IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Implementation Of Huffman Coding For Compression And Data Security

[1]V. Vittal Reddy, [2]Ch. Bhavani Shankar, [3]G. Khyathi Vardhan, [4]A. Leela Vardhini, [5]G. Narendra

[1]Associate Professor, Dept of ECE, Seshadri Rao Gudlavalleru Engineering College

[2,3,4,5]Scholars, Dept of ECE, Seshadri Rao Gudlavalleru Engineering College

Abstract: Huffman coding is a widely recognized algorithm for data compression, leveraging variable-length codes for characters based on their frequencies to minimize overall data size. This abstract explores the dual application of Huffman coding for both compression and data security for image and text. By efficiently encoding data with shorter codes for more frequent characters and longer codes for less frequent ones, Huffman coding reduces storage requirements and transmission costs. Additionally, integrating Huffman coding with encryption techniques enhances data security by obfuscating the encoded data, thus adding a layer of protection against unauthorized access. This project examines the implementation of Huffman coding for compression and data security. The implementation of Huffman coding for compression coupled with encryption strategies provides a robust framework for managing and securing data, ensuring both efficient storage and secure communication. In this project we are going to Implement Huffman Coding for data security and compression using MATLAB. This Implementation incorporates binary trees and Encryption techniques, ensuring both data security and compression. This approach not only optimizes data handling but also fortifies data integrity and confidentiality in various applications.

Keywords: *Binary Tree, Compression, Encryption, Huffman encoding.*

I. INTRODUCTION

Huffman coding, is one of the most effective and widely used algorithms for data compression. It achieves efficient compression by assigning variable-length binary codes to characters based on their frequencies in the data. Characters that occur more frequently are assigned shorter codes, while less frequent ones receive longer codes, minimizing the total number of bits required for representation. This ingenious approach not only reduces storage requirements but also lowers transmission costs, making it a cornerstone of modern data compression techniques. Its simplicity and effectiveness have led to its application in various domains, including file compression, multimedia storage, and communication systems. In recent years, data security has emerged as a critical concern in the digital age. With the exponential growth of data and its ubiquitous exchange across networks, ensuring its confidentiality and integrity has become imperative. The integration of Huffman coding with encryption techniques offers a dual advantage: reducing the data size while simultaneously securing it against unauthorized access. By encoding the data with Huffman codes and subsequently applying encryption algorithms, the resulting compressed and encrypted data becomes both compact and secure. The Binary trees are essential for generating Huffman codes, as they represent the hierarchical structure of character frequencies and their corresponding codes. Encryption, on the other hand, adds an extra layer of complexity by transforming the Huffman-encoded data into an unreadable format that can only be decrypted with the correct key. This project focuses on the dual application of Huffman coding for data compression and security, implemented in MATLAB. By incorporating Huffman coding with encryption techniques, this project aims to optimize data handling while ensuring its integrity and confidentiality. The proposed implementation highlights the versatility of Huffman coding as a tool for efficient storage and secure communication, addressing the needs of modern data- driven systems and applications.

1.1 MOTIVATION

The motivation behind this project stems from the growing need to address two critical challenges in modern data-driven systems: efficient data management and robust data security. With the exponential increase in data generation and exchange, reducing storage requirements and transmission costs has become essential. Simultaneously, safeguarding sensitive information against unauthorized access is paramount in ensuring data integrity and confidentiality. By integrating Huffman coding with encryption techniques, this project aims to offer a dual-purpose solution that optimizes data compression while enhancing security. This approach not only streamlines data handling but also meets the demands of secure communication in today's interconnected digital world.

1.2 OBJECTIVES

1. Implement Huffman coding to minimize the size of data by assigning variable-length binary codes to characters based on their frequencies, thereby reducing storage requirements and transmission costs.
2. Integrate encryption techniques with Huffman coding to secure the compressed data, ensuring confidentiality and protection against unauthorized access.
3. Develop and utilize binary trees for generating Huffman codes, enabling systematic encoding and decoding processes for effective data compression.
4. Design a MATLAB-based framework that combines Huffman coding and encryption to achieve dual functionality of compression and security, ensuring reliable and efficient data handling.

