

**DIGITAL IMAGE PROCESSING MINI PROJECT**

**Name:R.E Gowtam**

**RRN:231541601008**

**Department:Bachelor of Computer Application**

**Specilization : Artificial Intelligence**

**🎯 Project Title:**

**Image Compression using Discrete Cosine Transform (DCT)**

**📝 Objective:**

The primary goal of this mini project is to **implement a simplified JPEG-like image compression** technique using the **Discrete Cosine Transform (DCT)**. The project demonstrates how an image can be compressed by reducing its frequency domain information and later reconstructed (decompressed) while maintaining acceptable quality. Additionally, the project helps visualize the effects of compression on image quality using metrics like **PSNR (Peak Signal-to-Noise Ratio)**.

**🧠 Concepts Covered:**

**1. Discrete Cosine Transform (DCT):**

* Converts spatial image data into frequency domain.
* Concentrates most of the signal information in low frequencies.
* Helps in discarding high-frequency components for compression.

**2. Quantization:**

* Reduces precision of DCT coefficients to achieve compression.
* Uses a standard JPEG quantization matrix.
* High-frequency values are reduced more aggressively than low-frequency ones.

**3. Inverse DCT (IDCT):**

* Reconstructs the image from compressed (quantized) DCT coefficients.
* Ensures that the image is visually similar to the original.

**4. Block-based Processing:**

* Image is divided into 8x8 blocks (as in JPEG).
* DCT, quantization, and IDCT are applied block-wise.

**5. PSNR (Peak Signal-to-Noise Ratio):**

* A common metric to measure quality loss.
* Higher PSNR indicates better image quality after compression.

**🛠️ Tools & Technologies Used:**

* **Python**
* **OpenCV** – for image processing
* **NumPy** – for numerical operations
* **Matplotlib** – for visualization
* **scikit-image** – for PSNR metric
* **Google Colab** – for easy testing and sharing

**🔁 Workflow / Pipeline:**

1. **Image Upload** (via Google Colab)
2. **Preprocessing**:
   * Convert to grayscale
   * Resize image to be a multiple of 8
3. **DCT Compression**:
   * Apply DCT block-wise
   * Quantize the coefficients
4. **Decompression**:
   * Dequantize coefficients
   * Apply IDCT block-wise
5. **Evaluation & Visualization**:
   * Display original and compressed images side-by-side
   * Show PSNR score to measure compression quality

**✅ Strengths of the Project:**

* **Educational**: Clearly demonstrates how DCT-based compression works under the hood.
* **Visual**: Shows how compression affects image quality.
* **Practical**: Based on the same principles used in JPEG compression.
* **Portable**: Easy to run and test using Google Colab.
* **Clean Code**: Well-structured, modular code with reusable functions

**📈 Future Enhancements (Ideas):**

1. **Interactive slider** to adjust compression level in real-time.
2. **Save compressed image** to disk and compare file sizes.
3. **Batch compression** for multiple images.
4. **Graphical analysis** – Plot PSNR vs quality level.
5. **Mobile/web app** version using Flask + OpenCV.js for demonstrations.

**📊 Evaluation Metrics:**

| **Metric** | **Result (Example)** |
| --- | --- |
| PSNR | ~30–40 dB |
| Visual Quality | Slightly blurred edges |
| Compression | Blocks of high similarity simplified |

**🎓 Skills Demonstrated:**

* Digital Image Processing
* Frequency domain analysis
* Matrix manipulation & quantization
* Performance measurement with PSNR
* Modular Python programming
* Google Colab usage for demos

**Programs:**

# 📦 Install required packages

!pip install opencv-python-headless matplotlib scikit-image

# 📁 Upload image

from google.colab import files

uploaded = files.upload()

# ✅ Required imports

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.metrics import peak\_signal\_noise\_ratio as psnr

from PIL import Image

from io import BytesIO

# 🖼️ Load the uploaded image

for fn in uploaded.keys():

    img\_path = fn

# Constants

BLOCK\_SIZE = 8

# JPEG quantization matrix (luminance)

QUANT\_MATRIX = 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]

])

# 📋 Helper functions

def preprocess\_image(img\_path):

    img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

    h, w = img.shape

    h -= h % BLOCK\_SIZE

    w -= w % BLOCK\_SIZE

    return cv2.resize(img, (w, h))

def dct2(block):

    return cv2.dct(block.astype(np.float32))

def idct2(block):

    return cv2.idct(block)

def compress\_image(img, quant\_matrix):

    h, w = img.shape

    dct\_coeffs = np.zeros((h, w), dtype=np.float32)

    for i in range(0, h, BLOCK\_SIZE):

        for j in range(0, w, BLOCK\_SIZE):

            block = img[i:i+BLOCK\_SIZE, j:j+BLOCK\_SIZE]

            dct\_block = dct2(block)

            quant\_block = np.round(dct\_block / quant\_matrix)

            dct\_coeffs[i:i+BLOCK\_SIZE, j:j+BLOCK\_SIZE] = quant\_block

    return dct\_coeffs

def decompress\_image(quantized, quant\_matrix):

    h, w = quantized.shape

    reconstructed = np.zeros((h, w), dtype=np.float32)

    for i in range(0, h, BLOCK\_SIZE):

        for j in range(0, w, BLOCK\_SIZE):

            block = quantized[i:i+BLOCK\_SIZE, j:j+BLOCK\_SIZE]

            dequant\_block = block \* quant\_matrix

            idct\_block = idct2(dequant\_block)

            reconstructed[i:i+BLOCK\_SIZE, j:j+BLOCK\_SIZE] = idct\_block

    return np.clip(reconstructed, 0, 255).astype(np.uint8)

def visualize\_results(original, compressed):

    psnr\_value = psnr(original, compressed)

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)

    plt.title("Original")

    plt.imshow(original, cmap='gray')

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.title(f"Compressed\nPSNR: {psnr\_value:.2f} dB")

    plt.imshow(compressed, cmap='gray')

    plt.axis('off')

    plt.tight\_layout()

    plt.show()

# 🚀 Main execution

print("🔄 Preprocessing image...")

original = preprocess\_image(img\_path)

print("⚙️ Compressing image using DCT...")

compressed\_dct = compress\_image(original, QUANT\_MATRIX)

print("♻️ Decompressing using IDCT...")

decompressed = decompress\_image(compressed\_dct, QUANT\_MATRIX)

print("📊 Visualizing results...")

visualize\_results(original, decompressed)

**OUTPUT:**

****