Image Compression: Implementation & Analysis

Course Project Report - CS663

Harsh | Pranav | Swayam

November 24, 2024

Contents

Abstract

This report describes the implementation of an image compression engine inspired by the JPEG algorithm. The implemented system performs discrete cosine transform (DCT) on image patches, quantizes the coefficients, applies Huffman coding, and writes the compressed data to a file. A decoding module reconstructs the image for analysis. Results for multiple images are analyzed using metrics like RMSE and BPP across varying quality factors. The project's merits and limitations are also discussed.

1 Introduction

Image compression is a critical aspect of digital storage and transmission. The JPEG algorithm achieves significant compression by exploiting redundancies in image data while maintaining perceptual quality. This project implements the essential components of JPEG compression on grayscale images and evaluates its performance using RMSE and BPP metrics.

2 Algorithm Description

The implemented algorithm consists of the following steps:

1. **Discrete Cosine Transform (DCT):** The input image is divided into non-overlapping 8×8 blocks, and the 2D DCT is applied to each block to transform the image into the frequency domain.

```
% Function to perform block-wise 2D DCT
function dct_blocks = block_dct(image, block_size)
    [rows, cols] = size(image);
    dct_blocks = zeros(size(image));
    for i = 1:block_size:rows
        for j = 1:block_size:cols
            block = image(i:i+block_size-1, j:j+block_size-1);
            dct_blocks(i:i+block_size-1, j:j+block_size-1) = dct2(block);
        end
    end
end
```

2. **Quantization:** A quantization table is used to reduce the precision of the DCT coefficients. The quantization table values are scaled based on a user-defined quality factor.

```
% Function for quantization (applies block-by-block)
function quantized = quantize_dct(dct_coeffs, quality_factor)
    quant_table = [
        16 11 10 16 24 40 51 61;
        12 12 14 19 26 58 60 55;
        ...
    ];
    quant_table = quant_table * (100 / quality_factor);
    [rows, cols] = size(dct_coeffs);
    block_size = 8;
    quantized = zeros(size(dct_coeffs));
    for i = 1:block_size:rows
        for j = 1:block_size:cols
            block = dct_coeffs(i:i+block_size-1, j:j+block_size-1);
            quantized(i:i+block_size-1, j:j+block_size-1) = round(block
                ./ quant_table);
        end
    end
end
```

3. **Run-Length Encoding (RLE):** The quantized DCT coefficients are scanned in a zigzag order, and sequences of zero coefficients are encoded efficiently using run-length encoding.

```
% Add the run-length encoding function
function encoded = run_length_encode(data)
    encoded = [];
    count = 1;
    for i = 2:length(data)
        if data(i) == data(i-1)
            count = count + 1;
        else
            encoded = [encoded, data(i-1), count];
            count = 1;
        end
    end
    encoded = [encoded, data(end), count]; % Append last element
end
```

4. **Huffman Encoding:** The RLE output is further compressed using Huffman encoding, which replaces frequently occurring patterns with shorter codes.

5. **Decoding:** The compressed data is decoded by reversing the Huffman and RLE encoding processes, followed by inverse quantization and the Inverse Discrete Cosine Transform (IDCT).

```
% Function to perform inverse quantization and inverse DCT
function reconstructed = inverse_quantize_dct(quantized, quality_factor,
    img_size)
    quant_table = [
        16 11 10 16 24 40 51 61;
        12 12 14 19 26 58 60 55;
        ...
    ];
    quant_table = quant_table * (100 / quality_factor);
    reconstructed = zeros(img_size);
    for i = 1:8:img_size(1)
        for j = 1:8:img_size(2)
            block = quantized(i:i+7, j:j+7) .* quant_table;
            reconstructed(i:i+7, j:j+7) = idct2(block);
        end
    end
end
```

3 Dataset Description

The algorithm was tested on a dataset of grayscale BMP and TIF images, converted from color images if necessary. Each image had varying dimensions and complexity to evaluate the algorithm's robustness. The dataset included:

- SegPC-2021: Segmentation of Multiple Myeloma Plasma Cells in Microscopic Images.
- "Standard" test images (a set of images found frequently in the literature) from Image Processing Place.

The images were processed for five quality factors: 10, 20, 50, 75, and 90.

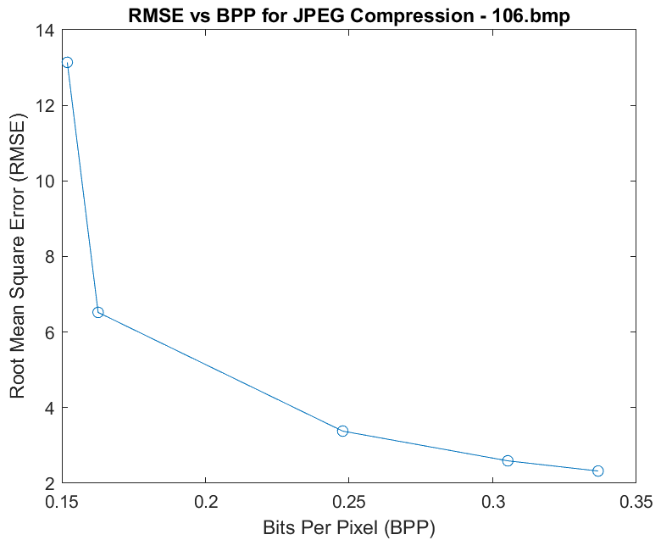

Figure 1: Instance of RMSE vs. Bits Per Pixel (BPP) for different quality factors.

