FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 9 No. 4, April, 2025, pp 99 - 105
DOI: https://doi.org/10.33003/fjs-2025-0904-3555

PERFORMANCE COMPARISON OF RUN-LENGTH, HUFFMAN AND LEMPLE-ZIV ALGORITHMS ON GRAY-SCALE PNG AND JPG IMAGES COMPRESSION

[1]Okude Joshua Okude, [1]Emmanuel Ogala and *[2]Terseer Andrew Gaav

[1]Department of Computer Science, College of Physical Sciences, Joseph Sarwuan Tarta University, Makurdi.
[2]Department of Computer Science. Faculty of Computing, Federal University of Lafia, Lafia

*Corresponding authors' email: gaavterseer@gmail.com

ABSTRACT

Image compression plays a crucial role in optimising storage and transmission efficiency. This paper evaluates the performance of Run-Length Encoding (RLE), Huffman Coding, and Lempel-Ziv-Welch (LZW) algorithms for compressing grayscale PNG and JPG images. The study analyses their effectiveness using compression ratio, bits per pixel, and compression time as key performance metrics. Results indicate that LZW achieved the highest compression ratio, ranging from 1.0113 to 2.4020, making it the most efficient for file size reduction. RLE performed moderately, with compression ratios between 0.5456 and 2.3895, while Huffman Coding exhibited the lowest ratios, ranging from 0.2646 to 1.0680. In terms of bits per pixel, LZW recorded the lowest values, highlighting its ability to reduce data while preserving image quality. Compression time analysis revealed that RLE was the fastest, with processing times between 0.0019 and 0.0468 seconds, making it suitable for real-time applications. LZW and Huffman Coding demonstrated a trade-off between compression efficiency and speed. These findings establish LZW as the most effective algorithm for high compression with minimal quality loss, while RLE remains the best option for speed-critical applications.

Keywords: Image Compression, Run-Length Encoding (RLE), Huffman Coding, Lempel-Ziv-Welch (LZW), Compression Efficiency

INTRODUCTION

Image compression is a widely used technique to reduce file sizes for storage and processing, particularly as image quality and resolution continue to increase. With the growing reliance on cloud storage, compression plays a crucial role in efficiently managing large volumes of images online (Kodukulla, 2020).

Generally, data compression involves transforming files such as text, audio, and video into a more compact form, allowing for full recovery without loss of information. This is essential for optimising storage space and reducing resource requirements for data transmission. Compression techniques balance space and time complexity, requiring trade-offs between compression level, distortion, and computational cost for encoding and decoding data (Sharma, 2020).

Image compression is broadly classified into lossy and lossless techniques (Vemuri et al., 2014). Lossless methods preserve all original data, while lossy methods discard some information to achieve higher compression ratios (Mathpal & Mehta, 2017). Among lossless techniques, Run-Length Encoding, Huffman Coding, and Lempel-Ziv Coding are commonly used. These methods are evaluated based on metrics such as compression ratio, compression time, and Peak Signal-to-Noise Ratio (PSNR) (Sara et al., 2019). Huffman coding, for example, assigns shorter codes to frequently occurring symbols and longer codes to less frequent ones, constructing a binary tree for efficient encoding.

As image data continues to grow exponentially in various domains such as healthcare, security, and multimedia, the need for efficient compression techniques becomes critical for optimising storage, reducing transmission time, and improving computational efficiency (Ungureanu et al., 2024). Although lossy compression methods can achieve higher compression rates, they are unsuitable for applications requiring exact data preservation, such as medical imaging and forensic analysis (Bourai et al., 2024). This highlights the importance of lossless compression techniques in ensuring data integrity while maintaining reasonable compression efficiency.

Lossless compression algorithms such as Run-Length, Huffman, and Lempel-Ziv are widely used for image data storage and transmission due to their ability to retain original data without loss (Fitriya et al., 2017), However, their efficiency varies across different image formats, influencing compression ratios and processing speeds (Agber et al., 2024). Given that gray-scale PNG and JPG images are extensively used in critical applications such as medical imaging and biometric security, where detail preservation is essential (Akhtarkavan et al., 2023), evaluating the performance of these algorithms on such images is crucial. Agber et al. (2024) highlights that while the Lempel-Ziv algorithm is suitable for JPEG, PNG, and BMP formats, it is not as efficient as Huffman compression, and further research is needed to compare lossless algorithms for optimising image quality in both original and compressed formats. Understanding how these algorithms perform in terms of compression ratio, speed, and reconstructed image quality is essential for selecting the most suitable approach for various applications requiring high data fidelity.

