Image Compression using Linear Algebra

Image compression is a fundamental task in digital image processing, aiming to reduce the storage size of images while maintaining their visual quality. One powerful method for achieving this is through the use of linear algebra, particularly the Singular Value Decomposition (SVD). This technique allows us to represent an image matrix in a more compact form, effectively discarding less important information while retaining the essential features.

The following Python code demonstrates how to compress an image using SVD. The process involves decomposing the image into its constituent components, compressing these components by retaining only a subset of the most significant features, and then reconstructing the compressed image. This approach can be applied to both grayscale and color images, offering a flexible and mathematically sound method for reducing image size.

```python
import numpy as np
from PIL import Image
import argparse
import os

def compress_image(image_path, compression_factor=0.1):
    # Open the image and convert it to a numpy array
    img = Image.open(image_path)
    img_array = np.array(img, dtype=float)

    # Check if the image is grayscale or color
    if len(img_array.shape) == 2:  # Grayscale image
        # Perform SVD on the image array
        U, S, Vt = np.linalg.svd(img_array, full_matrices=False)

        # Compress the image by keeping only the top singular values
        k = int(compression_factor * min(img_array.shape))
        S_compressed = np.diag(S[:k])
        U_compressed = U[:, :k]
        Vt_compressed = Vt[:k, :]

        # Reconstruct the compressed image
        img_compressed = np.dot(U_compressed, np.dot(S_compressed, Vt_compressed))
    else:  # Color image
        # Perform SVD on each channel separately
        img_compressed = np.zeros_like(img_array)
        for i in range(img_array.shape[2]):  # Iterate over each channel
```

```
            channel = img_array[:, :, i]
            U, S, Vt = np.linalg.svd(channel, full_matrices=False)

            # Compress the channel by keeping only the top singular values
            k = int(compression_factor * min(channel.shape))
            S_compressed = np.diag(S[:k])
            U_compressed = U[:, :k]
            Vt_compressed = Vt[:k, :]

            # Reconstruct the compressed channel
            img_compressed[:, :, i] = np.dot(U_compressed, np.dot(S_compressed, Vt_compressed))

    # Clip the values to be between 0 and 255, and convert back to uint8
    img_compressed = np.clip(img_compressed, 0, 255).astype(np.uint8)

    # Generate the output path by adding '_compressed' to the original filename
    file_name, file_extension = os.path.splitext(image_path)
    output_path = f"{file_name}_compressed{file_extension}"

    # Save the compressed image
    compressed_img = Image.fromarray(img_compressed)
    compressed_img.save(output_path)

    return output_path

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Compress an image using SVD.")
    parser.add_argument("input_file", help="Path to the input image file")
    parser.add_argument("--compression_factor", type=float, default=0.1, help="Compression factor (default: 0
    args = parser.parse_args()

    output_file = compress_image(args.input_file, args.compression_factor)
    print(f"Compressed image saved as: {output_file}")
```