AMERICAN JOURNAL OF SMART TECHNOLOGY AND SOLUTIONS (AJSTS)

ISSN: 2837-0295 (ONLINE)

VOLUME 3 ISSUE 2 (2024)

PUBLISHED BY

E-PALLI PUBLISHERS, DELAWARE, USA

American Journal of Smart Technology and Solutions (AJSTS)

Volume 3 Issue 2, Year 2024
ISSN: 2837-0295 (Online)
DOI: https://doi.org/10.54536/ajsts.v3i2.3566
https://journals.e-palli.com/home/index.php/ajsts

A Comparative Study of Text-Based Lossless Compression

Shyam Maharjan[1*], Er. Sujan Poudel[1], Dipesh Tandukar[1]

Article Information

Received: August 18, 2024

Accepted: September 20, 2024

Published: September 24, 2024

Keywords

Compression Ratio, Data Compression, Huffman Encoding, Lempel-Ziv-Welch (LZW), Lossless Compression

ABSTRACT

Lossless data compression is a critical technique used to reduce file sizes without any loss of information during the encoding and decoding processes. This study presents a comparative analysis of two widely-used lossless compression algorithms: Huffman Encoding and Lempel-Ziv-Welch (LZW). The primary objective is to evaluate the performance of these algorithms in terms of compression ratio, compression time, decompression time, and space savings. The analysis was conducted on 100 files of varying sizes. The results demonstrate that the LZW algorithm outperforms Huffman Encoding, offering superior compression ratios, faster compression and decompression times, and greater disk space savings. These findings highlight the effectiveness of LZW for efficient data compression in practical applications.

INTRODUCTION

Data compression is a fundamental technique that reduces the size of digital information by encoding it using fewer bits than its original representation. This process is crucial for optimizing storage and transmission of data, making it a key component in various applications such as file storage, data transfer, and media streaming. Data compression can be broadly categorized into two types: lossless and lossy compression (Holtz, 1993).

Lossless data compression ensures that no data is lost during the compression process. It works by eliminating redundancy within the data, allowing the original file to be perfectly reconstructed from the compressed version. This makes lossless compression ideal for text, software, and other types of data where any loss of information is unacceptable. On the other hand, lossy data compression permanently removes some data, which is often acceptable for media files like audio, video, and images where minor data loss does not significantly impact the quality.

There are Two Widely-Used Algorithms

Huffman Encoding and Lempel-Ziv-Welch (LZW). Both of these algorithms are integral to the field of data compression and are known for their efficiency in reducing file sizes without losing any information.

Huffman Encoding operates by constructing a binary tree of nodes, encoding symbols or characters based on their frequency of occurrence. The more frequent a character is, the fewer bits are required to represent it, making Huffman Encoding particularly effective for files with a skewed frequency distribution. LZW compression, on the other hand, uses a dynamic dictionary to encode data. As the file is processed, substrings are added to the dictionary, allowing repeated substrings to be represented by shorter codes, which can significantly reduce the file size.

In this study, we analyze the performance of Huffman Encoding and LZW compression in terms of compression ratio, compression time, decompression time, and space savings. By conducting experiments on 100 files of varying sizes, we aim to provide a comprehensive comparison of these two algorithms and determine their effectiveness in real-world applications.

This research's primary objective is to comprehensively evaluate two prominent lossless data compression algorithms: Huffman Encoding and Lempel-Ziv-Welch (LZW).

LITERATURE REVIEW

The concept of data compression has a rich history, dating back to the invention of Morse code in 1838, which can be considered an early example of data compression. However, modern data compression began in the late 1940s with the development of information theory, which laid the groundwork for more sophisticated compression methods. One of the most significant milestones in this field was the introduction of the Huffman coding algorithm in 1951, which provided an optimal method for lossless data compression. In the late 1970s, software compression programs began to emerge, many of which were based on adaptive Huffman coding. By the mid-1980s, the Lempel-Ziv-Welch (LZW) algorithm had become a staple in general-purpose compression systems, demonstrating its effectiveness across various applications.