Recent advancements in image compression have incorporated artificial intelligence and statistical methods for improved performance. (Lu et al., 2018) demonstrated the potential of convolutional neural networks (CNNs) in lossy compression, surpassing traditional techniques such as JPEG2000. Similarly, Li and Ji (2020) introduced NICE, a framework combining neural explanation with semantic image compression, achieving higher compression rates while preserving classification accuracy. Archana and Jeevaraj (2024) reviewed image processing techniques, comparing traditional and deep learning approaches. They highlighted the strengths and challenges of methods like denoising, segmentation, and classification. The study emphasised deep learning's effectiveness while addressing computational and interpretability concerns.

Peng et al. (2023) applied transfer learning to reduce training times while maintaining performance across datasets. Divya et al. (2020) explored lossless text document compression to optimise storage and transmission efficiency. They discussed Huffman, Run-Length, and Lempel-Ziv Welch encoding techniques, highlighting their distinct approaches to reducing data size. The study emphasised imprsoved security and compression management through encoding and decoding processes, while Joshi et al. (2025) explored hybrid compression techniques for medical images, optimising PSNR and compression ratio. They highlighted deep learning-based autoencoders for reducing data size while preserving diagnostic integrity. Ma (2023) compared Huffman and LZW algorithms for image compression, evaluating Compression Ratio (CR), Mean Square Error (MSE), Peak Signal to Noise Ratio (PSNR), and Bits Per Pixel (BPP). The study found Huffman achieved better CR for small images (≤300KB), while LZW provided higher image quality. Future research may explore larger images and processing time. Additionally, Peng et al. (2023) leveraged large language models to enhance perceptual similarity metrics. Singh et al. (2024) incorporated structural similarity and PSNR evaluations for a more comprehensive assessment of compression techniques.
This paper systematically compares the performance of Run-Length, Huffman, and Lempel-Ziv algorithms in compressing gray-scale PNG and JPG images, assessing their effectiveness in maintaining image quality.

MATERIALS AND METHODS

Data Collection

The dataset used in this study comprises grayscale PNG and JPG images, primarily featuring logos sourced from publicly available repositories. The image search tool adopted in this paper is Content based image retrieval (CBIR). The choice of CBIR is due to its ability to extract visual content of an image, like colour, texture, and shape automatically which enhances the compression techniques accuracy with lower computational time (Ibrahim & Ahmad, 2019). The Logos were selected due to their distinct shapes, simplicity, and high contrast, making them well-suited for assessing compression algorithms in applications where image clarity is essential. The dataset was predominantly logos of varying sizes, resolutions, and complexities, enabling a thorough evaluation of the algorithms. This selection ensures that the study's findings are applicable to real-world scenarios, such as web design, branding, and digital media, where efficient logo compression is necessary.

Traditional Lossless Image Compression Algorithms

Lossless compression algorithms reduce data size without losing any original information, which is crucial in fields where data accuracy and fidelity are paramount. There are basically three lossless image compression algorithms as presented below.

Run-Length Encoding (RLE)

Run-Length Encoding (RLE) is a compression technique that reduces file size by replacing sequences of repeated elements with a single value followed by the count of occurrences. For instance, the sequence "AAAAAA" is encoded as "6A," making it more compact than the original. This method is highly effective for data with frequent repetition, such as binary images and certain text format (Ma, 2023).

Pseudocode for Run-Length Algorithm

```
RLE(input)
   initialize encoded_output as empty
   count = 1
   for i = 1 to length(input) - 1
      if input[i] == input[i - 1]
         count = count + 1
      else
         append (input[i-1], count) to encoded_output
         count = 1
   end for
   append (input[length(input)-1], count) to encoded_output
   return encoded_output
```

Illustration of Run-Length Algorithm

For the string "AAABBBCCCDAA," RLE produces the output A3B3C3D1A2, reducing storage, especially effective in areas like fax transmission and monochrome image (Nitu et al., 2019).

Table 1: Compression Efficiency Table

Original Data	Compressed Data	Compression Ratio
AAAABBBCCCCDD	A4B3C4D2	2.5:1
ABCD	A1B1C1D1	1:1

RLE is suitable where data has long runs, but less effective for files with high entropy (Smith et al., 2021).