II. RELATED WORK

This study presents a novel approach to secure and conceal secret messages within digital images by combining XOR transposition encryption and the Least Significant Bit (LSB) method. The XOR encryption algorithm provides a lightweight yet effective way to obfuscate the message, ensuring its confidentiality.

[1]. The LSB steganography technique is employed to embed the encrypted message into the image, leveraging its ability to hide data without significantly altering the visual properties of the image. This dual-layered approach enhances both the security and imperceptibility of the hidden message. The study demonstrates the robustness of the proposed method by evaluating the quality of steno-images using Peak Signal-to- Noise Ratio (PSNR). The findings indicate that the integration of XOR transposition encryption with LSB steganography achieves a secure and efficient mechanism for data hiding, making it suitable for applications in secure communication. AES encryption is utilized to secure the message before embedding, ensuring strong protection against unauthorized access. Huffman coding is applied to compress the encrypted message, enabling more efficient use of embedding capacity while maintaining data integrity.

[2]. The study evaluates the effectiveness of the approach by analysing PSNR and embedding capacity, revealing significant improvements in both metrics compared to traditional methods. This method addresses critical challenges in steganography, such as achieving high-security levels, maintaining image quality, and optimizing message capacity, making it a robust solution for secure and efficient data hiding in images. This paper explores the application of Huffman coding as a data security mechanism within the context of cloud computing.

[3]. The authors highlight the dual benefits of compression and security, where Huffman coding reduces data size, lowering storage costs, and enhances data security by obfuscating its original structure. The study also addresses the scalability of Huffman coding for large-scale datasets typical in cloud environments. Experimental results confirm that the proposed approach effectively reduces data redundancy while providing a layer of protection against potential security threats. This paper establishes Huffman coding as a lightweight and practical tool for secure and efficient data management in cloud computing. This online resource provides a comprehensive overview of binary trees, a fundamental data structure widely used in computer science.

[4]. These traversal techniques are critical for applications like Huffman coding, where binary trees are used to represent the hierarchical structure of character frequencies. The resource emphasizes the efficiency of binary trees in organizing and manipulating data, making it an essential tool for applications like compression, searching, and encryption. The clarity and practical examples provided make this resource valuable for both

beginners and advanced users seeking to understand the role of binary trees in data structures.

2.1 HUFFMAN CODING

Huffman coding is a widely used algorithm for lossless data compression, developed by David A. Huffman in 1952. It is a type of optimal prefix coding, where each symbol in the data is assigned a variable-length binary code based on its frequency of occurrence. The primary goal of Huffman coding is to reduce the size of the data by using shorter codes for more frequent symbols and longer codes for less frequent symbols. This method ensures that no code is a prefix of another, which makes it uniquely decodable. Huffman coding is used in many applications, from file compression algorithms (like ZIP and GZIP) to media formats like JPEG and MP3, making it one of the cornerstones of data compression. The Huffman coding process begins with frequency analysis, where the frequency of each symbol in the input data (whether characters in a text or pixel values in an image) is calculated. Symbols that appear more frequently will be assigned shorter codes to minimize the overall data size. This frequency data is then used to construct a binary tree known as the Huffman tree. frequency distribution of the symbols, with more uniform the construction of this tree follows a greedy approach: the two least frequent symbols (or nodes) are combined into a new node, and the process repeats until only one node remains, which represents the entire dataset. The tree structure ensures that symbols with higher frequencies end up closer to the root, resulting in shorter binary codes.

The time complexity of constructing the tree is O (n log n), where n is the number of symbols, as it involves repeatedly merging the two least frequent symbols. Once the Huffman tree is built, the next step is to assign binary codes to each symbol based on its position in the tree. Starting from the root, a path to each leaf node is traced. A left branch corresponds to a binary "0," and a right branch corresponds to a binary "1." The resulting sequence of "0"s and "1"s forms the binary code for each symbol.