Gopinath and Ravisankar (2020) stated that transmitting large volumes of data from a monitoring field to a central unit is particularly challenging when communication bandwidth is limited, leading to potential data overflow and loss on single-board computers. Given the fixed sampling frequency and unchangeable bandwidth, data

[1] Nepal Kasthamandap College, Nepal
[*] Corresponding author's e-mail: samymhr31@gmail.com

compression becomes essential to prevent these issues by minimizing the size of data for storage and transmission. Gupta (2017) explored various lossless data compression techniques and evaluates their performance in terms of time and space complexity. Focusing on compressing different media formats like text, DOC, BMP, PNG, and WAV files, the study examines Huffman coding, which generates variable-length codes for each symbol; LZW, a dictionary-based technique; and Shannon-Fano coding, which generates binary codes based on symbol frequency. By calculating the mean compression ratio, compression factor, and compression time for these methods, the analysis helps identify the most suitable compression technique for specific file formats in practical applications.
Verdú (2014) presented a comprehensive analysis of the best achievable rate and other fundamental limits in variable-length strictly lossless compression. It reveals a strong connection between the fundamental limits of fixed-to-variable lossless compression, both with and without prefix constraints, in the non-asymptotic regime. Precise quantitative bounds are established, linking the optimal code lengths to the source information spectrum, and providing an exact analysis for arbitrary sources. The study also proved fine asymptotic results for general mixing sources and introduces explicit Gaussian approximation bounds for the best achievable rate on Markov sources. Key concepts such as source dispersion and varentropy rate are defined and characterized, offering a tight approximation of the fundamental non-asymptotic limits for fixed-to-variable compression, except for very small block lengths.
Patel *et al.* (2012) presented parallel algorithms and implementations of a bzip2-like lossless data compression scheme optimized for GPU architectures. Our approach parallelizes the key stages of the bzip2 compression pipeline: Burrows-Wheeler transform (BWT), move-to-front transform (MTF), and Huffman coding. Specifically, we employ a two-level hierarchical sort for BWT, introduce a novel scan-based parallel MTF algorithm, and implement a parallel reduction method for constructing the Huffman tree. Through detailed performance analysis, we highlight the strengths and weaknesses of each algorithm and propose potential improvements. Despite these optimizations, our GPU implementation is 2.78× slower than bzip2, with BWT and MTF-Huffman being 2.89× and 1.34× slower on average, respectively.
Konecki *et al.* (2011) stated that data compression plays a crucial role in information and communication technologies by saving storage space and reducing network transmission bandwidth. This paper focused on lossless data compression, providing an overview of the algorithms used in popular data archiving tools. Since the compression rate varies significantly depending on the data type, the study tests a range of commonly used file types. By examining different tools that implement known algorithms in various forms, the paper identifies which tools offer the best compression capabilities, the fastest performance, and the most optimal balance between compression efficiency and speed.
Jones (2003) introduced the X-MatchPRO, a high-speed lossless data compression algorithm with a hardware implementation that achieves data-independent throughputs of 1.6 Gbit/s for both compression and decompression using low-cost reprogrammable field-programmable gate array (FPGA) technology. The full-duplex implementation enables a combined performance of 3.2 Gbit/s. The paper detailed the features of the algorithm and architecture that facilitate these high throughputs, and compares the X-MatchPRO with other commercially available data compressors in terms of technology, compression ratio, and throughput. X-MatchPRO is a fully synchronous design, proven in silicon, and is specifically aimed at enhancing Gbit/s storage and communication applications.
Patauner *et al.* (2011) introduced a compression system optimized for reducing data from pulse digitizing electronics, commonly used in High Energy Physics (HEP) experiments, such as those in calorimeters and Time Projection Chambers (TPCs). Using the ALICE experiment's TPC as a case study, the paper presents a novel compression method that surpasses conventional lossless compression by compressing entire vectors of digitized samples rather than individual uncorrelated samples. The method approximates incoming vectors with digitized reference vectors stored in memory and compresses only the differences using Huffman coding, a process akin to vector quantization combined with Huffman coding. Initial evaluations in Matlab using ALICE TPC data achieved a 49% compression rate, exceeding the 38% theoretical limit set by the data's entropy. The method was further implemented in Verilog and tested on a Virtex-4 development board at 80 MHz, with successful results demonstrated using cosmic ray data.