Huffman Coding

Huffman Coding is a lossless compression technique that assigns variable-length binary codes to symbols based on their frequency. More frequent symbols receive shorter codes, while less common symbols are assigned longer ones, minimising the average bit length required for storage (Nitu et al., 2019). By constructing a binary tree, Huffman coding effectively reduces redundancy, making it suitable for text and image compression (Fauzan et al., 2022). The process starts with building a frequency table, followed by the creation of a binary tree where lower-frequency symbols are placed deeper. The efficiency of Huffman coding depends on the distribution of symbol frequencies (Sujatha & Selvam, 2022).

Huffman Coding Process

The algorithm begins by creating a frequency table, constructing a tree, and then assigning binary codes based on the tree paths. The compression efficiency depends on symbol frequency (Archana & Jeevaraj, 2024).

Formula for Average Bit Length

$$L = \sum_{i=1}^{n} (\times\ l_i) \quad (1)$$

Where f_i the frequency of each is character, and l_i is the bit length of the Huffman code. This approach has been widely applied in file compression formats like JPEG and MP3 Ma (2023).

Lempel-Ziv-Welch (LZW)

Lempel-Ziv-Welch (LZW) is a dictionary-based compression algorithm that builds a table of repeating sequences as it processes data. Initially, each symbol has its own dictionary entry. As new patterns emerge, they are assigned unique codes, allowing for efficient encoding of repeated sequences (Nitu et al., 2019). This adaptability enables LZW to achieve high compression rates without requiring prior knowledge of the data's characteristics. It is widely used in formats such as GIF and ZIP file compression (Das et al., 2024).

LZW Algorithm Pseudocode

```
LZW(input)
  initialize dictionary with unique symbols in input
  w = empty string
  for each character k in input
    if w + k exists in dictionary
      w = w + k
    else
      add (w + k) to dictionary
      output code for w
      w = k
  end for
  output code for w
```

Table 2: Dictionary Growth Example for "ABABABA"

Step	Sequence Processed	Dictionary State	Output Code
1	A	{A: 1, B: 2, AB: 3}	1
2	B	{A: 1, B: 2, AB: 3, BA: 4}	2
3	AB	{A: 1, B: 2, AB: 3, BA: 4}	3

LZW is commonly used in GIF compression, providing a balance between speed and efficiency, especially for images with repeating patterns (Sujatha & Selvam, 2022).

Performance Metrics

Performance metrics are key to assessing the efficiency and effectiveness of data compression algorithms. These metrics quantify how well an algorithm performs in terms of compression efficiency, speed, quality retention, and resource consumption. By understanding these metrics, one can optimize algorithms based on the specific requirements of different applications, such as real-time systems, archival purposes, or media streaming.

Compression Ratio (CR)

The compression ratio is one of the most basic and widely used performance metrics. It is defined as the ratio of the original data size to the compressed data size. A higher compression ratio indicates more effective compression. The formula is given by:

$$\text{Compression Ratio (CR)} = \frac{\text{Original Size}}{\text{Compressed Size}} \qquad (2)$$

Compression Speed (CS)

This refers to the amount of time taken to compress the data. This metric is particularly important in scenarios where time constraints are critical, such as video conferencing or real-time media streaming. Faster compression speeds are essential to ensure smooth operation, especially in systems that handle large volumes of data. The formula to calculate compression speed is:

$$\text{Compression Speed (CS)} = \frac{\text{Size of Data}}{\text{Time Taken to Compress}} \qquad (3)$$

For example, if a 1 GB file is compressed in 2 seconds, the compression speed would be 0.5 GB per second. The speed of compression is often inversely related to the compression ratio, with algorithms that offer higher compression ratios tending to take more time.

Decompression speed (DS)

This refers to the time required to restore the compressed data to its original form. This metric is especially important in real-time applications where rapid data retrieval is essential. Efficient decompression ensures that the system can quickly serve the decompressed data without causing delays in playback or processing. The formula for decompression speed is similar to compression speed:

$$\text{Decompression speed (DS)} = \frac{\text{Size of Data}}{\text{Time Taken to Decompress}} \qquad (4)$$

As with compression speed, decompression speed should be optimized for applications like cloud storage and media streaming where data must be readily accessible (Sujatha & Selvam, 2022).