4 Results and Analysis

This document contains a reference to the images used in this analysis. You can access the photos at the following link: **Base Image Compression - Google Drive**.
A comparison table of image sizes is provided in Table 1.

4.1 Discussion

The algorithm performs well in reducing file size while maintaining acceptable image quality. However:

- **Strengths:**
  - Efficient use of frequency domain for compression.
  - Effective reduction in redundancy using RLE and Huffman encoding.
  - Customizable quality factor allows flexible trade-offs.
- **Weaknesses:**
  - High computational overhead due to block-wise DCT and Huffman encoding.
  - Loss of detail in high-frequency regions at low quality factors.
  - Limited optimization for real-time applications.
  - Fixed quantization matrix limits adaptability.

Table 1: Comparison of Image Sizes

Name	Original image size	Compressed size	Recovered size
106	14.4 MB	1617 KB	4802 KB
108	14.4 MB	1635 KB	4802 KB
109	14.4 MB	1574 KB	4802 KB
111	14.4 MB	1574 KB	4802 KB
112	14.4 MB	1481 KB	4802 KB
114	14.4 MB	1570KB	4802 KB
1697	9.18 MB	984 KB	3062 KB
1698	9.18 MB	793 KB	3062 KB
1714	9.18 MB	977 KB	3062 KB
barbara_grey	258 KB	299 KB	258 KB
camera man	257 KB	174 KB	258 KB
house	513 KB	186 KB	514 KB
jetplane	513 KB	247 KB	514 KB
lake	513 KB	308 KB	514 KB
lena_grey_512	258 KB	63 KB	258 KB
lena_gray_256	65 KB	190 KB	66 KB
living room	257 KB	245 KB	258 KB
mandrill_gray	257 KB	327 KB	258 KB
peppers_grey	513 KB	229 KB	514 KB
pirate	257 KB	242 KB	258 KB
walkbridge	513 KB	389 KB	518 KB
woman_blonde	257 KB	211 KB	258 KB
woman_darkhair	257 KB	137 KB	258 KB

5 Introducing Color Images

5.1 Algorithm Extension

To extend the algorithm to color images, we can use the YCbCr color space. The Y channel contains the luminance information, while the Cb and Cr channels contain the chrominance information. The DCT is applied to each channel separately, and the quantization and encoding steps are performed independently.

```
% Convert the image to YCrCb color space
ycbcr_image = rgb2ycbcr(original_image);
Y_channel = double(ycbcr_image(:,:,1));
Cb_channel = double(ycbcr_image(:,:,2));
Cr_channel = double(ycbcr_image(:,:,3));
```

5.2 Results and Analysis

This document contains a reference to the images used in this analysis. You can access the photos at the following link: **Color Image Compression - Google Drive**.

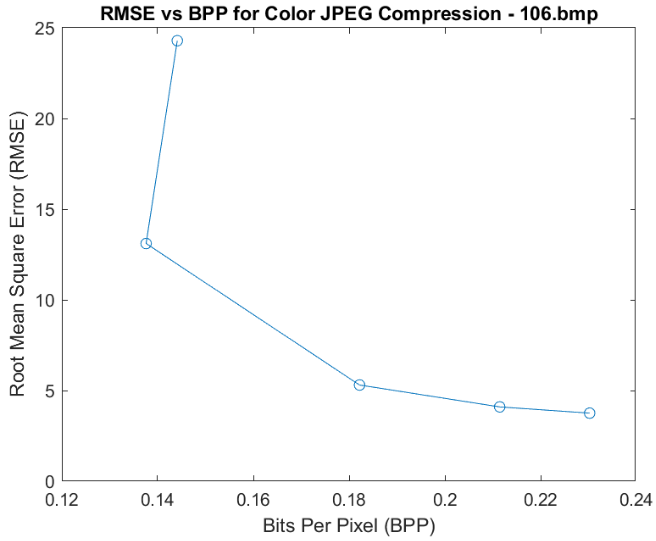

Figure 2: Instance of RMSE vs. Bits Per Pixel (BPP) for different quality factors.

6 Conclusion

The implemented JPEG-like compression algorithm successfully demonstrates the principles of lossy image compression. The RMSE and BPP results confirm the effectiveness of the approach in achieving a balance between image quality and compression ratio. The extension of the JPEG-like compression algorithm to color images demonstrates its versatility. Using the YCbCr color space enables efficient compression with minimal perceptual loss. However, the computational cost and susceptibility to artifacts in chrominance channels require further optimization. Future work could involve:

- Optimizing color space conversions for real-time applications.
- Using adaptive quantization for better quality preservation.
- Exploring more advanced entropy coding techniques like arithmetic coding.