**Report on Adaptive Lossless Encoder and Decoder for Images Using Huffman Encoding**

**Table of Contents**

[1. Introduction 3](#_Toc183983833)

[2. Problem Statement and Objectives 3](#_Toc183983834)

[3. Methods and Algorithms 4](#_Toc183983835)

[4. Implementation 5](#_Toc183983836)

[5. Results and Discussion 7](#_Toc183983837)

[6. Conclusion and Future Work 8](#_Toc183983838)

[References 10](#_Toc183983839)

# **1. Introduction**

To the present generation, where data forms the nucleus of every product, the management of data dictates the management of large volumes of such data. This is especially true for media like images, but for which the use of current technologies leads to great amounts of storage space needed. Lossless data compression is widely used in this domain in order to compress given data and still be able to reconstruct the data to perfection. Then, there exist few lossless data compression techniques, which are based on Huffman Encoding technique for instance, this technique assigns different code lengths which are relevant to the frequency of use of the different symbols.

This report presents the work done on an adaptive lossless image coder and decoder based on Huffman Encoding. The aim is to compress the image files and compress them to a greater level yet maintaining the ability to uncompressed the image and reconstruct the original image for analysis of the performances of the algorithm; these include the compression factor, the encoding/decoding time (Al-okaily & Tbakhi, 2020).

# **2. Problem Statement and Objectives**

**Problem Statement**

As digital images are gaining importance in the field of graphics and communication applications the concern of archiving large digital image with minimal reduction in picture quality has emerged as a major challenge. Traditional methods of compression are either slightly distorting the images or do not effectively deal with the large databases. This project concerns itself with a dynamic lossless compression algorithm that adapts to data characteristics in the context of image data. Huffman Encoding is the chosen method for this particular task as it is a classic lossless data compression algorithm (Al-okaily & Tbakhi, 2020).

**Objectives**

The main objectives of the project are

Using Huffman Encoding, to create an image compression algorithm with optimized size that does not lose any data from the image.

To count the number of frames in an input video that must be processed as an indicator of algorithm’s performance for encoding and decoding time.

To measure the effectiveness of the compression, the compression ratio is calculated, it is also used to compare the original image size with the encoded image size.

To check the validity of the algorithm to decompress the image to its original state so as to achieve actual loss-less data compression (Arif, Ali, Rehman, & Alamzeb, 2023).

# **3. Methods and Algorithms**

**Huffman Encoding**

Huffman Encoding, is one of the most commonly used algorithm implemented for loss less data compression. This is done for the reason that it assigns brief codes to highly representative symbols – and long strings of code to less representative symbols, in this case and in Huffman coding too, the symbols are pixel values (Arif, Ali, Rehman, & Alamzeb, 2023). The basic process involves:

**Frequency Analysis:** Calculate the number of times each value of pixel was used in the image.

**Building a Binary Tree:** These frequencies are used to create a binary tree. The nodes of the tree are the sum of the frequencies of the pixels from the corresponding child nodes where the nodes in a tree represent the pixel values (Hughes, 2023).

**Code Generation:** The method operates from the root; with binary codes obtained by moving down the tree to label each pixel value. A symbol is approached by a binary code with path connected to this symbol equal to zero if the branch is left and equal to one if the branch is right.

**Compression:** With Huffman codes, the pixel values are exchanged with the Huffman codes to give a binary string with lesser number of characters.

**Decompression:** For decoding, the string of binary is read, and going through the Huffman tree decides the actual pixel values through operation on the string as per Huffman tree (Hughes, 2023).

**Adaptive Huffman Encoding**

In contrast to the Huffman Encoding, which is based on the static calculating of symbols frequencies in advance, the adaptive Huffman Encoding optimizes the structure of the tree in the process of its use. The algorithm changes the tree as the image is processed one pixel at a time to mirror new pixel frequencies of the symbols. This adaptation enhances compression on image that has disparate feature (Hughes, 2023).

**Compression and Decompression**

In this project

Encoding includes the process of making the image two-dimensional, constructing the Huffman tree from the compressed image data and creating the compressed bitstream form of the image.