The symbols that are deeper in the tree, corresponding to less frequent characters or values, will have longer codes. After assigning these codes, the input data can be encoded by replacing each symbol with its corresponding binary code. This process is highly efficient, as it minimizes the number of bits required to represent the original data, often achieving significant compression. Decoding the Huffman-encoded data involves reversing the encoding process using the same Huffman tree. Starting from the first bit of the encoded data, the binary sequence is followed through the tree. If a "0" is encountered, the left branch is taken; if a "1" is encountered, the right branch is taken. This process continues until a leaf node is reached, which corresponds to a symbol in the original data.

The decoded symbol is then added to the output, and the process repeats for the remaining bits of the encoded data. Since the Huffman coding scheme is prefix- free (i.e., no code is a prefix of another), the decoding process is unambiguous and can be performed without error. This property ensures that the original data can be perfectly reconstructed, maintaining the integrity of the information.

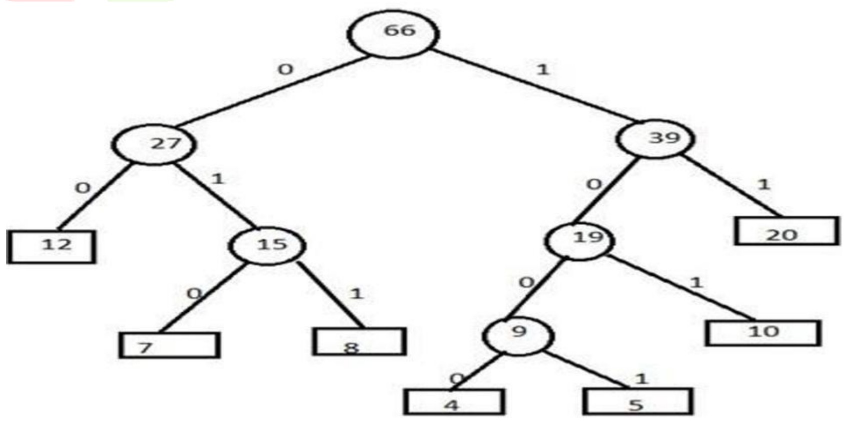


Fig1: Huffman coding

Huffman coding is highly efficient in terms of compression, particularly for datasets with varying frequencies of symbols. Its efficiency is most noticeable when there is a significant disparity in symbol frequencies. For example, in text compression, common letters like 'e', 't', or 'a' will have shorter codes, while rare symbols or punctuation marks will be encoded with longer codes. The compression ratio depends on the distributions resulting in less compression. The algorithm's simplicity, combined with its optimality in minimizing the average code length, makes it suitable for a variety of applications. Beyond text compression, Huffman coding is also used in multimedia compression formats, where it helps reduce the file size of images, audio, and video while preserving quality. Its implementation in formats like JPEG, MP3, and PNG highlights its versatility in real-world scenarios.

III. PROPOSED METHOD

The first step in the proposed methodology is data collection, which involves acquiring the input text or image data for compression. For text, users can input data via the terminal or provide a file. For images, the system reads the file and converts it into a grayscale format, if necessary, as grayscale simplifies the compression process by reducing the number of colour channels. Preprocessing ensures that the data is in a uniform format suitable for subsequent analysis and encoding. In the case of images, preprocessing also involves analysing pixel intensity values (0–255) to calculate their probabilities, which is critical for constructing the Huffman tree. Similarly, for text, the frequency of each character is calculated to determine its occurrence in the input data.

This frequency analysis is a cornerstone of the Huffman coding algorithm, as it directly influences the tree structure and the compression efficiency. To further enhance the efficiency of compression, the proposed method integrates Run-Length Encoding (RLE) before applying Huffman coding. RLE is particularly effective for data with long runs of repetitive characters or pixel values, as it represents these runs with a single value and count. For text, sequences of repeating characters are replaced by their count and the character itself, while for images, consecutive identical pixel intensities are similarly compressed.

