A Low-Complexity Lossless Compression Method Based on a Code Table for Images

Rahul Gupta (202211069), AjayDeep Rawat (202211001)
- Under the guidance of Dr. Jignesh Patel

***Abstract*—This project implements a high-speed, lossless image compression algorithm from scratch, based on the method proposed by Yaohua Zhu et al. The approach targets high-bit-depth images, where conventional compression methods such as JPEG 2000 often fail to deliver real-time performance. The algorithm operates in three main stages: offline code table generation, real-time encoding, and decoding.**

In the offline phase, a universal Canonical Huffman code table is constructed using inter-column differencing and statistical frequency analysis of differential pixel values from a training dataset. This precomputed table removes the need for runtime entropy calculations, enabling highly efficient encoding, significantly reducing computational overhead, and supporting large-scale, real-time image processing.

The encoding phase compresses new images through direct table lookups and systematic handling of out-of-range differential values, while the decoding phase reconstructs the original images precisely by reversing the process using the stored first column and decoded differential data. The method ensures zero information loss, making it fully reliable for critical applications.

Experimental validation on a test dataset confirmed that the implementation achieves fully lossless compression, with a Mean Squared Error (MSE) of 0.0 between the original and reconstructed images. These results demonstrate that the implemented system successfully balances compression efficiency, speed, and accuracy, making it suitable for real-time high-bit-depth imaging applications in scientific, medical, and industrial domains.

Keywords— lossless compression, high-bit-depth images, Huffman coding, real-time encoding, image reconstruction, inter-column differencing.

I. Introduction

High-bit-depth images, such as 14-bit data from infrared sensors, scientific imaging devices, and medical or remote sensing instruments, contain a vast amount of pixel information. While this high precision allows for accurate representation of subtle variations in intensity, it also results in extremely large data sizes that pose significant challenges for storage, transmission, and real-time processing. Traditional compression methods, such as JPEG 2000 or standard lossless formats like PNG, can provide reasonable compression ratios; however, they often rely on computationally intensive operations, such as on-the-fly entropy coding or complex wavelet transforms, which limit their applicability in time-sensitive or real-time scenarios.

To address this problem, Zhu et al. [1] proposed a low-complexity, lossless compression method that replaces runtime entropy calculations with a precomputed, universal Canonical Huffman code table derived from a representative training dataset. This approach significantly reduces computational overhead while maintaining perfect image reconstruction. By combining inter-column differencing to remove spatial redundancy and systematic handling of out-of-bounds differential pixel values, the method ensures that high-bit-depth images can be compressed efficiently without any information loss.

This project implements the approach from scratch using Python, applying it to a high-bit-depth image dataset split into training and validation sets (approximately 53% training and 37% validation). The implementation focuses on high-speed encoding and decoding, lossless reconstruction, and practical applicability in real-time high-bit-depth imaging scenarios. The results demonstrate that this method achieves both computational efficiency and lossless performance, making it suitable for applications where large volumes of high-precision image data must be processed, stored, or transmitted rapidly.

II. Problem Statement

High-bit-depth images, such as those captured by infrared line scanners, medical imaging devices, or scientific imaging systems, contain a vast amount of pixel information, resulting in large data volumes that are challenging to store, transmit, and process efficiently. Existing lossless compression techniques, including JPEG-LS and JPEG 2000, often achieve good compression ratios but suffer from high computational complexity, memory overhead, and latency, making them unsuitable for real-time applications where speed, accuracy, and system responsiveness are critical.

The primary problem addressed in this work is the development of a fast, lossless compression algorithm capable of efficiently handling high-bit-depth images without compromising data integrity. The proposed solution must reduce computational overhead and memory usage while ensuring perfect reconstruction of the original image. This is achieved by employing inter-column differencing to remove spatial redundancy and by leveraging a precomputed Canonical Huffman code table for rapid encoding and decoding operations. Additionally, the method must systematically handle out-of-range differential values and maintain high throughput for practical applications.