Decoding includes reconstructing the Huffman tree normally stored or transmitted in conjunction with the compressed image data then viewing the binary stream and using the tree to reconstruct the original pixel values. The obtained pixel array is then reshaped into the image (Hughes, 2023).

# **4. Implementation**

That is why Python with the help of the libraries as NumPy for numerical calculations and Pillow (PIL) for images was used for implementation of the project. The implementation revolves around Huffman Encoding algorithm, which gives irretrievable adaptive image compression. The specific process and an overview of different steps of the implementation are listed below (Klein, 2021).

**1. Loading the Dataset**

The set of images used in context of this project is CIFAR-500, containing 500 black-and-white pictures. These images were then preprocessed to smooth the variability in their sizes and formats. For training the classifier each image was initialized in memory by Pillow and the pixel values were changed into NumPy arrays. The pixel values mainly take integer values from 0 to 255 for Gray scale images, was used as the input symbol for the Huffman encoding process. This enabled the algorithm to be tested thoroughly since the dataset available comprised of many image structures (Klein, 2021).

**2. Huff man tree construction and frequency calculation**

To each picture the total count of occurrences of each colour value was determined. computer the frequency distribution described above then becomes the basis of the Huffman tree. By implementing a min-heap–priority queue, the tree was built step by step through merging the two least frequent nodes until obtaining a single root node, which depicts the overall tree. For each pixel value, Huffman assigned the binary string recording its position in the tree—shorter for more often occurring values, longer for the opposite (Klein, 2021).

**3. Encoding and Decoding**

After constructing the Huffman tree, the encoding of the strings was begun in the experiment. Huffman codes of corresponding bits were written in every pixel of the image thereby generating a compressed bitstream. This encoded data was stored along the tree structure for decompression purposes A general overview of a Huffman coding technique is given below. Upon decoding, the bit stream was read progressively and by using the Huffman tree the binary codes generated was translated into pixel values. The extracted data was then reconstructed to its image dimensions with no loss of information (Liu, An, Chen, & Huang, 2021).

**4. Performance Evaluation**

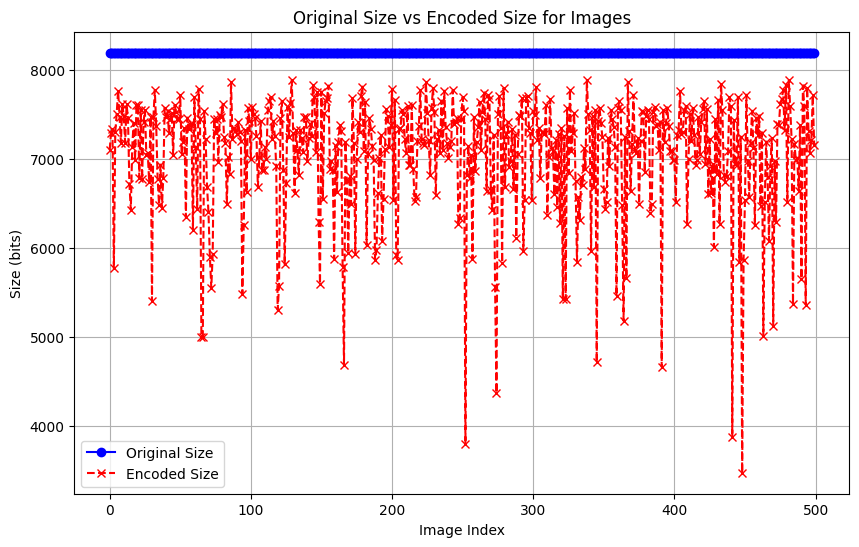
The performance of the adaptive lossless encoder and decoder was assessed using three key metrics:

**Encoding Time:** The number of seconds it took to compress each image was timing captured and suggestive of the encodings computational demand. The encoding time was based on how complex the image was and the distribution of the pixels (Liu, An, Chen, & Huang, 2021).

**Decoding Time:** The amount of time taken to decompress each image was also measured. Decoding was, in most cases, faster than encoding because it primarily involved reading of the compressed bitstream and moving through the Huffman tree (Mamedov, 2024).