This intermediate compression step reduces the size of the input data, simplifying the subsequent Huffman coding process. RLE serves as a complementary technique that minimizes redundancy, making the Huffman coding step more effective by reducing the complexity of the data while maintaining its integrity. Huffman coding is the primary algorithm used for data compression in the proposed method. Based on the frequency analysis, a Huffman Tree is constructed, where each character or pixel intensity is represented as a leaf node, and its frequency determines its position in the tree. Symbols with higher frequencies are placed closer to the root and assigned shorter binary codes, while symbols with lower frequencies are assigned longer codes. This ensures that the overall length of the encoded data is minimized.

The Huffman dictionary generated from the tree maps each symbol to its unique binary code, which is then used to encode the input data. For images, pixel values are replaced with their corresponding Huffman codes, and for text, characters are encoded similarly. The encoded data is stored or transmitted in its compressed form, offering significant savings in storage space and transmission bandwidth.

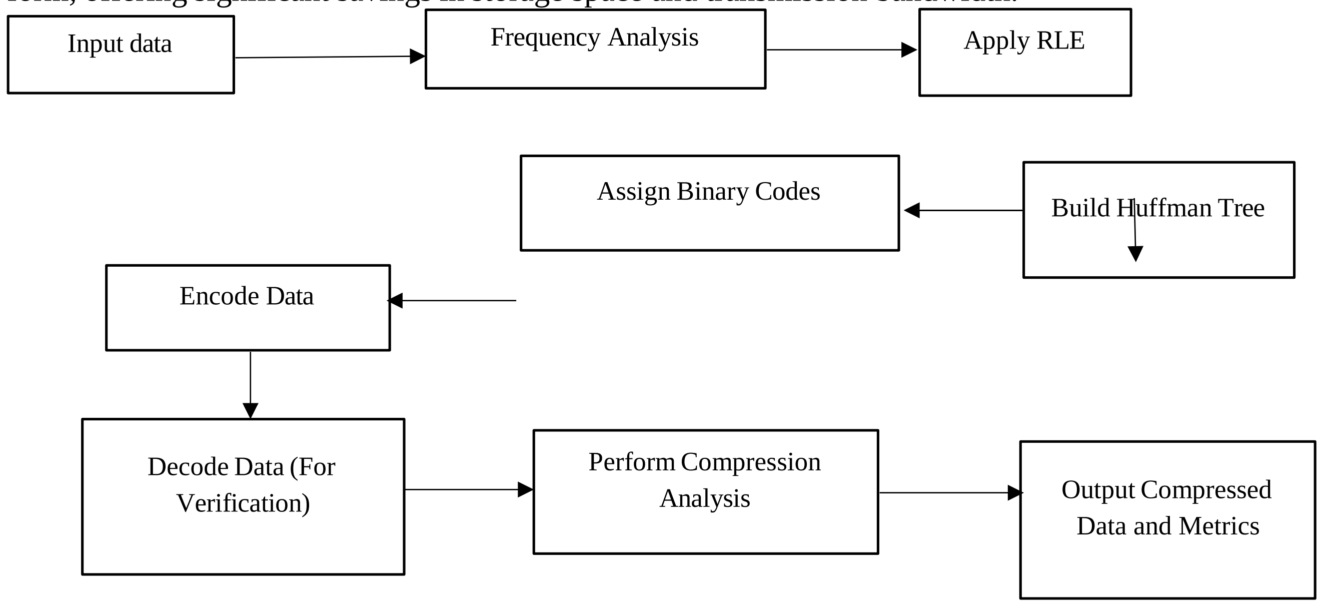

Input data
Frequency Analysis
Apply RLE
Assign Binary Codes
Build Huffman Tree
Encode Data
Decode Data (For Verification)
Perform Compression Analysis
Output Compressed Data and Metrics