MATERIALS AND METHODS

This research aims to evaluate and compare the performance of Huffman Encoding and Lempel-Ziv-Welch (LZW) algorithms in lossless data compression. The study involves implementing both algorithms, conducting tests across a diverse set of files, and analyzing their performance through various metrics, including compression ratio, compression and decompression times, and disk space savings. Statistical methods are employed to assess the significance of differences in performance, and the results are validated through cross-verification. A total of 100 text based samples were taken for the purpose of analyzing the loss compression. For each sample original size, size after compression, entropy, compression ratio, disk saving, compression time and decompression time using two algorithm i.e. LZW and Huffman Encoding algorithm.

LZW (Lempel-Ziv-Welch) Algorithm

LZW (Lempel-Ziv-Welch) is a widely used lossless data compression algorithm that operates by replacing sequences of characters in the input data with shorter codes from a dynamically constructed dictionary. The algorithm starts with a dictionary containing all possible single-character sequences, and as it processes the input, it identifies the longest match in the dictionary, outputs its corresponding code, and then adds a new entry that extends the current match by one character. This process allows the dictionary to grow and capture increasingly complex patterns, leading to efficient compression, especially for data with repeated sequences. LZW is commonly used in applications like GIF image compression and file compression utilities due to its simplicity and effectiveness.

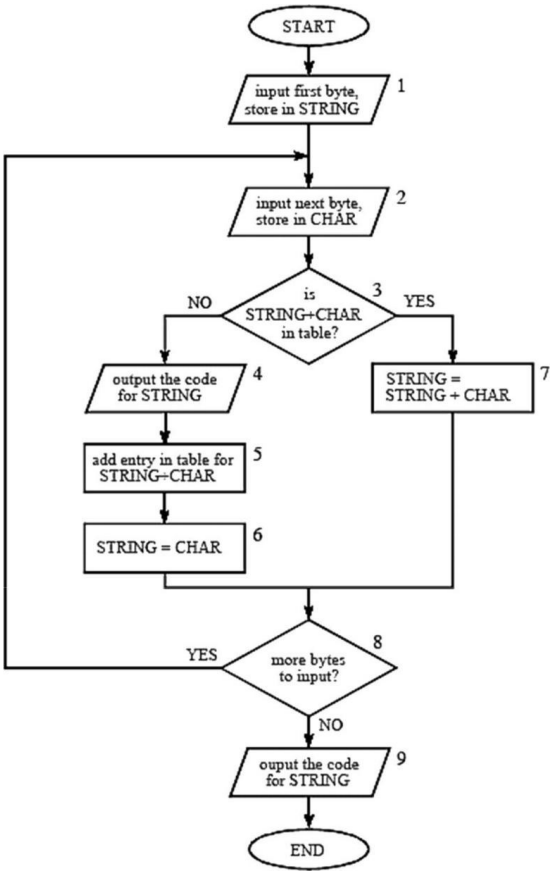

Figure 1: Flow Chart of LZW

Huffman Encoding Algorithm

Huffman Encoding is a fundamental lossless data compression algorithm that assigns variable-length codes to characters based on their frequencies in the input data, with more frequent characters receiving shorter codes. The algorithm constructs a binary tree, known as a Huffman Tree, where each character is represented by a leaf node, and the path from the root to the leaf determines the character's binary code. By prioritizing shorter codes for frequent characters, Huffman Encoding minimizes the overall length of the encoded data, making it highly efficient for compressing files with uneven character frequencies. This algorithm is widely used in formats like JPEG and MPEG, as well as in general-purpose compression utilities, due to its optimal compression capabilities.