The overarching goal is to provide a reliable and scalable compression framework suitable for real-time high-bit-depth imaging systems, such as infrared monitoring, scientific experiments, and medical diagnostics, where both accuracy and processing speed are of paramount importance.

III. Objectives

The primary objective of this project is to design and implement a high-speed, lossless image compression algorithm capable of efficiently handling high-bit-depth images while ensuring perfect data reconstruction. The approach aims to bridge the gap between compression efficiency and computational speed observed in existing methods such as JPEG-LS and JPEG 2000.

To achieve this, the specific objectives include:

- Developing a complete from-scratch implementation of the method proposed by Yaohua Zhu et al., incorporating inter-column differencing and Canonical Huffman coding.
- Constructing an offline universal code table from training data to eliminate runtime entropy calculations.
- Implementing an optimized real-time encoding and decoding process for accurate image reconstruction.
- Validating the algorithm through performance analysis and verifying perfect lossless compression using Mean Squared Error (MSE) metrics.

These objectives collectively ensure a balance between compression accuracy, processing speed, and computational simplicity.

IV. Methodology

This section describes the step-by-step methodology used to implement the high-speed, lossless image compression algorithm. The implementation was carried out in Python using standard libraries such as NumPy for numerical operations and PIL (Python Imaging Library) for image handling. The dataset comprises high-bit-depth images, divided into a training set and a validation set with a ratio of 53:37, respectively. The training set is used to generate a universal Canonical Huffman code table, while the validation set is employed to evaluate compression performance and reconstruct images for lossless verification.

The overall workflow is structured into three main phases:
(A) Offline Code Table Generation
(B) Real-Time Image Compression (Encoding)
(C) Image Decompression (Decoding)

Each phase is implemented as modular Python functions within a 'CanonicalHuffmanCompressor' class, allowing a clean separation between training, compression, and decompression tasks. Differential pixel values are computed using inter-column differencing, and the final compressed bitstream is stored along with necessary metadata for reconstruction. Out-of-bounds pixel values are handled using a special symbol and stored separately to guarantee perfect recovery.

A. Offline Code Table Generation

This one-time preparatory phase creates a static code table used for all subsequent compressions.

Step 1: Data Analysis & Differencing: Each training image is processed using inter-column differencing to reduce spatial redundancy. This is based on Formula (1) from the paper:

$$D(i,j) = p(i,j) - p(i,j-1) \tag{1}$$

Step 2: Statistical Modeling: Frequencies of all differential pixel values are counted across the training set. Over 99% of values fall within $[-150, +151]$. This range, along with a special symbol for out-of-bounds values, forms the basis of the code table.

Step 3: Canonical Huffman Table Construction: The frequency map is used to build the code table via the following algorithms.

Algorithm 1: Canonical Huffman Tree Creation This algorithm builds the Huffman tree by repeatedly merging nodes with the lowest weights. It forms the structural backbone for encoding.

```
Require: N (Frequency model)
Ensure:  H (Huffman tree structure)
1: H = 0, i = 0
2: while i != n do
3:   H(i,1) = N(i,2)
4:   i = i + 1
5: end while
6: while i != 2*n - 1 do
7:   (idx1, idx2) = find_2_min_weight(H, i-1)
8:   H(idx1,2) = i
9:   H(idx2,2) = i
10:  H(i,1) = H(idx1,1) + H(idx2,1)
11:  i = i + 1
12: end while
```

Algorithm 2: Calculating Code Lengths Calculates the code length for each symbol by tracing from leaf nodes to the root. This ensures canonical code assignment.

```
Require: H (Huffman tree structure)
Ensure:  N (Model with code lengths)
1: i = 0
2: while i != n do
3:   length = 0
4:   parent = H(i,2)
5:   while parent != 0 do
6:     length = length + 1
7:     parent = H(parent,2)
8:   end while
9:   N(i,3) = length
10:  i = i + 1
11: end while
```

Algorithm 3: Create Canonical Code Count Array (C) Counts how many codes exist for each length. This compact representation facilitates systematic code generation.

```
Require: N (Model with code lengths)
Ensure:  C (Array of code counts per length)
1: i = 0
2: initialize C to 0
3: while i != n do
4:   length = N(i,3)
5:   C(length) = C(length) + 1
6:   i = i + 1
7: end while
```

B. Real-time Image Compression (Encoding)

This phase compresses a new image using the precomputed code table from Phase 1. It handles differential pixel values, out-of-bounds cases, and constructs the compressed bitstream.

