Report on the JPEG Compression Pipeline
Image and Video Processing Laboratory

Jayadithya Sreekar V *Samineni Rahul*

21EC39050 *21EC39021*

---

1. Introduction

JPEG (Joint Photographic Experts Group) is one of the most widely used formats for image compression, specifically designed for photographs and complex images. The JPEG compression algorithm is essential for reducing image file sizes, making it possible to store and transmit images more efficiently without significantly compromising visual quality. This report aims to detail the implementation of the JPEG compression and decompression pipeline, explaining each component and the reasoning behind it.

2. Objective

The objective of this project is to implement and understand the complete JPEG compression and decompression algorithm. This includes:

- Transforming images into a format that retains most of their visual fidelity while occupying less storage space.
- Providing insights into the step-by-step transformation, encoding, and reconstruction processes.
- Understanding the significance of each stage in the pipeline, particularly in terms of balancing image quality and compression ratio.

3. Theory

These concepts not only underpin the JPEG algorithm but also explain why specific techniques are employed.

3.1 Color Space Conversion

Concept: Color space conversion is the process of changing an image from one color representation (e.g., RGB) to another (e.g., YCbCr). RGB (Red, Green, Blue) is commonly used for image representation, but for compression purposes, YCbCr is preferred.

Explanation:

- **RGB Color Space**: Represents images as a combination of red, green, and blue components. Each pixel's color is defined by the intensity of these three colors.

- **YCbCr Color Space**: Separates the image into luminance (Y) and chrominance (Cb and Cr). The **Y channel** represents brightness, while **Cb** and **Cr** channels represent color differences (blue-difference and red-difference, respectively).
- **Reason for Conversion**: Human vision is more sensitive to changes in brightness than to changes in color. This means that more data can be discarded from the chrominance channels without significantly impacting perceived image quality.

3.2 Discrete Cosine Transform (DCT)

Concept: The Discrete Cosine Transform (DCT) converts an image block from the spatial domain (pixel values) to the frequency domain, enabling the separation of an image into components of varying importance.

Explanation:

- **Purpose**: The DCT is used to transform each 8x8 block of an image into a sum of cosine functions oscillating at different frequencies. This helps to highlight which parts of the image are low-frequency (smooth changes, most visible to the human eye) and high-frequency (sharp changes, less perceptible).
- **How it Works**: The DCT computes a matrix where the top-left corner represents the DC component (average brightness), and the rest of the matrix holds the AC components (varying detail frequencies).
- **Energy Compaction**: DCT concentrates most of the signal's energy in the lower frequencies, making it easier to compress since high-frequency components often contain less information.

JPEG uses the DCT because it efficiently compacts image data with real numbers, unlike the Discrete Fourier Transform (DFT), which involves complex numbers.

3.3 Quantization

Concept: Quantization is the process of reducing the precision of DCT coefficients, making it possible to compress the data by discarding less important information.

Explanation:

- **How It Works**: The DCT coefficients are divided by a quantization matrix and rounded to the nearest integer. The quantization matrix determines how aggressively different frequency components are reduced.
- **Impact on Image Quality**: Low-frequency coefficients (top-left) are kept more accurate as they represent important visual details, while high-frequency coefficients (bottom-right) are quantized more aggressively since they contribute less to perceived image quality.
- **Luminance vs. Chrominance**: The luminance (Y) channel uses a different quantization table from the chrominance (Cb and Cr) channels, reflecting the fact that the human eye is more sensitive to luminance details.

Quantization is the *only lossy step* in the JPEG compression process, directly affecting the file size and quality. By adjusting the quantization matrix, the compression ratio can be controlled, leading to higher compression but lower image quality or vice versa.

3.4 Zigzag Scanning

Concept: Zigzag scanning is a method of reordering the 8x8 block of quantized coefficients into a 1D array that places lower frequency components at the beginning and higher frequency components towards the end.

Explanation:

- **Purpose**: This ordering helps group significant coefficients (typically at the start) and trailing zeros (typically at the end) together, which facilitates efficient run-length encoding in the next step.
- **Process**: The coefficients are scanned starting from the top-left (DC coefficient), moving diagonally in a zigzag manner until the bottom-right is reached.

The zigzag pattern is used because it follows the order in which frequency importance typically decreases, making subsequent run-length encoding more effective by grouping zeros together.

3.5 Run-Length Encoding (RLE)

Concept: Run-length encoding (RLE) is a form of data compression that represents consecutive identical values (such as zeros) with a count and a single value.

Explanation:

- **Application in JPEG**: After zigzag scanning, the coefficients often contain long sequences of zeros due to quantization. RLE compresses these sequences by replacing them with a pair (run-length, value), which significantly reduces the amount of data needed for storage.
- **AC Coefficients**: RLE is applied to AC coefficients (all coefficients after the DC coefficient), which often have many zeros following the quantization process.