Bits per Pixel (BPP)

This Represents the average number of bits required to store each pixel in a compressed image. It provides insights into how compactly image data is stored, with lower values indicating better compression. It is given as:

$$\text{BPP} = \frac{\text{Size of Compressed Image (in bits)}}{\text{Time Number of pixels in Image}} \qquad (5)$$

RESULTS AND DISCUSSION

Results of the Utilised Algorithms

This section presents and discusses the results of the utilized algorithms (RLE, LZW and Huffman) using various metrics.

Table 3: Results of Run-length Encoding (RLE)

Image Name	Original Size (KB)	Num Characters	Compressed Size (KB)	Compression Ratio	Compression Time (s)	Bits/Pixel	Decompression Time (s)
A_Man .jpg	129.40	220,314	121.88	1.0617	0.0412	4.5319	0.0852
android_logo.png	002.09	016,384	001.32	1.5895	0.0020	0.6577	0.0036
bird.jpg	133.71	218,988	162.94	0.8206	0.0468	6.0953	0.0962
facebook_logo.png	002.00	016,384	000.83	2.3895	0.0019	0.4175	0.0034
location_logo .png	002.32	016,384	001.06	2.1917	0.0253	0.5298	0.0377
Radio.png	006.22	050,625	011.39	0.5456	0.0452	1.8435	0.0659
search.png	002.00	050,625	001.63	1.2237	0.0061	0.2642	0.0107
twitter_Icon.png	002.23	016,384	001.02	2.1990	0.0020	0.5078	0.0035
wifi_Icon.png	002.38	016,384	001.17	2.0350	0.0020	0.5859	0.0035
Zebra.png	002.45	014,568	001.24	0.9368	0.0019	0.6577	0.0206
AVERAGE	028.48	063,704	030.448	1.4993	0.0174	1.6091	0.0330

The result in table 3 indicate that Run-Length Encoding (RLE) achieved compression ratios average of 1.4992, performing better on smaller images. Its compression time average of 0.0174 seconds, demonstrating fast processing, especially for simple images. However, bits per pixel values varied from 0.2642 to 6.0953, highlighting inconsistent data retention. This suggests that while RLE is effective for images with large uniform areas, it is less suitable for complex images with diverse patterns.

Table 4: Results of Lempel-Ziv-Welch (LZW)

Image Name	Original Size (KB)	Num Characters	Compressed Size (KB)	Compression Ratio	Compression Time (s)	Bits/Pixel	Decompression Time (s)
A_Man .jpg	129.40	220,314	53.87	2.4020	0.0964	2.0032	0.0964
android_logo.png	002.09	016,384	01.27	1.6482	0.0055	0.6343	0.0055
bird.jpg	133.71	218,988	78.74	1.6982	0.1095	2.9454	0.1095
facebook_logo.png	002.00	016,384	00.85	2.3510	0.0053	0.4243	0.0053
location_logo .png	002.32	016,384	01.11	2.0878	0.0293	0.5562	0.0293
Radio.png	006.22	050,625	06.15	1.0113	0.0356	0.9946	0.0356
search.png	002.00	050,625	01.69	1.1793	0.0170	0.2742	0.0170
twitter_Icon.png	002.23	016,384	01.09	2.0420	0.0054	0.5469	0.0054
wifi_Icon.png	002.38	016,384	01.22	1.9474	0.0054	0.6123	0.0054
Zebra.png	002.43	015,345	01.43	1.3940	0.0065	0.5981	0.0245
AVERAGE	28.478	063,782	14.742	1.7761	0.0316	0.9590	0.0334

The result in table 4 reveals that, the Lempel-Ziv-Welch (LZW) algorithm achieved compression ratios average of 1.7761, demonstrating its effectiveness in reducing file sizes across different image types. Compression and decompression times were stable, ranging from 0.0055 to 0.1095 seconds, indicating efficient processing. With bits per pixel values between 0.2742 and 2.9454, LZW maintains a balance between compression efficiency and image quality, making it suitable for applications requiring both data reduction and detail preservation.

