Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL603

COURSE NAME: Image Processing and Computer Vision Laboratory

CLASS: T Y B. TECH

Name: Anish Sharma **Roll no:I011**

EXPERIMENT NO. 8

CO/LO: Apply Image Compression Technique.

AIM / OBJECTIVE:

Apply image compression technique and analyse compressed data using Compression Ratio and MSE.

EXERCISE

Apply the following data compression techniques. Calculate the compression ratio and MSE as applicable.

1. Lossless Compression: To compress data using LZW coding.
2. Lossy Compression: To compression data using Transform Coding (JPEG)

Steps to perform LZW coding:

1. Input: text: A sequence of characters in the ASCII character set.
2. Output: A sequence of indices into a dictionary.
3. For each character c in the ASCII character set:
    a. Insert c into the dictionary at the index equal to c's numeric code in ASCII.
4. Set s to the first character from text.
5. While text is not exhausted, do the following:
    a. Take the next character from text, and assign it to c.
    b. If s c is in the dictionary, then set s to s c.
    c. Otherwise (s c is not yet in the dictionary), do the following:

i. Output the index of s in the dictionary. ii. Insert s c into the next available entry in the dictionary. iii. Set s to the single-character string c.

6. Output the index of s in the dictionary. CODE:

```
import numpy as np

def lzw_compress(text):
    # Initialize the dictionary with ASCII characters      dictionary
= {chr(i): i for i in range(256)}      next_code
= 256  # Next available index in dictionary

    s = text[0]  # First character
compressed = []

    for c in text[1:]:
if s + c in dictionary:
s += c  # Extend s          else:
compressed.append(dictionary[s])  # Output index of s
dictionary[s + c] = next_code  # Insert s+c into dictionary
next_code += 1              s = c  # Reset s to c

    compressed.append(dictionary[s])  # Output last index
return compressed

def compression_ratio(original, compressed):       original_size = len(original) * 8  #
Assuming 8-bit ASCII characters      compressed_size = len(compressed) *
np.ceil(np.log2(max(compressed) + 1))      return original_size / compressed_size

def mse(original, decompressed):
```

original_values = np.array([ord(c) for c in original], dtype=np.float64) decompressed_values = np.array([ord(c) for c in decompressed], dtype=np.float64) return np.mean((original_values - decompressed_values) ** 2)

# Example usage
text = "TOBEORNOTTOBEORTOBEORNOT"
compressed_output = lzw_compress(text) print("Compressed Output:", compressed_output) print("Compression Ratio:", compression_ratio(text, compressed_output))

OUTPUT:

```
Compressed Output: [84, 79, 66, 69, 79, 82, 78, 79, 84, 256, 258, 260, 265, 259, 261, 263]
Compression Ratio: 1.3333333333333333
```

Steps to perform JPEG coding:

1. The image is divided into 8x8 non-overlapping blocks.
2. Each block is level-shifted by subtracting 128 from it.
3. Each (level-shifted) block is transformed with Discrete Cosine Transform (DCT).

4. Each block (of DCT coefficients) is quantized using a quantization table. The quantization table is modified by the “quality” factor that controls the quality of compression.
5. Each block (of quantized DCT coefficients) is reordered in accordance with a zigzag pattern.
6. In each block (of quantized DCT coefficients in zigzag order) all trailing zeros (starting immediately after the last non-zero coefficient and up to the end of the block) are discarded and a special End-Of-Block (EOB) symbol is inserted instead to represent them.

