Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur

DIGITAL IMAGE AND VIDEO PROCESSING LAB (EC69211)

MINI PROJECT

Title: JPEG Compression

Date of submission: 08.11.2024

Submitted By:
Unnati Singh, 21EC39027
Sneha Dash, 21EC39023

Instructors:
Prof. Saumik Bhattacharya
Prof. Prabir Kumar Biswas

Introduction:

JPEG (Joint Photographic Experts Group) is a widely used method for compressing photographic images, especially for digital images. Developed in the late 1980s, JPEG became the standard for compressing images with complex colour gradients and details, like photographs, where preserving visual quality while reducing file size is crucial.

JPEG uses **lossy compression**, which means some data is discarded to reduce file size. The human eye is more sensitive to brightness (luminance) than to colour (chrominance), so JPEG exploits this by compressing colour information more than brightness, minimising the loss of perceptible image quality. JPEG compression can reduce image file sizes by up to 90%, making it ideal for storing and sharing images efficiently.

Necessity:

Digital images, especially high-resolution photos, consume substantial storage space. Large image sizes can lead to issues with storage limitations, slower data transfer rates, and increased bandwidth usage. JPEG compression helps by significantly reducing file sizes, making it easier to store, share, and display images on various devices without occupying excessive storage or requiring high bandwidth.

In this mini project, we implement the JPEG Compression Pipeline to see how JPEG compression reduces file size, its trade-offs, and the implications of the same on image quality.

JPEG Compression Pipeline:

JPEG compression is a widely-used lossy compression technique for digital images, especially for photographic images. The pipeline includes:

1. Colour Space Conversion
2. Downsampling
3. Block Formation
4. Discrete Cosine Transform (DCT)
5. Quantization
6. Run Length Encoding
7. Huffman Encoding

1. Colour Space Conversion

In JPEG, images are often converted from the RGB colour space to the YCbCr colour space. This is because the human eye is more sensitive to brightness (luminance, Y) than colour (chrominance, Cb and Cr), allowing chrominance to be compressed more heavily without significantly affecting perceived quality. The following transformation has been used for colour space conversion:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.81 & -0.81 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix},$$

2. Sub-sampling:

JPEG compression typically reduces the resolution of the Cb and Cr channels since human vision is less sensitive to colour detail than brightness. This downsampling is commonly done in a 4:2:0 ratio, meaning that for every 4x4 block of Y samples, there is a 2x2 block for Cb and Cr, which halves the chrominance resolution both horizontally and vertically.

3. Block Formation:

JPEG divides each channel (Y, Cb, Cr) into 8x8 pixel blocks. This segmentation is necessary for the subsequent Discrete Cosine Transform (DCT) step, as DCT is applied on a block-by-block basis. The code handles this by splitting each colour channel into non-overlapping 8x8 blocks.

4. DCT:

It transforms each 8x8 block from the spatial domain to the frequency domain. Low-frequency components, which are more visually significant, are concentrated in the upper-left corner of the DCT matrix, while high-frequency components are spread toward the lower-right.
The DCT formula for an 8x8 block:

$$F(u,v) = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right)$$

The code applies DCT on each 8x8 block of Y, Cb, and Cr components.

5. Quantization:

Quantization is the primary source of loss in JPEG compression. The DCT coefficients are divided by a quantization matrix and then rounded to reduce precision, eliminating less visually important information (mostly high-frequency components).
Quantization matrix used:

```
quant_matrix_y = 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]
]) * (100 - quality) / 50
```

6. Run Length Encoding:

After the Discrete Cosine Transform (DCT) and quantization steps, each 8x8 block of DCT coefficients has mostly non-zero values concentrated in the top-left corner (low frequencies), while the remaining coefficients (high frequencies) tend to be closer to zero. Zigzag ordering is a specific sequence for traversing these coefficients in a diagonal path from the top-left to the bottom-right corner of the block.
The purpose of zigzag ordering is to:

- Place the most significant (non-zero) coefficients first.
- Group the trailing zero coefficients towards the end of the sequence.
  Zigzag Ordering used:

```
zigzag_order = [
    (0, 0), (0, 1), (1, 0), (2, 0), (1, 1), (0, 2), (0, 3), (1, 2), (2, 1), (3, 0),
    (3, 1), (2, 2), (1, 3), (0, 4), (0, 5), (1, 4), (2, 3), (3, 2), (4, 0), (5, 0),
    (4, 1), (3, 3), (2, 4), (1, 5), (0, 6), (0, 7), (1, 6), (2, 5), (3, 4), (4, 3),
    (5, 1), (6, 0), (7, 0), (6, 1), (5, 2), (4, 4), (3, 5), (2, 6), (1, 7), (0, 7),
    (1, 6), (2, 5), (3, 4), (4, 3), (5, 1), (6, 0), (7, 0), (6, 1), (5, 2), (4, 4),
    (3, 5), (2, 6), (1, 7), (0, 7)
]
```

After zigzag ordering, many of the quantized DCT coefficients at the end of the sequence will be zero. **Run-Length Encoding (RLE)** is used to compress these repeated zero values. This encoding represents consecutive zeros as a single value indicating how many zeros are present, which reduces storage space.

7. Huffman Encoding:

It is a variable-length encoding technique that assigns shorter codes to frequently occurring symbols and longer codes to less frequent ones, based on their frequency of occurrence. In JPEG compression, Huffman coding is applied after RLE to further compress the data.

The process works by constructing a Huffman tree where:

- Leaf nodes represent symbols (RLE pairs) with their frequencies.
- Each path from the root to a leaf represents a binary code for that symbol, with shorter paths for more frequent symbols.

Results:

We tested our pipeline on several images, saving the Huffman encoded binary file for each image.