Table 5: Results of Huffman Coding Algorithm

Image Name	Original Size (KB)	Num Characters	Compressed Size (KB)	Compression Ratio	Compression Time (s)	Bits/Pixel	Decompression Time (s)
A_Man .jpg	129.40	220,314	121.17	1.0680	0.0632	4.5054	0.0632
android_logo.png	002.09	016,384	003.84	0.5442	0.0051	1.9210	0.0051
bird.jpg	133.71	218,988	182.28	0.7335	0.0606	6.8186	0.0606
facebook_logo.png	002.00	016,384	003.11	0.6406	0.0047	1.5576	0.0047
location_logo .png	002.32	016,384	003.56	0.6526	0.0253	1.7795	0.0253
Radio.png	006.22	050,625	017.93	0.3466	0.0134	2.9019	0.0135
search.png	002.00	050,625	007.55	0.2646	0.0131	1.2220	0.0131
twitter_Icon.png	002.23	016,384	003.59	0.6218	0.0054	1.7963	0.0054
wifi_Icon.png	002.38	016,384	003.54	0.6746	0.0051	1.7676	0.0051
Zebra.png	003.47	015,478	004.23	0.6418	0.0136	1.5467	0.0049
AVERAGE	028.582	063,795	035.08	0.6188	0.0210	2.5817	0.0201

In table 5 above, the Huffman Coding algorithm achieved compression ratios average of 0.6188, performing more efficiently on smaller images. Compression times ranged from 0.0047 to 0.0632 seconds, indicating relatively fast processing. With bits per pixel values average of 2.5817, the algorithm balances compression efficiency with image detail retention. This makes Huffman Coding a suitable choice for reducing file sizes while preserving essential image details.

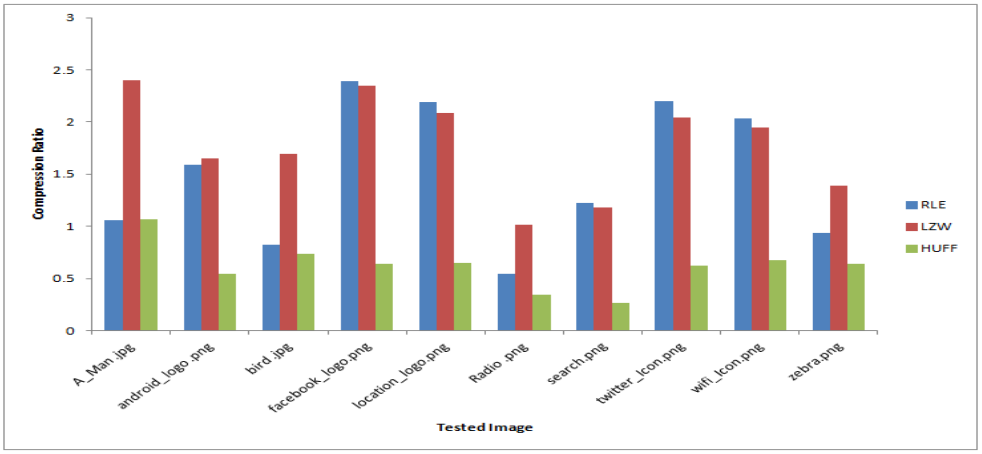

Figure 1: Comparison of the employed Methods (RLE, LZW and Huffman) Based on Compression Ratio

The comparison of compression methods based on compression ratio in figure 1 shows that LZW performed the best, with values ranging from 1.0113 to 2.4020, demonstrating its strong ability to reduce file sizes. RLE achieved moderate compression ratios, particularly for simpler images, with values between 0.5456 and 2.3895. Huffman Coding had the lowest compression ratios, ranging from 0.2646 to 1.0680, making it the least efficient among the three. The results highlight LZW as the most effective method for achieving higher compression ratios.

