centroids

This function assigns each pixel to the nearest centroid.

```
def assign_pixels_to_centroids(pixels, centroids):
    distances = np.linalg.norm(pixels[:, np.newaxis] - centroids, axis=2)
    return np.argmin(distances, axis=1)
```

4. Update Centroids

This function updates the centroids by calculating the mean of the assigned pixels.

```
def update_centroids(pixels, labels, k):
    new_centroids = np.array([pixels[labels == i].mean(axis=0) for i in range(k)])
    return new_centroids
```

5. K-Means Clustering Algorithm

The main function that performs K-Means clustering.

```
def kmeans(pixels, k, max_iters=100):
    centroids = initialize_centroids(pixels, k)
    for i in range(max_iters):
        labels = assign_pixels_to_centroids(pixels, centroids)
        new_centroids = update_centroids(pixels, labels, k)
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
    return centroids, labels
```

6. Create Quantized Image

This function creates the quantized image by mapping the pixels to their assigned centroids.

```
def create_quantized_image(pixels, labels, centroids, height, width):
    quantized_pixels = centroids[labels]
    quantized_image = quantized_pixels.reshape(height, width, 3)
    return quantized_image
```

7. Calculate Compression Ratio

This function calculates the compression ratio between the original and quantized images.

```
def calculate_compression_ratio(original_image_path, quantized_image_path):
    original_size = os.path.getsize(original_image_path)
    quantized_size = os.path.getsize(quantized_image_path)
    compression_ratio = original_size / quantized_size
    return compression_ratio
```

8. Main Code

The main part of the code reads the image, applies K-Means clustering, and displays the results.

```
# Read the image using OpenCV
image_path = 'pic.jpg'
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Get image dimensions
height, width, _ = image.shape

# Reshape the image to a 2D array of pixels
pixels = image.reshape(-1, 3)
```

Image Quantization using K-Means Clustering

```
# Take the number of clusters (k) as input from the user
k = int(input('Enter the number of colors (k): '))

# Apply K-Means clustering to the image pixels
centroids, labels = kmeans(pixels, k)

# Create the quantized image
quantized_image = create_quantized_image(pixels, labels, centroids, height, width)

# Save quantized image
quantized_image_path = f'quantized_image_{k}.png'
quantized_image_bgr = cv2.cvtColor(quantized_image.astype(np.uint8), cv2.COLOR_RGB2BGR)
cv2.imwrite(quantized_image_path, quantized_image_bgr)

# Calculate compression ratio
compression_ratio = calculate_compression_ratio(image_path, quantized_image_path)

# Display the original and quantized images side by side
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image)
plt.axis('off')
```

Image Quantization using K-Means Clustering

```
plt.subplot(1, 2, 2)
plt.title(f'Quantized Image with k={k}')
plt.imshow(quantized_image.astype(np.uint8))
plt.axis('off')

print(f'Compression ratio for K={k}: {compression_ratio:.2f}')
plt.show()
```

Explanation

1. Initialize Centroids: The centroids are initialized randomly from the pixels to serve as the starting points for clustering.

2. Assign Pixels: Each pixel is assigned to the nearest centroid based on Euclidean distance.

3. Update Centroids: The centroids are updated by calculating the mean of all pixels assigned to each centroid.

4. K-Means Algorithm: The algorithm iteratively assigns pixels to centroids and updates the centroids until convergence or reaching the maximum number of iterations.

5. Quantized Image: The quantized image is created by mapping each pixel to its centroid.

6. Compression Ratio: The compression ratio is calculated by comparing the file sizes of the original and quantized images.

Results

Image Quantization using K-Means Clustering

The original and quantized images are displayed side by side, and the compression ratio is printed to the console. The quantized image shows a reduction in the number of colors, leading to potential file size reduction.

This implementation provides a straightforward way to perform image quantization using K-Means clustering and demonstrates the benefits of color reduction for image compression.