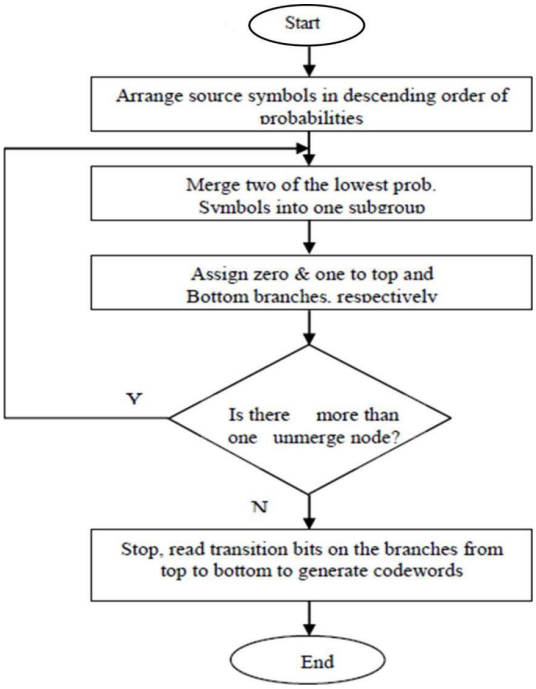

Figure 2: Flow Chart of Huffman

RESULTS AND DISCUSSION

Based on our comparative analysis of the Huffman Encoding and Lempel-Ziv-Welch (LZW) algorithms, we evaluated their performance on 100 files of varying sizes and entropies. The comparison focused on four key metrics: compression ratio, disk space savings, compression time, and decompression time.

Result

The results of our comparative analysis between the Huffman Encoding and Lempel-Ziv-Welch (LZW) algorithms demonstrate that the LZW algorithm outperforms Huffman in multiple aspects, including compression ratio, disk space savings, and both compression and decompression times.

From the analysis of 100 test files with varying sizes and entropies, the following key findings were observed:

Compression Ratio

The LZW algorithm achieved a higher average compression ratio of 2.16, indicating that it compresses data more efficiently than the Huffman algorithm, which had an average compression ratio of 1.7. This suggests that LZW is particularly effective at reducing the size of data, especially in cases with repetitive patterns, making it a more efficient choice for certain types of data compression.

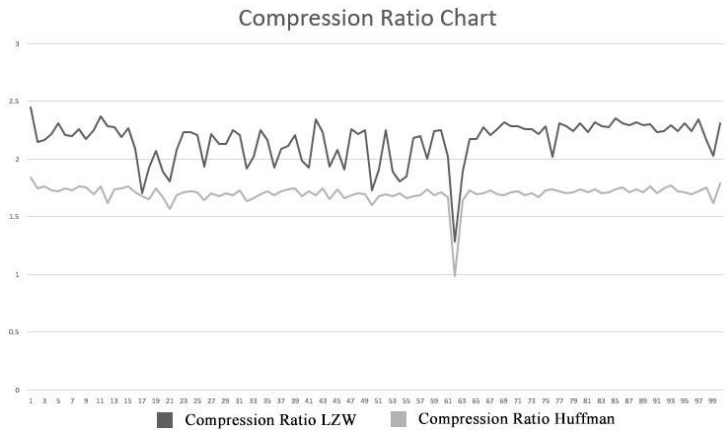

Figure 3: Compression ratio LZW vs Huffman

Compression Time

The LZW algorithm proved to be faster, taking an average of 9.11 seconds to compress files, whereas the Huffman algorithm required significantly more time, averaging 23.97 seconds for compression. This indicates that LZW not only provides better compression efficiency but also performs the compression process more quickly compared to Huffman, making it a more time-efficient choice for data compression tasks.

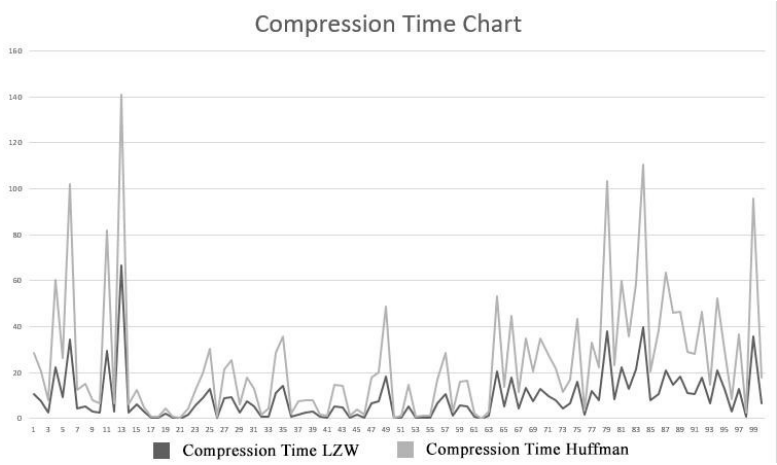

Figure 4: Compression time LZW vs Huffman

Decompression Time

The LZW algorithm demonstrated faster decompression, with an average time of 6.77 seconds, while the Huffman algorithm took longer, averaging 10.62 seconds. This suggests that LZW not only compresses data more efficiently but also decompresses it more quickly than Huffman, making it a more effective choice for applications where speed is a critical factor.