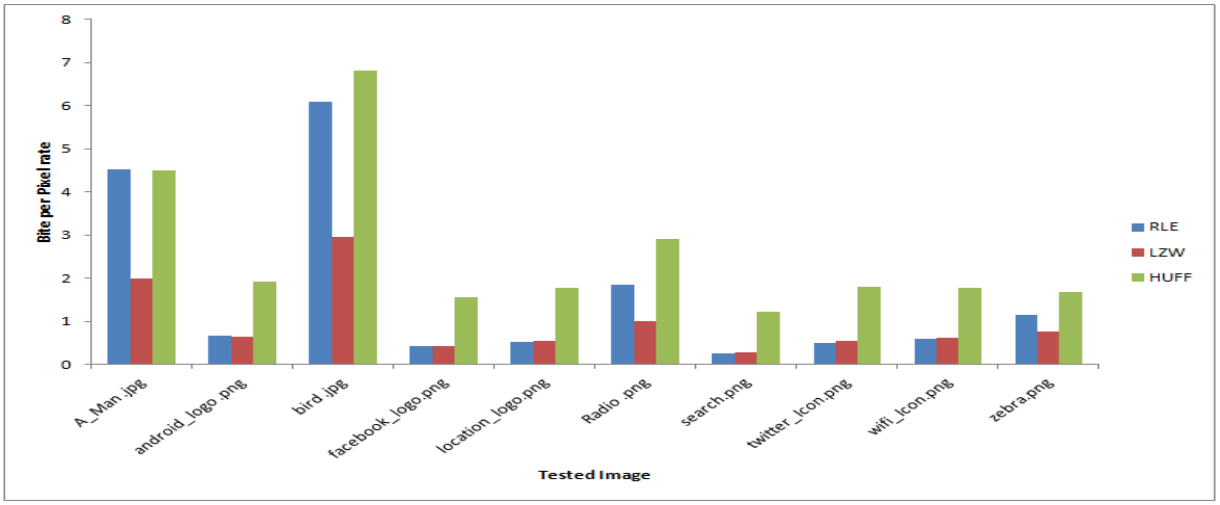

Figure 2: Compression of Compression Methods Based On images in bit/pixel

The comparison of compression methods based on bits per pixel in figure 2 highlights notable differences in performance. LZW achieved the lowest bits per pixel values, ranging from 0.2742 to 2.9454, indicating efficient compression while preserving image detail. RLE had higher values, between 0.2642 and 6.0953, reflecting its lower efficiency in maintaining image quality. Huffman Coding recorded the highest bits per pixel values, from 1.2220 to 6.8186, suggesting it retains more detail despite compressing the image. These findings confirm LZW as the most effective method for balancing data reduction and image quality.

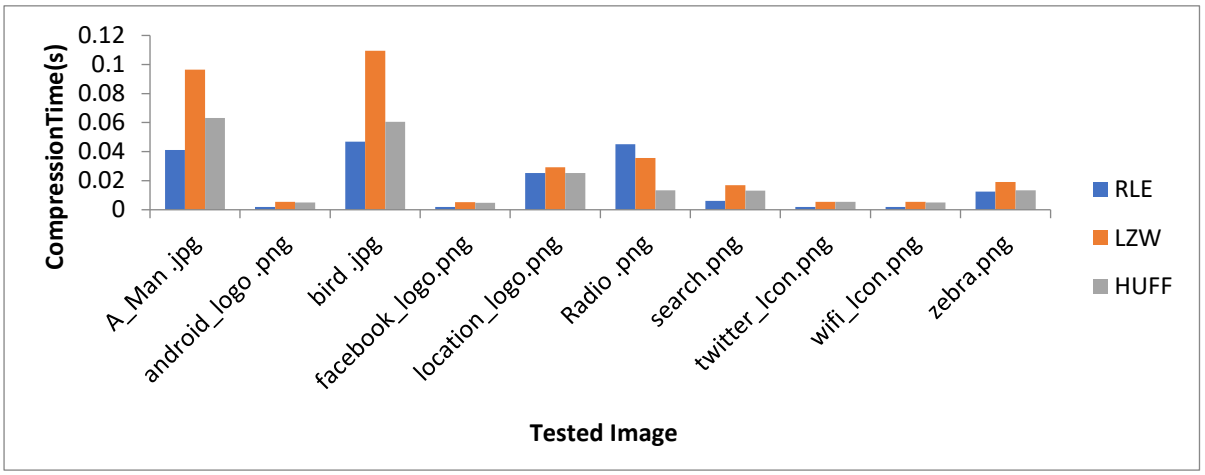

Figure 3: Compression of Compression Methods Based on Images Compression Time (s)

The comparison of image compression methods based on compression time in figure 4 reveals notable differences in speed. RLE was the fastest, with compression times ranging from 0.0019 to 0.0468 seconds, making it ideal for applications requiring quick processing. LZW had longer compression times, between 0.0053 and 0.1095 seconds, reflecting a trade-off between efficiency and speed. Huffman Coding fell in between, with times ranging from 0.0047 to 0.0632 seconds, offering a balanced performance. These findings highlight RLE's advantage in speed-critical applications, while LZW and Huffman may be preferable when compression quality is a priority.

Comparative Analysis with Existing Works

In this section, the results of the compression size and compression ratio obtained in this paper is compared with the results in the existing works.