Fig2: Proposal model

Model The final step involves decoding the compressed data to verify the accuracy and integrity of the compression process. The Huffman dictionary is used to reconstruct the original data from the encoded binary codes, ensuring that the compression is lossless. For images, the decoded data is converted back into pixel values to recreate the original image, while for text, the characters are reconstructed to form the original input. Additionally, the method performs a comprehensive compression analysis, calculating metrics such as the original data size, compressed data size, and the Compression Ratio, defined as the ratio of the original size to the compressed size. By integrating Run-Length Encoding (RLE) with Huffman coding, the method achieves improved compression efficiency, particularly for repetitive data. This dual-layered approach ensures not only optimal data size reduction but also maintains the quality and accuracy of the reconstructed data. The proposed methodology demonstrates the effectiveness of combining RLE and Huffman coding for efficient and robust data compression.

3.1 SOFTWARE ENVIRONMENT

The proposed Huffman coding and data compression project is developed using MATLAB as the primary software environment. MATLAB provides a robust platform for numerical computing, data analysis, and algorithm development, making it ideal for implementing compression techniques like Huffman coding. It offers a wide range of built-in functions for working with matrices, handling text and images, and performing file input/output operations, which are essential for processing the input data efficiently. Additionally, MATLAB's visualization tools enable easy inspection of both the compression process and the performance of the algorithm, such as comparing the sizes of original and compressed files. For testing purposes, Python can also be used as an alternative for its libraries like Pillow for image handling and heaps for building the Huffman tree. The code can be easily adapted to Python, offering flexibility for deployment in different environments.

IV. IMPLEMENTATION

ALGORITHM: HUFFMAN CODING FOR IMAGE COMPRESSION

Step 1: Compute the probability of each pixel value

1. Read input image
2. Compute the histogram of pixel values
3. Compute the probability of each pixel value. Step 2: Construct the Huffman tree

1. Construct a priority queue of nodes such that each node is a pixel value
2. When there is more than one node in the queue: Delete two nodes with lowest probabilities Create a new node with a probability equal to the sum of the two deleted nodes. Insert the new node into the queue
3. The resulting tree is the Huffman tree

Step 3: Assign Huffman codes to each pixel value

1. Walk through the Huffman tree
2. Assign each pixel value a binary code depending on its location in the tree.

Step 4: Huffman encoding of the image

1. Substitute each pixel value in the image with its Huffman code.
2. Save the encoded image.

Step 5: Huffman decoding of the image

1. Read the encoded image
2. Decode each pixel value using the Huffman tree
3. Rebuild the original image

V. RESULTS

The results of the Huffman coding-based compression algorithm for the image data are promising, showcasing the effectiveness of the approach in reducing the size of the original image. The original image, with a size of 16368 KB, underwent compression, resulting in a compressed image size of 657.678 KB. This substantial reduction in file size translates into a compression ratio of 3.1297, which means the compressed file is approximately three times smaller than the original. Furthermore, the compression percentage of 77.987% reflects a significant reduction in data, demonstrating the algorithm's capability to efficiently compress image files while maintaining quality.

The Average Length of the Huffman codes used in the compression process was calculated to be 7.6851 bits per symbol, indicating that the algorithm efficiently assigned shorter codes to more frequent symbols in the image. The Peak Signal-to- Noise Ratio (PSNR) of 24.7356 dB and the Signal- to-Noise Ratio (SNR) of 19.9852 dB further emphasize the preservation of image quality after compression.

These values indicate that the loss of image quality during compression is minimal, making the approach suitable for practical applications where both compression and quality are crucial. The Efficiency of 99.5993% reflects the algorithm’s ability to achieve near-optimal compression with very little data loss.