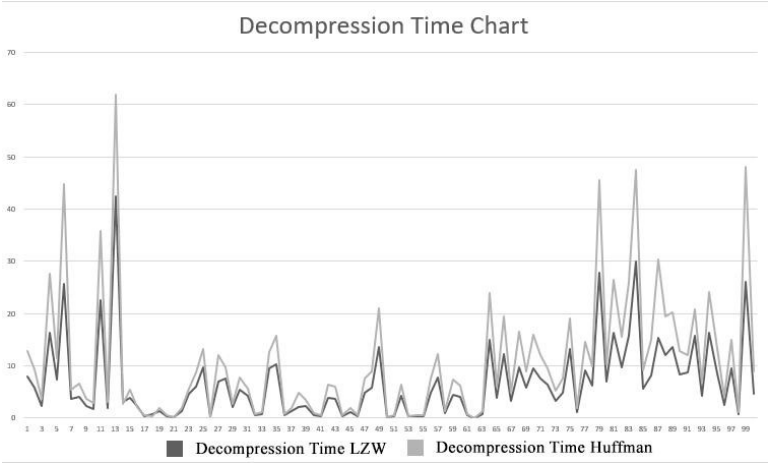

Figure 5: Decompression time LZW vs Huffman

Disk Space Savings

The LZW algorithm achieved greater disk space savings, averaging 53.25%, compared to the Huffman algorithm, which provided an average saving of 40.98%. This indicates that LZW is more effective at reducing file sizes, making it a better choice for maximizing storage efficiency.

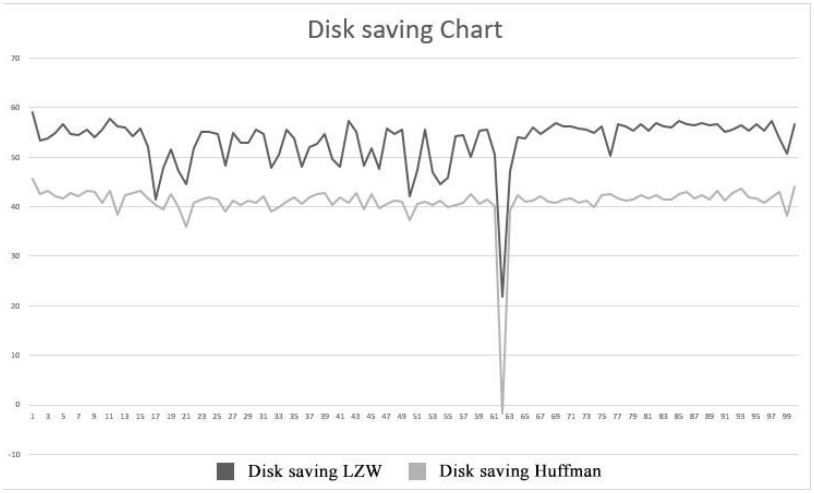

Figure 6: Disk saving LZW vs Huffman

Regression Analysis of Huffman

Table 1: Coefficients

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	4.464	.792		5.638	<.001
	Entropy	-.517	.171	-.260	-3.017	.003
	Original size(KB)	.000	.000	.463	.534	.595
	Compression Time(s)	-.024	.015	-1.405	-1.634	.106
	Decompression Time(s)	.033	.032	1.377	1.055	.294

a. Dependent Variable: Compression Ratio

Regression Analysis of LWZ

In both algorithms, entropy negatively affects the compression ratio, meaning both Huffman and LZW are less effective at compressing data with higher entropy. The effect size and statistical significance are similar for both algorithms, indicating that entropy is a critical factor in both contexts.

Neither Huffman nor LZW shows a significant relationship between the original file size and the compression ratio. This suggests that the size of the file is less important than the content's structure (as indicated by entropy).

Both algorithms show similar, non-significant relationships with compression and decompression times. This might imply that while time is important for performance considerations, it does not directly influence the effectiveness of compression in terms of the ratio achieved. Since entropy is the only significant predictor, choosing between Huffman and LZW might depend more on other factors like processing time or ease of implementation, rather than just compression ratio, especially for data with varying entropy.

