इमेज कम्प्रेशन लिनियर एल्जेब्रा का उपयोग करके

चित्र संपीड़न डिजिटल चित्र प्रोसेसिंग का एक मूलभूत कार्य है, जिसका उद्देश्य चित्रों की संग्रहण आकार को कम करने के साथ-साथ उनकी दृश्य गुणवत्ता को बनाए रखना है। इस लक्ष्य को प्राप्त करने के लिए एक शक्तिशाली विधि है लीनियर एल्जेब्रा का उपयोग, विशेष रूप से सिंग्युलर वैल्यू डिकम्पोजिशन (□□□)। यह तकनीक हमें एक चित्र मैट्रिक्स को एक अधिक संक्षिप्त रूप में प्रस्तुत करने की अनुमति देता है, प्रभावी रूप से कम महत्वपूर्ण जानकारी को छोड़ते हुए, जबकि मूलभूत विशेषताओं को बनाए रखते हुए।

निम्नलिखित पाइथन कोड दिखाता है कि □□□ का उपयोग करके एक चित्र को कैसे संपीड़ित किया जा सकता है। इस प्रक्रिया में चित्र को उसके घटकों में विभाजित करना शामिल है, इन घटकों को केवल सबसे महत्वपूर्ण विशेषताओं को बनाए रखते हुए संपीड़ित करना, और फिर संपीड़ित चित्र को पुनर्निर्मित करना। यह विधि दोनों ग्रेस्केल और रंगीन चित्रों पर लागू की जा सकती है, एक लचीला और गणितीय रूप से सही तरीका प्रदान करता है चित्र आकार को कम करने के लिए।

```
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}")
```