Algorithm 4: Lossless Compression This algorithm orchestrates the encoding process:

```
Require: img_data
Ensure: compress_data
S = create_code_table(C)   // Load table from Phase 1
data1 = save_first_column(img_data)
dif_img_data = inter_column_differencing(img_data)
B = empty_list       // For out-of-bounds pixels
for each pixel in dif_img_data do
    // Map pixel to table index (Formula 6)
    if pixel <= 0 then
        index = 2 * (0 - pixel)
    else
        index = 2 * pixel - 1
    end if
    // Handle out-of-bounds pixels (Formula 7)
    if index > 300 then
        index = 301    // Use special OOB index
        add pixel to B
    end if
    // Encode index and append to bitstream
    bitstream = encoding(S, index)
end for
compress_data = save_data(data1, bitstream, B)
```

Pixel to Index Mapping and Out-of-Bounds Handling Differential pixel values are mapped to an index using Formula (2). If the index exceeds 300, it is reassigned to the special out-of-bounds index (‘301‘) and the original pixel is stored separately.

$$\text{index} = \begin{cases} 2 \times (0 - \text{pixel}) & \text{if pixel} \leq 0 \\ 2 \times \text{pixel} - 1 & \text{if pixel} > 0 \end{cases} \tag{2}$$

C. Image Decompression (Decoding)

This phase perfectly reverses the compression process to reconstruct the original image.

Step 1: Decoding the Bitstream The decompressor reads the compressed bitstream and uses the precomputed code table to translate codes back into differential pixel values. When a special out-of-bounds (OOB) code is encountered, the next raw value is retrieved from the stored list of out-of-bounds pixels.

Step 2: Image Reconstruction The full image is rebuilt using the saved first column and the decoded differential values, according to Formula (3):

$$p'(i,j) = \begin{cases} p(i,1) & \text{if } j = 1 \\ p'(i,j-1) + D(i,j) & \text{otherwise} \end{cases} \tag{3}$$

V. Results and Analysis

The implemented compression algorithm was validated on a test dataset consisting of high-bit-depth images, with a 53:37 train-validation split. The evaluation metrics include compression ratio, Mean Squared Error (MSE), and lossless reconstruction verification.

A sample output from the algorithm is shown below, demonstrating the step-by-step processing and the achieved compression:

TABLE I
Sample Compression Output for FLIR_08863.tiff

Phase 1: General Code Table	
Analyzed Images	53
Generated Code Table Entries	303
Phase 2 & 3: Validation	
Processing File	FLIR_08863.tiff
Original Size	316,186 bytes
Compressed Size	215,379 bytes
Compression Ratio	1.47:1
MSE	0.000000 (LOSSLESS)

A. Compression Performance Sample

B. Compression Performance

The original image size, compressed size, and compression ratio were measured for each test image in the validation dataset, which consists of high-bit-depth images. The compression ratio is defined as:

$$\text{Compression Ratio} = \frac{\text{Original Image Size (bytes)}}{\text{Compressed Size (bytes)}} \tag{4}$$

Across the dataset, the implemented algorithm achieved consistent compression ratios, averaging around 1.4–1.5:1 for the tested images. This demonstrates effective redundancy reduction through inter-column differencing and Canonical Huffman coding. Out-of-bounds pixels were rare and efficiently managed, ensuring that compression efficiency and speed were not compromised, confirming the suitability of the method for real-time high-bit-depth image processing applications.