When dealing with data with high entropy, consider using other methods to preprocess the data or choose an

Table 2: Coefficients

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	4.464	.792		5.638	<.001
	Entropy	-.517	.171	-.260	-3.017	.003
	Original size(KB)	.000	.000	.463	.534	.595
	Compression Time(s)	-.024	.015	-1.405	-1.634	.106
	Decompression Time(s)	.033	.032	1.377	1.055	.294

a. Dependent Variable: Compression Ratio

algorithm that is better suited for high-entropy data.
Given the similar outcomes, further research could explore additional independent variables or consider non-linear models to see if more complex relationships exist that aren't captured in this linear model.

CONCLUSION

The comparative analysis of Huffman Encoding and Lempel-Ziv-Welch (LZW) algorithms on 100 files of varying sizes and entropies has highlighted the superior performance of the LZW algorithm in several key areas. LZW consistently outperformed Huffman Encoding, achieving a higher compression ratio, greater disk space savings, and faster compression and decompression times. On average, LZW was able to reduce disk space usage by 53.25%, compared to 40.98% with Huffman Encoding. LZW also maintained a better compression ratio of 2.16, while Huffman achieved a ratio of 1.7. Furthermore, LZW demonstrated faster processing times, with average compression and decompression times of 9.11 seconds and 6.77 seconds, respectively, in contrast to Huffman's 23.97 seconds for compression and 10.62 seconds for decompression.
The regression analysis further supports these findings, showing that entropy significantly influences the compression ratio for both algorithms, with higher entropy leading to less effective compression. Neither algorithm displayed a significant relationship between the original file size and compression ratio, indicating that the content's structure, as reflected by entropy, is more critical. Additionally, the analysis revealed no significant impact of compression or decompression times on the compression ratio, suggesting that while time is a factor in performance, it does not directly affect compression efficiency.
These findings suggest that LZW is generally more efficient and effective for lossless data compression, especially in scenarios where time efficiency and storage optimization are critical. However, Huffman Encoding still has its merits and may be preferable in specific use cases depending on the nature of the data. Ultimately, the choice of algorithm should be tailored to the specific requirements of the application at hand. For data with high entropy, other preprocessing methods or compression algorithms may be necessary to achieve optimal results.

REFERENCES

Gopinath, A., & Ravisankar, M. (2020). Comparison of lossless data compression techniques. In *2020 International Conference on Inventive Computation Technologies (ICICT)* (pp. 628–633). https://doi.org/10.1109/ICICT48043.2020.9112516

Holtz, K. (1993). The evolution of lossless data compression techniques. In *Proceedings of WESCON '93* (pp. 140–145). https://doi.org/10.1109/WESCON.1993.488424

Konecki, M., Kudelić, R., & Lovrenčić, A. (2011). Efficiency of lossless data compression. In *2011 Proceedings of the 34th International Convention MIPRO* (pp. 810–815). https://ieeexplore.ieee.org/abstract/document/5967166

Kontoyiannis, I., & Verdú, S. (2014). Optimal Lossless Data Compression: Non-Asymptotics and Asymptotics. *IEEE Transactions on Information Theory, 60*(2), 777–795. IEEE Transactions on Information Theory. https://doi.org/10.1109/TIT.2013.2291007

Nunez, J. L., & Jones, S. (2003). Gbit/s lossless data compression hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 11*(3), 499–510. https://doi.org/10.1109/TVLSI.2003.812288

Patauner, C., Marchioro, A., Bonacini, S., Rehman, A. U., Pribyl, W. (2011). A Lossless Data Compression System for a Real-Time Application in HEP Data Acquisition. *IEEE Transactions on Nuclear Science, 58*(4), 1738–1744. https://doi.org/10.1109/TNS.2011.2142193

Patel, R. A., Zhang, Y., Mak, J., Davidson, A., & Owens, J. D. (2012). Parallel lossless data compression on the GPU. In *2012 Innovative Parallel Computing (InPar)* (pp. 1–9). https://doi.org/10.1109/InPar.2012.6339599

Sharma, K., & Gupta, K. (2017). Lossless data compression techniques and their performance. In *2017 International Conference on Computing, Communication and Automation (ICCCA)* (pp. 256–261). https://doi.org/10.1109/CCAA.2017.8229810