The results can be seen as:

```
Enter image path: wall.png
Compressed file saved at: output/wall_bin.pkl.gz
Original Size: 0.12 MB
Compressed Size: 0.05 MB
```

```
Y : [6, 7, 11, 9, 10, 9, 6, 10, 9, 9, 7, 8, 7, 4, 2, 5, 6, 6, 7, 5, 5, 5,
Cb : [7, 10, 14, 12, 13, 12, 8, 12, 12, 12, 9, 11, 9, 5, 3, 7, 7, 8, 9, 7
Cr : [4, 6, 8, 7, 7, 7, 4, 7, 7, 7, 5, 6, 5, 3, 2, 4, 4, 5, 5, 4, 4, 3, 3
huffman_Y : 1101111010100101010010011101001101110011101001010011010101101
huffman_Cb : 1011001100011100010001111001010011001000100010000110000001106
huffman_Cr : 0010100000000100010001001000100010001011010011110111001001010
```

Results for image 1

```
Enter image path: im1.png
Compressed file saved at: output/im1_bin.pkl.gz
Original Size: 2.66 MB
Compressed Size: 1.58 MB
```

```
Y : [5, 6, 9, 8, 8, 8, 5, 8, 7, 8, 6, 7, 6, 3, 2, 4, 5, 5, 6, 4, 4, 4, 3,
Cb : [6, 7, 11, 9, 10, 10, 6, 10, 9, 9, 7, 8, 7, 4, 3, 5, 6, 6, 7, 5, 5,
Cr : [5, 7, 10, 9, 9, 9, 6, 9, 9, 9, 7, 8, 7, 4, 2, 5, 5, 6, 7, 5, 5, 4,
huffman_Y : 1010100101010100111001101001010101001110101001001010101001011
huffman_Cb : 1100100110101011000110111101111001101110001000100110100100111
huffman_Cr : 1010100100110000000001011000000000100110001100100101111010101
```

Results for image 2

```
Enter image path: blob.png
Compressed file saved at: output/blob_bin.pkl.gz
Original Size: 0.04 MB
Compressed Size: 0.02 MB
```

```
Y : [(5, 1), (1, 1), (2, 1), 1, (2, 1), (2, 1), (2, 1), (3, 1), (4, 1)
Cb : [6, 8, 12, 10, 11, 11, 7, 11, 10, 10, 8, 9, 8, 5, 3, 6, 6, 7, 8,
Cr : [6, 8, 12, 10, 11, 11, 7, 11, 10, 10, 8, 9, 8, 5, 3, 6, 6, 7, 8,
huffman_Y : 000111111111110100000000000011111111110100011111111111010
huffman_Cb : 000101010001000110010001010110001000110011101010001110101
huffman_Cr : 000101010001100110010001010110001000110011101010000101010
```

Results for image 3

```
Enter image path: green.png
Compressed file saved at: output/green_bin.pkl.gz
Original Size: 1.94 MB
Compressed Size: 0.24 MB
```

```
Y : [7, 9, 12, 11, 11, 11, 7, 11, 10, 11, 8, 10, 8, 5, 3,
Cb : [3, 3, 5, 4, 4, 4, 3, 4, 4, 4, 3, 4, 3, 2, 1, 2, 3,
Cr : [5, 6, 9, 8, 8, 8, 5, 8, 7, 8, 6, 7, 6, 3, 2, 4, 5,
huffman_Y : 010101111101111010100101001010010101010010100
huffman_Cb : 001001000011011011001011011011001011001101111
huffman_Cr : 111010111101101100010001000111010001001100001
```

Results for image 4

The size of a pickle file is an indicator of the compression; however, it is not an accurate measure. In some cases, it can even be larger than the image being compressed.

This is due to several reasons. First, pickle has serialization overhead, as it stores not only the data but also metadata about the objects, which adds extra size. Additionally, pickle does not apply compression by default, so storing uncompressed image data, like a NumPy array, can take up more space than the image's compressed format (e.g., PNG or JPEG). Furthermore, pickle stores raw pixel data less efficiently compared to specialized image formats that use compression techniques. Redundant information, such as image metadata or additional attributes, can also contribute to the increased file size.

Discussion:

1. Advantages of JPEG:
    - JPEG is universally supported across platforms, devices, and software, making it a go-to choice for images on the web.

- The quality of JPEG compression can be controlled by adjusting the compression factor, allowing for a balance between file size and image quality.
- JPEG is optimised for quick decoding, which is useful for display in web browsers and image viewers.

2. Disadvantages of JPEG:
   - As a lossy format, JPEG discards some image data during compression, which can lead to blocking or blurring at lower quality settings.
   - JPEG performs poorly on images with sharp edges, text, or solid colours (like graphics or diagrams), where lossless compression methods are preferred.
   - Repeatedly saving a JPEG image causes further quality degradation, as each save involves compression.
3. Trade-offs in JPEG Compression:
   - **Quantization**: This step involves reducing the precision of the frequency components. The trade-off here is between compression ratio and image quality—higher quantization leads to greater compression but more visible artifacts.
   - **DCT**: DCT separates image data into frequency components, with most visual information concentrated in lower frequencies. The trade-off is between removing high-frequency information (which contributes to sharp edges) and reducing file size.
4. RLE is often used in JPEG for compressing the DC coefficients (i.e., the low-frequency components after DCT), where there tend to be many consecutive zeros, particularly when the image has smooth regions. Huffman Encoding is used to encode the AC coefficients (i.e., the higher frequency components) in a more efficient way by using variable-length codes based on symbol frequency. While Huffman is optimal in certain cases, it is not always the most efficient, especially when there are only a few distinct symbols with highly uneven probabilities. Optimising entropy encoding by exploring alternative algorithms (such as arithmetic coding) could further improve compression efficiency and adapt better to different types of data.