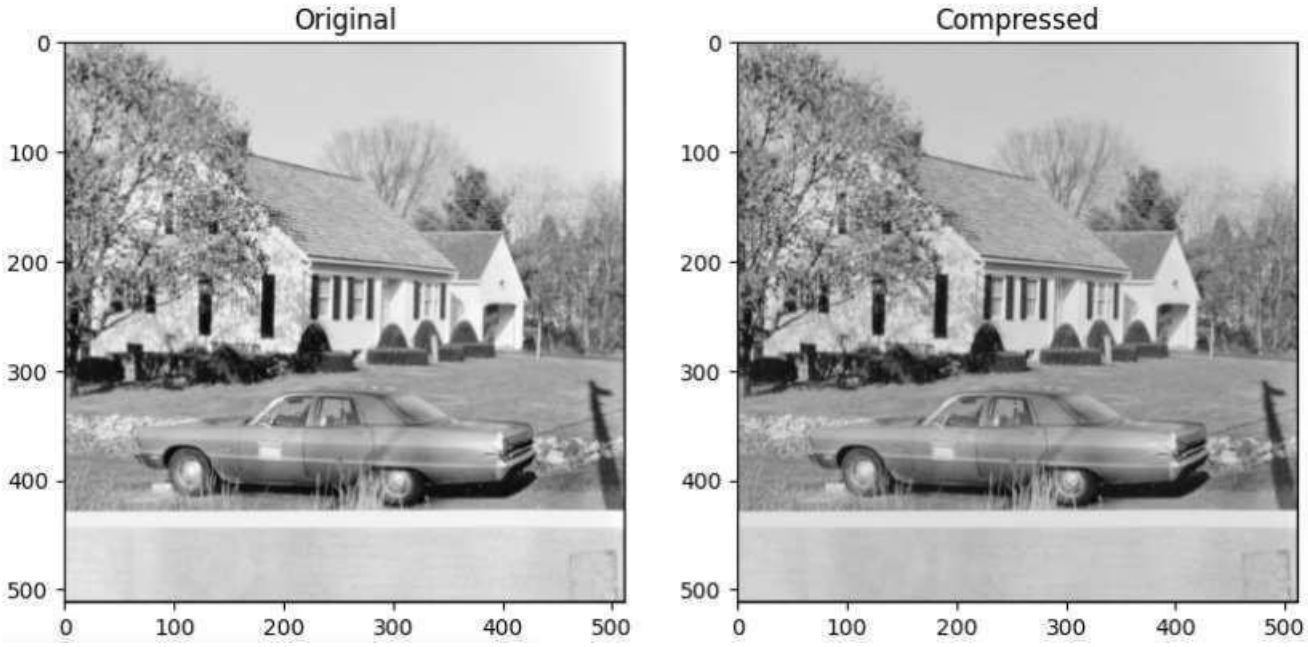
CODE:

```
import cv2 import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct

def apply_dct(block):      return dct(dct(block.T,
norm='ortho').T, norm='ortho')  def apply_idct(block):
return idct(idct(block.T, norm='ortho').T, norm='ortho')

def quantize(block, q_table):
return np.round(block / q_table)

def dequantize(block, q_table):
return block * q_table

# Load image gray = cv2.imread("house (1).tiff",
cv2.IMREAD_GRAYSCALE)

# Resize to be a multiple of
8x8 h, w = gray.shape h = h -
(h % 8) w = w - (w % 8) gray =
gray[:h, :w]

# Define JPEG quantization table (example for quality factor 50) q_table
= np.array([
    [16, 11, 10, 16, 24, 40, 51, 61],
    [12, 12, 14, 19, 26, 58, 60, 55],
    [14, 13, 16, 24, 40, 57, 69, 56],
```

```
    [14, 17, 22, 29, 51, 87, 80, 62],
    [18, 22, 37, 56, 68, 109, 103, 77],
    [24, 35, 55, 64, 81, 104, 113, 92],
    [49, 64, 78, 87, 103, 121, 120, 101],
    [72, 92, 95, 98, 112, 100, 103, 99]
], dtype=np.float32)

# Process 8x8 blocks compressed = np.zeros_like(gray,
dtype=np.float32) for i in range(0, h, 8):     for j in range(0, w, 8):
block = gray[i:i+8, j:j+8].astype(np.float32) - 128  # Level shift
dct_block = apply_dct(block)         q_block = quantize(dct_block,
q_table)         dq_block = dequantize(q_block, q_table)
idct_block = apply_idct(dq_block) + 128  # Reverse level shift
compressed[i:i+8, j:j+8] = idct_block

# Convert to uint8 for visualization compressed =
np.clip(compressed, 0, 255).astype(np.uint8)

# Show original and compressed
images plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1) plt.title("Original")
plt.imshow(gray, cmap='gray')
plt.subplot(1, 2, 2)
plt.title("Compressed")
plt.imshow(compressed, cmap='gray')
plt.show()
```

Save the compressed image

cv2.imwrite("compressed.jpg", compressed) OUTPUT:



Conclusion: The LZW compression algorithm efficiently reduces data size by replacing repeated sequences with dictionary indices. The compression ratio measures effectiveness, showing how much storage is saved, while the mean squared error (MSE) evaluates potential data loss in lossy scenarios. This approach is useful for text and image compression where redundancy is high.