Table 6: Comparison of the Results of the Adopted Run-length Encoding (RLE) with the Result of the Existing Works

S/N	Authors	Compression Size			Compression Ratio		
		RLE	LZW	Huffman	RLE	LZW	Huffman
1	Agber et al. (2024)	3,986	8,566	6,120	0.0180	0.3767	0.0656
2	Azeez & Lasisi (2017)	1,198,026	344,264	293,486	770.043	74.6742	119.5180
3	Alarabeyyat et al. (2012)	-	-	-	1.1561	1.4451	1.1603
4	Adopted Method	30.448	14.7420	35.0800	1.4993	1.7761	0.6188

From the result in table 6, Agber et al (2024) revealed that, RLE algorithm gives a smaller image compression size than other algorithms. This result contradicts the outcomes in the work of Azeez and Lasisi (2017) which revealed Huffman as the algorithm that gives smaller image compression size. The results from the existing works did not tally with the result from the adopted method which revealed LZW with the ability to give smaller image compression size. The differences in the results of both the existing works and the method adopted could be as a result of the differences in the nature of the image datasets used by different authors. Similarly, in terms of the Compression ratio, while Agber et al (2024) revealed RLE with smaller compression ratio and LZW with larger compression ratio, Azeez and Lasisi (2017) revealed LZW with smaller and RLE with larger compression ratio. This variation in their result could be associated with diverse datasets utilized by different authors. However, this paper and Alarabeyyat et al. (2012) adopted small size images therefore, both revealed Huffam coding and LZW with smaller and larger compression ratio respectively.

CONCLUSION

This paper systematically compared the performance of Run-Length Encoding (RLE), Huffman Coding, and Lempel-Ziv-Welch (LZW) algorithms in compressing grayscale PNG and JPG images. The findings revealed that LZW consistently outperformed the other methods in compression ratio and bits per pixel, making it the most efficient in reducing file size while maintaining image quality. RLE demonstrated the fastest compression times, making it highly suitable for real-time applications, but its effectiveness was limited to images with large areas of uniform colour. Huffman Coding, though maintaining a balance between efficiency and detail preservation, had the lowest compression ratios, making it less suitable for applications requiring significant file size reduction. Overall, the results highlight LZW as the best choice for achieving high compression efficiency with minimal quality loss, while RLE is preferred for scenarios requiring rapid processing. By comparison with existing works, the choice of the algorithm depend on the type and nature of images under consideration. Future research could explore hybrid approaches that integrate the strengths of these algorithms to enhance compression efficiency across various image types.

REFERENCES

Agber, S., Isah Odoh, S., Gideon Atabo, O., Rufina Godwin, I., Piyinkir Ndahi, B., & Akumba, B. O. (2024). Efficiency Evaluation of Huffman, Lempel-Ziv, And Run-Length Algorithms in Lossless Image Compression for Optimizing Storage and Transmission Efficiency. *Article in International Journal of Computer Applications*, *186*(37), 975–8887. https://doi.org/10.5120/ijca2024923933

Akhtarkavan, E., Majidi, B., & Mandegari, A. (2023). Secure Medical Image Communication Using Fragile Data Hiding Based on Discrete Wavelet Transform and A Lattice Vector Quantization. *IEEE Access*, *11*, 9701–9715. https://doi.org/10.1109/ACCESS.2023.3238575

Alarabeyyat, A., Khdour, T., & Btoush, M. H. (2012). *Lossless Image Compression Technique Using Combination Methods*. *January*. https://doi.org/10.4236/jsea.2012.510088

Archana, R., & Jeevaraj, P. S. E. (2024). Deep learning models for digital image processing: a review. In *Artificial Intelligence Review* (Vol. 57, Issue 1). Springer Netherlands. https://doi.org/10.1007/s10462-023-10631-z

Azeez, N. A., & Lasisi, A. A. (2017). Empirical and Statistical Evaluation of the Effectiveness of Four Lossless Data Compression Algorithms. *Nigerian Journal of Technological Development*, *13*(2), 64. https://doi.org/10.4314/njtd.v13i2.4

Bourai, N. E. H., Merouani, H. F., & Djebbar, A. (2024). Deep learning-assisted medical image compression challenges and opportunities: systematic review. In *Neural Computing and Applications* (Vol. 36, Issue 17). Springer London. https://doi.org/10.1007/s00521-024-09660-8

Das, D., Guha, S., Brubaker, J., & Semaan, B. (2024). The "Colonial Impulse" of Natural Language Processing: An Audit of Bengali Sentiment Analysis Tools and Their Identity-based Biases. *Conference on Human Factors in Computing Systems - Proceedings*, 1–18. https://doi.org/10.1145/3613904.3642669

Divya, M. S., Chandrashekhara, J., Vinay, S., & Ramadevi, A. (2020). *Lossless Compression for Text Document Using Huffman Encoding , Run Length Encoding , and Lempel-Ziv Welch Coding Algorithms*. *9*(3), 100–105.