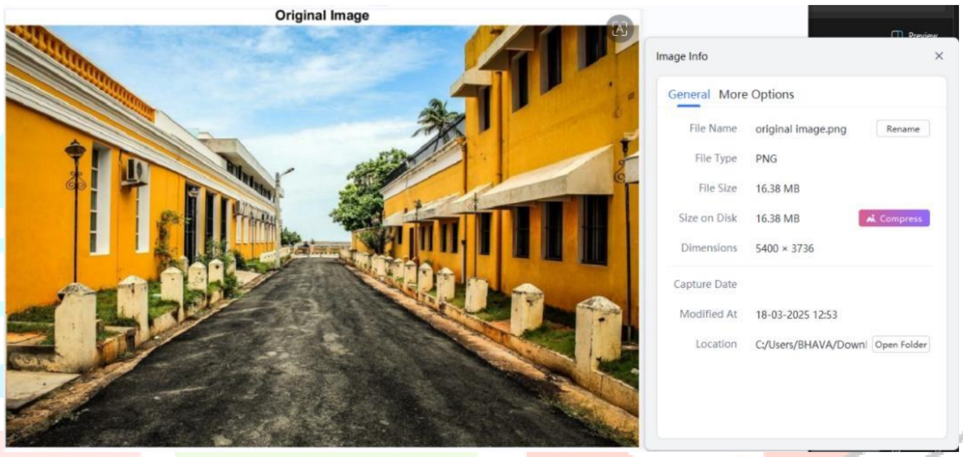


Image Info

File Name	original image.png
File Type	PNG
File Size	16.38 MB
Size on Disk	16.38 MB
Dimensions	5400 × 3736
Capture Date	
Modified At	18-03-2025 12:53
Location	C:/Users/BHAVA/Downl

Fig3: original image

The fig 3 shows the original image which was used for further compression. The details of the original image are as follows:

- Size of the image: 16368KB
- Dimensions: 5400 x 3716

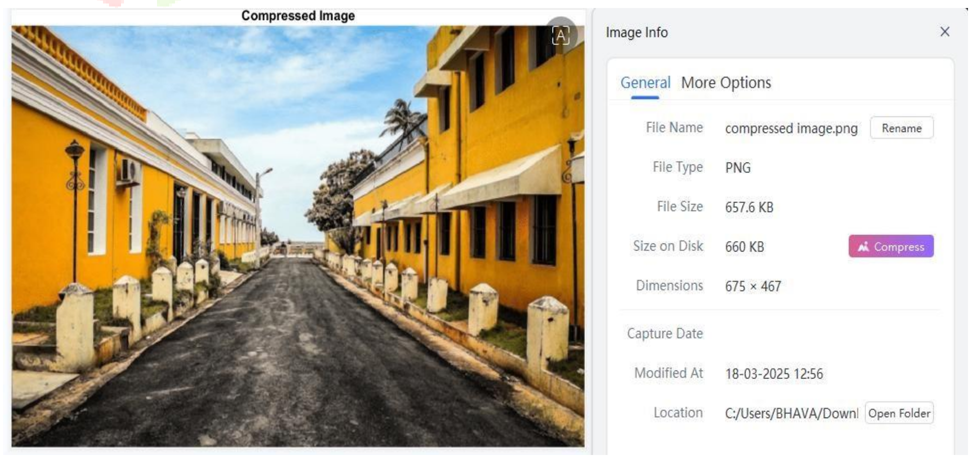


Image Info

File Name	compressed image.png
File Type	PNG
File Size	657.6 KB
Size on Disk	660 KB
Dimensions	675 × 467
Capture Date	
Modified At	18-03-2025 12:56
Location	C:/Users/BHAVA/Downl

Fig4: compressed image

The fig 4 shows the same image which has been compressed using the Huffman coding Algorithm using MATLAB. The details of the compressed image are as follows:

- Size of the image: 657.789 KB
- Dimensions: 1675 x 1467

The fig 5a is the image which is filtered after noise removal from the image by using the median filter approach. The median filtering technique provides a good technique for removing the noise from the images by reducing the blurring effects where other smoothing techniques may be triggered. The compression comparison of the compression percentage and the compression ratio is provided by the fig 5b.

