NAME:	Anish Gade
UID:	2021700022
BRANCH:	T.Y. CSE Data Science
BATCH:	B
SUBJECT:	FOSIP
EXP. NO.:	10
DATE:	18-11-2023

AIM: - To study image compression

OBJECTIVES:

(I) Perform Image Compression Operation

(II) Download the Conference paper from IEEE Xplorer published after 2018

(III) Develop the algorithm and process the following images for different types of images

(IV) Give your conclusion based on results obtained.

INTRODUCTION:

Image compression is the process of reducing the size of an image file while retaining its visual quality. This reduction in size is achieved by removing redundant or irrelevant information from the image. Image compression is essential for efficient storage and transmission of images, especially in applications where bandwidth or storage space is limited.

There are two main types of image compression:

Lossless Compression:
Lossless compression preserves all the original data in the image. When the compressed image is decompressed, it is identical to the original. Common lossless compression formats include PNG and GIF. Lossless compression is suitable for situations where exact reproduction of the image is critical, such as in medical imaging or text documents.

Lossy Compression:
Lossy compression achieves higher compression ratios by discarding some image information that is considered less perceptually important. The decompressed image may not be identical to the original, but the loss of quality is often imperceptible to the human eye. JPEG is a popular lossy compression format widely used for photographs on the web.

ALGORITHM (USING OPEN CV):

In the provided code example, we used OpenCV, a powerful computer vision library, to perform image compression using the Discrete Cosine Transform (DCT). The DCT is a mathematical transformation commonly used in JPEG compression.

Here's a breakdown of the steps in the OpenCV-based image compression code:

- **Read the Image:**
  The original image is read using cv2.imread from the specified file path.

- **Convert to YCbCr Color Space:**
  The image is converted from the default BGR color space to YCbCr. YCbCr separates the image into luminance (Y) and chrominance (Cb and Cr) components.

- **Split Channels and Apply DCT:**
  The YCbCr channels are separated, and the Discrete Cosine Transform (DCT) is applied independently to each channel. DCT transforms the image data from the spatial domain to the frequency domain.

- **Scale DCT Coefficients:**
  The DCT coefficients are scaled based on the specified quality factor. This step determines the level of compression applied.
- **Apply Inverse DCT:**
  The inverse DCT is applied to the scaled coefficients to reconstruct the image in the spatial domain.
- **Merge Channels and Convert Color Space:**
  The processed YCbCr channels are merged back to obtain the compressed image. The image is then converted back to the BGR color space for display.
- **Display Original and Compressed Images:**
  The original and compressed images are displayed side by side using Matplotlib.
  Size Comparison:

The sizes of the original and compressed images are compared and visualized in a bar chart. This step helps assess the effectiveness of the compression.

EXPERIMENTATION:

CODE:

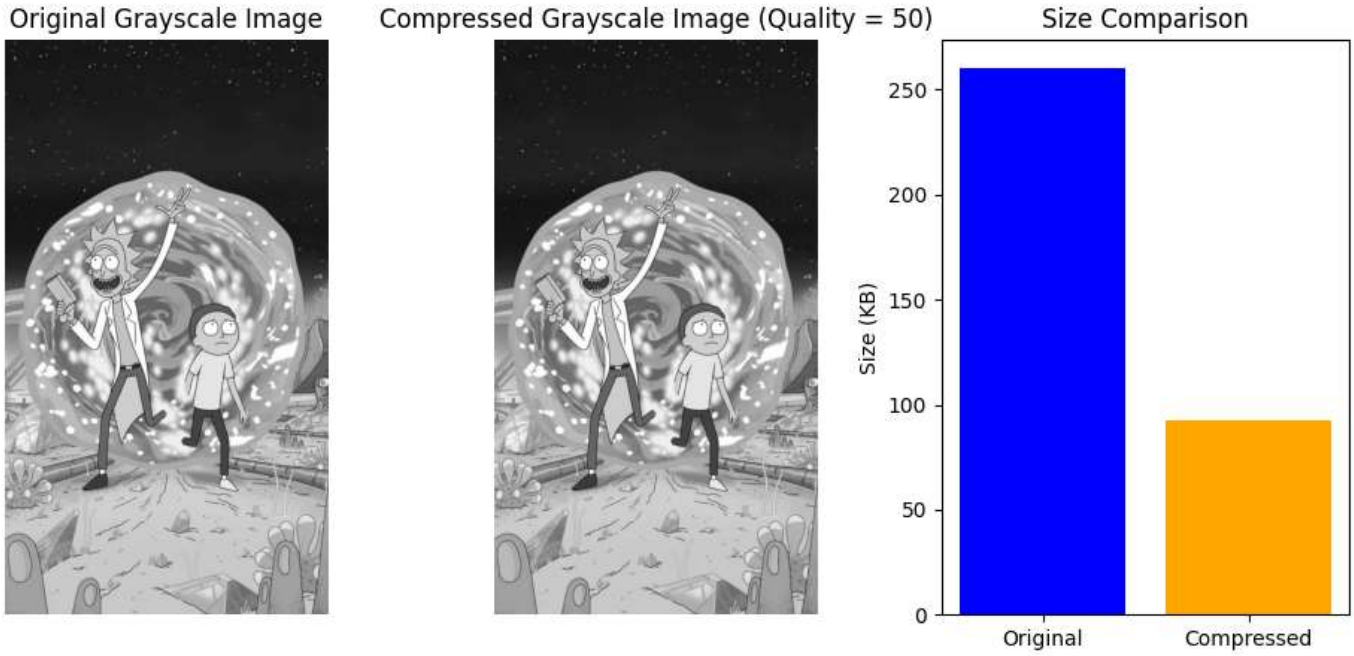
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_sample_image
import os
```
✓ 0.0s

```
def get_file_size(path):
    return os.path.getsize(path) / (1024.0)
```
✓ 0.0s

```
def compress_image(image_path, quality=50):
    # Read the grayscale image
    original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Check if the image is loaded successfully
    if original_image is None:
```

```
        print("Error: Image not loaded.")
        return

    # Get the size of the original image
    original_size = get_file_size(image_path)

    # Apply DCT to the grayscale image
    dct_image = cv2.dct(np.float32(original_image))

    # Set the quality factor for compression (0-100, higher means better quality)
    quality_factor = quality

    # Scale the DCT coefficients
    dct_image *= quality_factor / 100.0

    # Apply inverse DCT to the compressed image
    compressed_image = np.uint8(cv2.idct(dct_image))

    # Get the size of the original image
    original_size = get_file_size(image_path)

    # Save the compressed image with explicit compression settings
    compressed_path = f"compressed_grayscale_image_quality_{quality}.jpg"
    cv2.imwrite(compressed_path, compressed_image, [int(cv2.IMWRITE_JPEG_QUALITY),
quality])

    # Get the size of the compressed image
    compressed_size = get_file_size(compressed_path)

    # Display the original and compressed images
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 3, 1)
    plt.title("Original Grayscale Image")
    plt.imshow(original_image, cmap='gray')
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.title(f"Compressed Grayscale Image (Quality = {quality})")
    plt.imshow(compressed_image, cmap='gray')
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.title("Size Comparison")
    plt.bar(['Original', 'Compressed'], [original_size, compressed_size], color=['blue',
'orange'])
    plt.ylabel("Size (KB)")

    plt.show()
```

RESULT:



CONCLUSION:

I learned the implementation and application of image compression using python open cv library

REFERENCES:

1. https://docs.opencv.org/3.4/d2/de8/group__core__array.html