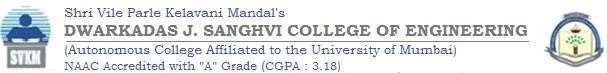
Academic Year 2024-25 Sap id:60003220045



**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:
   1. 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:
   1. Take the next character from text, and assign it to c.
   2. If s c is in the dictionary, then set s to s c.
   3. 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.

1. 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))

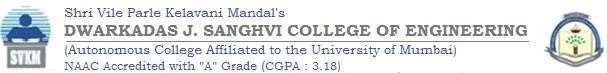
OUTPUT:



**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).

Academic Year 2024-25



1. 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.
2. Each block (of quantized DCT coefficients) is reordered in accordance with a zigzag pattern.
3. 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)

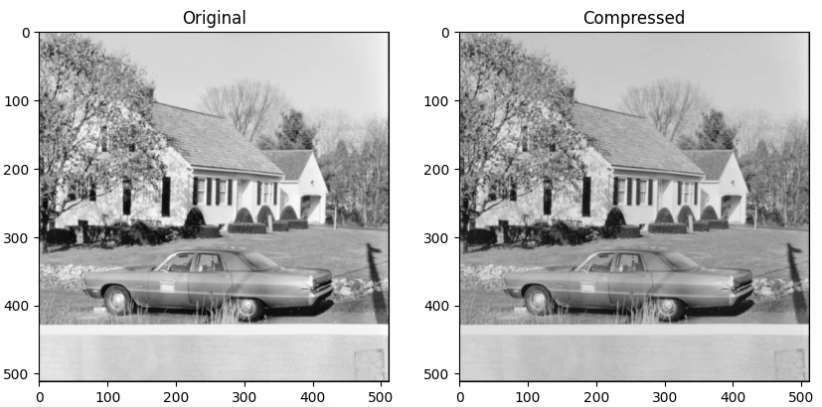
# Process 8x8 blocks compressed = np.zeros\_like(gray, dtype=np.float32) for i in range(0, h, 8): for j in range(0, w, 8): block = gray[i:i+8, j:j+8].astype(np.float32) - 128 # Level shift dct\_block = apply\_dct(block) q\_block = quantize(dct\_block, q\_table) dq\_block = dequantize(q\_block, q\_table) 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.