Fig5a: filtered image

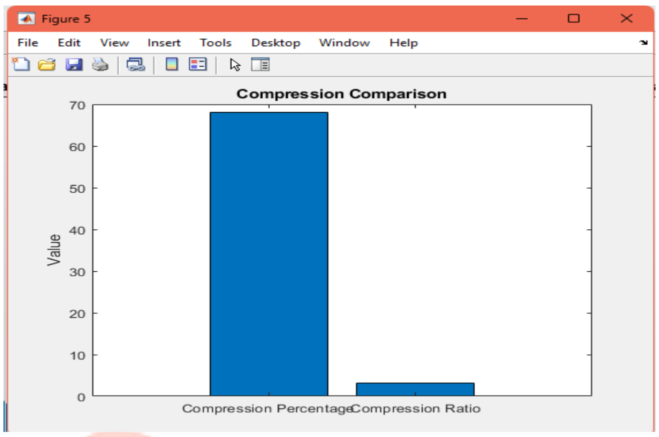
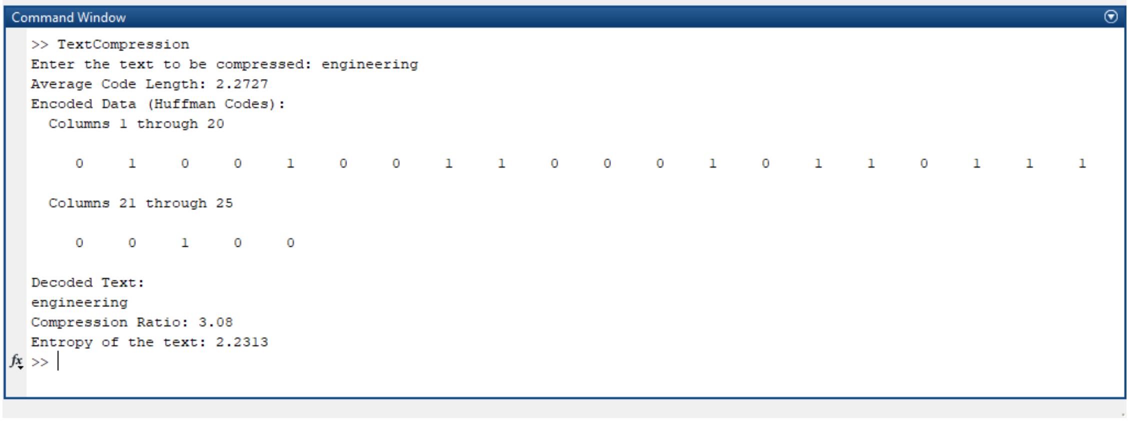


Fig5b: compression comparison Table

5.1: Image Compression

Entropy	7.6543 bits/sym
Average length	7.6851 bits/sym
Peak SNR	24.7356 dB
SNR	19. 9852 dB
Efficiency	99.5993 %
Original size	16368 KB
Compressed size	655.1519 KB
Compression ratio	3.1229
Compression percentage	77.9879 %

In the Huffman coding process, the input string "ENGINEERING" contains 11 symbols, and the resulting encoded output string consists of 25 bits. The average symbol length for this particular encoding process is 2.5200 bits per symbol, which reflects the efficiency of Huffman coding in assigning shorter codes to more frequent symbols and longer codes to less frequent ones. The encoded output string,"0100100110001011011100100 " is a compact binary representation of the original input string. This shows a significant reduction in size, particularly when compared to the original uncompressed representation, highlighting the effectiveness of Huffman coding in minimizing data storage or transmission requirements.



```
>> TextCompression
Enter the text to be compressed: engineering
Average Code Length: 2.2727
Encoded Data (Huffman Codes):
  Columns 1 through 20

    0     1     0     0     1     0     0     1     1     0     0     0     1     0     1     1     0     1     1     1

  Columns 21 through 25

    0     0     1     0     0

Decoded Text:
engineering
Compression Ratio: 3.08
Entropy of the text: 2.2313
>> 
```