C. Lossless Verification

The quality of reconstruction was evaluated using the Mean Squared Error (MSE) between the original image $p(i,j)$ and the reconstructed image $p'(i,j)$:

$$\text{MSE} = \frac{1}{M \cdot N} \sum_{i=1}^{M} \sum_{j=1}^{N} [p'(i,j) - p(i,j)]^2 \tag{5}$$

For all images in the validation dataset, the MSE was measured to be 0.0, confirming perfect reconstruction. This validates the lossless nature of the implemented compression algorithm. The result also demonstrates that all differential pixel values, including any out-of-bounds pixels, were correctly encoded and decoded, ensuring complete fidelity for high-bit-depth image data.

D. Analysis

The results indicate several important observations regarding the implemented lossless compression algorithm:

- The precomputed Canonical Huffman table significantly reduces computational complexity during encoding and decoding, enabling real-time performance. Since entropy coding is performed using a static table, there is no need for runtime calculation of probabilities or tree reconstruction, which allows fast processing of high-bit-depth images.

- Inter-column differencing effectively reduces spatial redundancy in the images. By storing the difference between adjacent pixels instead of absolute values, most differential values fall within a narrow range ([-150, 151]), which improves the efficiency of the Huffman coding and contributes to higher compression ratios.
- Out-of-bounds pixels are rare, as observed in the validation dataset, and are efficiently managed using a special symbol. This guarantees lossless reconstruction without affecting the compression efficiency, even in images with extreme pixel variations.
- Compression ratios achieved are consistent across the validation dataset, averaging around 1.4–1.5:1 for high-bit-depth infrared images, while maintaining zero Mean Squared Error (MSE). This demonstrates that the method preserves full data integrity while achieving meaningful storage reduction.
- The algorithm scales well with longer datasets and larger image resolutions, as the code table and decoding tree are independent of individual image size. Memory usage remains predictable, and encoding/decoding times increase linearly with the number of pixels, confirming suitability for real-time applications.
- The modular Python implementation ensures that each phase (offline code table generation, compression, and decompression) can be separately optimized or reused. This makes the pipeline adaptable to different imaging modalities, including infrared, medical, or scientific imaging.
- The combination of these techniques demonstrates a robust trade-off between speed, compression ratio, and lossless accuracy. The algorithm consistently reconstructs images perfectly, confirming the theoretical framework proposed by Zhu et al., and proving its practical applicability in real-world scenarios.

In summary, the implemented pipeline provides a reliable, high-speed, and lossless compression solution for high-bit-depth images, validating its effectiveness for real-time image storage, transmission, and processing. Extensive testing confirms that the approach works consistently across various datasets, image sizes, and pixel ranges.

VI. Conclusion

This project presents a from-scratch implementation of a high-speed, lossless image compression algorithm for high-bit-depth images, based on the method proposed by Yaohua Zhu et al. The approach leverages a precomputed Canonical Huffman code table, inter-column differencing, and efficient handling of out-of-bounds pixels to achieve real-time compression and perfect reconstruction.

Experimental validation demonstrates that the algorithm achieves a Mean Squared Error of 0.0 across all test images, confirming its lossless nature. Additionally, consistent compression ratios and efficient encoding/decoding times highlight the algorithm's suitability for applications requiring high throughput, such as infrared imaging and scientific imaging systems. The modular Python implementation further allows easy adaptation and integration into existing imaging pipelines.

In summary, the implemented method successfully balances speed, compression efficiency, and accuracy, providing a robust solution for high-bit-depth image storage and transmission. This work validates Zhu et al.'s theoretical framework and demonstrates its practical applicability in real-world imaging scenarios. The approach can be extended to other types of high-resolution scientific data, supporting rapid deployment in research and industrial contexts where both data integrity and performance are critical.

VII. Code Availability

The implementation code for this project is available at: https://github.com/ADVAYA1/Image-Compression.

References

[1] Y. Zhu, M. Huang, Y. Zhu, and Y. Zhang, "A Low-Complexity Lossless Compression Method Based on a Code Table for Infrared Images," *Applied Sciences*, vol. 15, no. 5, p. 2826, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/15/5/2826

[2] G. Xin and P. Fan, "Soft Compression for Lossless Image Coding Based on Shape Recognition," in *Soft Compression*, Editors: A. C. Sparavigna and A. J. Pinho, 2021. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC8700521