RLE is highly effective for compressing images with lots of continuous or repeating data, such as in the case of the high-frequency zeros in JPEG compression.

Example:

Input Data:

```
[0, 0, 0, 0, 15, 15, 0, 0, 0, 7, 7, 7, 0, 0, 0, 0, 0, 0]
```

Step-by-Step RLE Process:

1. Scan the data from left to right.
2. Count the number of consecutive occurrences of each value.

Resulting RLE Output:

- 4 consecutive zeros → `(4, 0)`
- 2 consecutive fifteens → `(2, 15)`
- 3 consecutive zeros → `(3, 0)`
- 3 consecutive sevens → `(3, 7)`
- 6 consecutive zeros → `(6, 0)`

Encoded Output:

```
[(4, 0), (2, 15), (3, 0), (3, 7), (6, 0)]
```

3.6 Huffman Coding

Concept: Huffman coding is a lossless data compression algorithm that assigns shorter binary codes to more frequently occurring symbols and longer codes to less frequent symbols.

Explanation:

- **Why Huffman Coding?**: By using variable-length codes where more common symbols have shorter representations, the overall data size is reduced.
- **Huffman Tree**: A binary tree is constructed where each leaf node represents a symbol, and the path from the root to the leaf represents the binary code for that symbol. The length of the path is inversely proportional to the symbol's frequency.
- **Compression Step**: The DC and RLE-encoded AC coefficients are encoded using Huffman coding. Each symbol's frequency is calculated, and a Huffman tree is built to create the code-book. This code-book is then used to encode the data.

Huffman coding is optimal for data where symbol frequency varies, as it minimizes the average length of the representation. It is commonly used in many compression algorithms beyond JPEG, such as ZIP file compression.

3.7 Theory to Calculate Compression Ratio

The **compression ratio** is a measure used to quantify the reduction in size achieved by compressing a data file. It is calculated as the ratio of the original data size to the compressed data size. The compression ratio provides an understanding of how much storage space is saved by applying a compression algorithm.

Compression Ratio Formula

The compression ratio is defined by the formula:

$$\text{Compression Ratio} = \text{(Size of Original Data)} / \text{(Size of Compressed Data)}$$

Explanation of Terms

1. **Size of Original Data (in bytes)**: This is the size of the data before applying the compression algorithm. In the context of image compression, this could be the size of the input image in bytes.
2. **Size of Compressed Data (in bytes)**: This represents the size of the data after compression. For the JPEG compression process, it can be calculated by summing up the sizes of the encoded blocks (Y, Cb, and Cr channels) produced after quantization, zigzag reordering, run-length encoding (RLE), and Huffman encoding.

Steps to Calculate Compression Ratio

1. **Calculate the Original Data Size**:
    - Read the original file to determine its size in bytes. For example, this can be done using `os.path.getsize(file_path)` for image files.
2. **Calculate the Compressed Data Size**:
    - Count the total number of bits used to store the compressed data.
    - Convert the total bits to bytes by dividing by 8 (1 byte = 8 bits).
3. **Apply the Formula**:
    - Compute the compression ratio using the formula above by dividing the size of the original data by the size of the compressed data.

Interpretation of the Compression Ratio

- A **higher compression ratio** indicates greater compression, meaning a larger reduction in data size.
- A **compression ratio of 1** means there was no change in data size.
- The **inverse of the compression ratio** gives the relative size reduction (e.g., if the compression ratio is 4, the compressed data is 41 or 25% of the original size).

Example Calculation

Consider an original image with a size of 1 MB (1,024,000 bytes). If, after compression, the size of the compressed data is 256 KB (262,144 bytes), then:

$$\text{Compression Ratio} = 1{,}024{,}000 \text{ bytes} / 262{,}144 \text{ bytes} = 3.91$$

This indicates that the compressed data is approximately 3.91 times smaller than the original data.

Visual Summary

These concepts are combined in JPEG compression as follows:

- **Color space conversion** prepares the image for compression by separating luminance from color information.
- **DCT** transforms 8x8 blocks into a frequency domain, allowing energy compaction.
- **Quantization** simplifies DCT coefficients to make the data more compressible.
- **Zigzag scanning** rearranges coefficients for effective run-length encoding.
- **Run-length encoding** compresses consecutive zeros.
- **Huffman coding** provides efficient, variable-length encoding for the final compressed data.