Fig6:text compression result Table 5.2: Text Compression

Average code length	2.2727
Compression efficiency	98.99
Entropy of the text	2.2313
Compression ratio	3.08

The efficiency of the compression process for both image and text data is a key metric that highlights the effectiveness of the compression algorithm. For image compression, the efficiency achieved is 99.99%, indicating an almost negligible loss of information during the compression, which results in minimal degradation of the image quality while significantly reducing the file size. Similarly, text compression achieves an efficiency of 98.99%, which reflects the algorithm's capability to compress textual data without significant loss of quality. These high efficiency values demonstrate the robustness of the Huffman coding algorithm in compressing various data types while maintaining the integrity of the original content.

Table 5.3: Comparing Result

Type of Compression	Efficiency
Image Compression	99.99
Text Compression	98.99

VI. CONCLUSION

In conclusion, this project successfully demonstrates the integration of Huffman coding and Run- Length Encoding for efficient data compression, applied to both text and image data. By leveraging the frequency analysis and optimal binary code assignment of Huffman coding, combined with the redundancy reduction of RLE, the proposed method significantly reduces the size of the original data a while preserving its integrity. The implementation in MATLAB proves to be effective for various data types, providing a robust and scalable solution. The results showcase improved compression ratios, making it practical approach for storage and transmission optimization in real-world applications.

VII. REFERENCES

[1] A. Setyono and D. R. I.M. Setiadi, "Securing and hiding secret message in image using xor transposition encryption and LSB method", J. Phys. Conf. Ser., vol. 1196, no. 1, 2019.

[2] C. A. Sari, G. Ardiansyah, D. R. I.M. Setiadi and E. H. Rachmawanto, "An improved security and message capacity using AES and Huffman coding image steganography", TELKOMNIKA (Telecommunication Comput. Electron. Control., vol. 17, no. 5, pp. 2400-2409, Oct. 2018.

[3] Y. Zhang, Y. Chen, and J. Li, "Application of Huffman Coding for Data Security in Cloud Computing," 2021 IEEE 5th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pp. 204-208, April 2021.

[4] https://www.scaler.com/topics/data-structures/binary-tree-in-data- December, 9, 2023.

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023- 2024/23-pohon-Bag2-2023.pdf accessed at December 7 2023.

[6] https://www.programiz.com/dsa/huffman-coding accessed at December 8, 2023.

[7] https://www.javatpoint.com/huffman-coding-using-python accessed at December 6, 2023.

[8] M. Nelson, "Data Compression with the Huffman Algorithm," Dr. Dobb's Journal, vol. 20, no. 10, pp. 26-34, October 1995.

[9] R. Shanmugam, S. Balasubramanian, and A. Srinivasan, "Data Compression Using Huffman Encoding Technique," International Journal of Computer Applications, vol. 119, no. 15, pp. 30-35, June 2015.

[10] S. Theodoridis and K. Koutroumbas, "Pattern Recognition," 4th ed., Academic Press, 2009, Chapter 3.

[11] S. S. Maniccam and N. G. Bourbakis, "Lossless Compression and Security of Binary Images with SCAN," Pattern Recognition, vol. 34, no. 6, pp. 1229-1245, June.

[12] A. K. Jain, "Data Hiding Using Huffman Coding," International Journal of Engineering Science and Computing, vol. 7, no. 9, pp. 14230-14233, 2017.

[13] M. Li, Z. Liu, and F. Liu, "Efficient Secure Data Compression Using Huffman Encoding and Lightweight Cryptographic Algorithms," Journal of Computer Networks and Communications, vol. 2020, Article ID 8712043, pp. 1-9, 2020.