Fauzan, M. N., Alif, M., & Prianto3, C. (2022). Comparison of Huffman Algorithm and Lempel Ziv Welch Algorithm in Text File Compression. *IT Journal Research and Development*, *7*(2), 155–169. https://doi.org/10.25299/itjrd.2023.10437

Fitriya, L. A., Purboyo, T. W., & Prasasti, A. L. (2017). A review of data compression techniques. *International Journal of Applied Engineering Research*, *12*(19), 8956–8963.

Ibrahim, Maryam Lawal, Ahmad, M. A. (2019). AN ENHANCED RGB AN ENHANCED RGB PROJECTION ALGORITHM FOR. *FUDMA Journal of Sciences (FJS), Vol. 3 No.*(March), pp 280 – 285.

Joshi, B., Vaseer, G., Science, C., No, G., Rd, S., Range, R., Science, C., No, G., Rd, S., & Range, R. (2025). *Tailoring Image Compression Algorithms for Optimal PSNR and Compression Ratio in Medical Diagnostic Imaging*. *54*(1), 1800–1811.

Kodukulla, S. T. (2020). Lossless Image compression using MATLAB. *Bachelor Thesis Electrical Engineering June 2020 Bachelor*, *June*.

Li, X., & Ji, S. (2020). Neural Image Compression and Explanation. *IEEE Access*, *8*, 214605–214615. https://doi.org/10.1109/ACCESS.2020.3041416

Lu, T., Liu, Q., He, X., Luo, H., Suchyta, E., Choi, J., Podhorszki, N., Klasky, S., Wolf, M., Liu, T., & Qiao, Z. (2018). Understanding and modeling lossy compression schemes on HPC scientific data. *Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium, IPDPS 2018*, *1*, 348–357. https://doi.org/10.1109/IPDPS.2018.00044

Ma, S. (2023). Comparison of image compression techniques using Huffman and Lempel-Ziv-Welch algorithms. *Applied and Computational Engineering*, *5*(1), 793–801. https://doi.org/10.54254/2755-2721/5/20230705

Mathpal Mittal Darji Assistant Professor Assistant Professor, D., & Mehta Assistant Professor, S. (2017). A Research Paper on Lossless Data Compression Techniques. *IJIRST-International Journal for Innovative Research in Science & Technology*|, *4*(1), 190–194. www.ijirst.org

Nitu, Kumar, Y., & Rishi, R. (2019). Fractal Image Compression Techniques. *International Journal of Computer Sciences and Engineering*, *7*(1), 229–233. https://doi.org/10.26438/ijcse/v7i1.229233

Peng, X., Zhang, Y., Peng, D., & Zhu, J. (2023). *Selective Run-Length Encoding*. http://arxiv.org/abs/2312.17024

Sara, U., Akter, M., & Uddin, M. S. (2019). Image Quality Assessment through FSIM, SSIM, MSE and PSNR—A Comparative Study. *Journal of Computer and Communications*, *07*(03), 8–18. https://doi.org/10.4236/jcc.2019.73002

Sharma, G. (2020). Analysis of Huffman Coding and Lempel-Ziv-Welch (LZW) Coding as Data Compression Techniques. *International Journal of Scientific Research in Research Paper. Computer Science and Engineering*, *8*(1), 37–44. www.isroset.org

Sujatha, T., & Selvam, K. (2022). Lossless Image Compression Using Different Encoding Algorithm for Various Medical Images. *ICTACT Journal on Image and Video Processing*, *12*(4), 2704–2709. https://doi.org/10.21917/ijivp.2022.0384

Ungureanu, V. I., Negirla, P., & Korodi, A. (2024). Image-Compression Techniques: Classical and "Region-of-Interest-Based" Approaches Presented in Recent Papers. *Sensors*, *24*(3). https://doi.org/10.3390/s24030791

Vemuri, B.C., Sahni, S., Kapoor, C., Leonard, C. and Fitzsimmons, J. (2014). Lossless Image Compression. *The Essential Guide to Image Processing*, *March*, 385–419. https://doi.org/10.1016/B978-0-12-374457-9.00016-0