4. Algorithm

The JPEG compression and decompression processes consist of several key stages:

Compression Algorithm:

1. **Color Space Conversion**:

    - Convert the input image from the RGB color space to the YCbCr color space.
2. **Image Padding**:
    - Pad the image so that its dimensions are a multiple of 8x8, ensuring consistent block processing.
3. **Block Splitting**:
    - Split the image into non-overlapping 8x8 blocks.
4. **DCT Transformation**:
    - Apply the 2D DCT to each 8x8 block to convert it from the spatial domain to the frequency domain.
5. **Quantization**:
    - Quantize the DCT coefficients using the standard luminance and chrominance quantization tables.
6. **Zigzag Scanning**:
    - Reorder the quantized coefficients into a 1D array following a zigzag pattern.
7. **Run-Length Encoding**:
    - Encode the AC coefficients using run-length encoding.
8. **Huffman Coding**:
    - Compress the DC and run-length encoded AC coefficients using Huffman coding.

Decompression Algorithm:

1. **Huffman Decoding**:
    - Decode the compressed data using Huffman decoding.
2. **Run-Length Decoding**:
    - Reconstruct the AC coefficients using run-length decoding.
3. **Inverse Zigzag Scanning**:
    - Convert the 1D array of coefficients back to an 8x8 block.
4. **Dequantization**:
    - Multiply the coefficients by the quantization table used during compression.
5. **IDCT Transformation**:
    - Apply the inverse DCT to convert the data back to the spatial domain.
6. **Block Merging**:
    - Merge the processed blocks to reconstruct the full image.
7. **Color Space Conversion**:
    - Convert the image back from YCbCr to RGB.

5. Pseudo Code

Compression:

```
function JPEG_Compress(image):
    Convert image from RGB to YCbCr
    Pad image to be divisible by 8x8 blocks
    Split image into 8x8 blocks
    For each block:
        Apply DCT to the block
        Quantize the block using a quantization table
        Scan the block in a zigzag pattern
        Apply run-length encoding to the scanned data
        Compress the data using Huffman coding
    Return the compressed image data
```

Decompression:

```
function JPEG_Decompress(compressed_data):
    Decode the data using Huffman decoding
    For each block:
        Apply run-length decoding to reconstruct the coefficients
        Rearrange the coefficients using inverse zigzag scanning
        Dequantize the block
        Apply inverse DCT to the block
    Merge the blocks to form the complete image
    Convert the image from YCbCr to RGB
    Return the decompressed image
```

GitHub link of the code: *Link for the code* [https://github.com/sreekarvjenr/JPEG-compression/tree/master]

6. **Results**

The results of implementing the JPEG compression pipeline are as follows:

- **Compression Efficiency**: The final file size is reduced significantly, with minimal loss of visual quality.
- **Visual Fidelity**: The decompressed image retains most of the original details, with some slight artifacts due to the quantization process.
- **Performance**: The algorithm demonstrates the trade-off between compression ratio and computational complexity.

IMAGE

cameraman.bmp – 256 by 256 grayscale image

Compression Ratio: 9.74

Compression time: 26.81 sec

Decompression time: 46.84 sec

garden.png – 101 by 180 color image

Compression Ratio: 18.78

Compression time: 12.08 sec

Decompression time: 13.49 sec

lena_512_color.bmp – 512 by 512 color image

Compression Ratio: 34.40

Compression time: 111.14 sec

Decompression time: 46.84 sec

7. Discussion

JPEG compression is highly effective for storing and transmitting photographic images due to its balance of quality and compression. The algorithm leverages human perception to prioritize data retention for brightness and low-frequency details while discarding less important high-frequency data. This report covered each step of the compression and decompression processes, focusing on their purpose, implementation, and the principles behind them.

The presented results show the performance of a JPEG compression algorithm applied to three different images: a 256x256 grayscale image (cameraman.bmp), a 101x180 color image (garden.png), and a 512x512 color image (lena_color_512.bmp). The results detail the compression ratio, the time taken for compression and decompression, and the efficiency achieved by the algorithm for each image.

General Observations

1. **Compression Ratio and Image Characteristics**:
    - Images with larger dimensions and higher detail levels tend to have higher compression ratios due to more data being subjected to lossy transformations and entropy coding. However, image type (grayscale vs. color) plays a key role in compression efficiency. The color images showed higher compression ratios compared to the grayscale image due to the separation and differential treatment of chrominance and luminance data.
2. **Compression and Decompression Times**:
    - The time taken for compression and decompression scales with the size and complexity of the image. Larger images require more processing time for block-based transformations, quantization, and entropy coding, while smaller images are faster to process. Decompression times are generally longer than compression due to inverse operations, including Huffman decoding, inverse quantization, and inverse DCT.
3. **JPEG Compression Efficiency**:
    - The results demonstrate that JPEG compression can achieve significant reductions in file size while maintaining an acceptable level of image quality. The balance between compression ratio and processing time highlights the trade-off inherent in using more complex transformations and encoding schemes.