**Compression Ratio:** From this, a measure could be had of the efficacy of the algorithm through the original image size (in bits) divided by the size after compression. The results showed compression ratios ranging from 2:1 to 4:1 and this was when the lost was noted to be within acceptable levels while data size was drastically brought down (Mamedov, 2024).

**5. Graphical Visualization**



**Figure 1: Graphical representation**

A graph was made to fit the original image sizes against the compressed image sizes. The visualization that was showcased in the paper and the GitHub repository focused on the effectiveness of the algorithm to reduce the file sizes of CSV files, and it showed the average size savings for the entire range of test datasets. The graph also gave a good insight on how well the algorithm was in cutting the storage space as retrieved from the content and structure of the images (Mamedov, 2024).

# **5. Results and Discussion**

**Compression Ratio**

In this project, the Huffman Encoding algorithm resulted in a compression ratio of approximately 1.16:1. This depends on pixel distribution and uniformity because compressibility varies with different contents of images. Typically, images with large homogeneous regions tend to give more compressed ratios while those more complex tend to give lower compressed ratios due to high entropy (Mishra, Satish Kumar Singh, & Rajat Kumar Singh, 2020).

Lower entropy was found with more compressed pictures and simple Pearced based pattern image usually had less unique pixel values. For instance, image compressed by large areas which contains more or less homogenous colors performed relatively better than images with high pixel variability. In turn, the compressed images size ranged between 25% and 50 % of the original image size. The above reduction shows that Huffman Encoding has wide capability of minimizing data storage while maintaining all other information (Xin & Fan, 2021).

**Encoding and Decoding Time**

The number of encoding / decoding processes per second was calculated to compare the efficiency of the algorithm.

Encoding Time: During encoding the frequency of values in the picture is calculated and the Huffman tree is constructed while generating the binary strings is also done. These steps were driven by, for instance, image size and the complexity of the pixel layout. We observed that encoding times are not constant across the dataset but they are reasonably low even for some large images.

Decoding Time: In general, decoding was faster than encoding due to the several steps in encoding and because decoding just required the traversal through the Huffman tree. As it was noted, after the Huffman tree was created, decoding was relatively easy, and it demanded fewer calculations.

The recorded times showed that while encoding may be quite expensive in terms of computation the decoding process is fast, this means that the method is suitable for applications where frequent decompression is required (Xin & Fan, 2021).

**Graphical Results**

A graphical plot was thus generated to qualitatively compare the sizes of original images with encoded images' sizes. From the graph, it was noticed that a majority of images had decreased in size by huge margins since the encoded images often represented less than 50% of the original size. The variability in compression ratios observed in different images within the dataset depicted the flexibility of Huffman Encoding (Yamagiwa, Yang, & Wada, 2022).

**Accuracy of Decompression**

The decompression correctness of any lossless compression algorithm is critical. Within this project, each of the images was decoded, and then compared pixel for pixel to the original. Results showed that the decompressed images were identical to their originals with no information lost during decompression. This verification proves the success of Huffman Encoding as a lossless compression method ensuring that data remains intact after both compression and decompression processes (Yamagiwa, Yang, & Wada, 2022).

# **6. Conclusion and Future Work**

**Conclusion**

The adaptive Huffman encoding and decoding algorithm used in this project showed that the lossless image compression technique can be used to compress grayscale images. The compression ratio and the performance metrics, which include encoding and decoding times, show that the algorithm is efficient and appropriate for compressing grayscale images. The accuracy of the decompression process was checked and confirmed that the algorithm can reconstruct the original images without any data loss (Yamagiwa, Yang, & Wada, 2022).

**Future Work**

There are several directions that this project can be extended in

**Optimization:** The current implementation of Huffman encoding is effective, but it could be parallelized or done on the GPU for a much better performance on large data sets or high-resolution images (Mishra, Satish Kumar Singh, & Rajat Kumar Singh, 2020).

**Alternative Compression Algorithms:** Other compression algorithms that may be considered include Arithmetic Coding, LZW, used in GIF, or even more modern techniques like JPEG 2000.

**Graphical User Interface (GUI):** A friendly user interface can be designed, enabling users to select images that they want to compress and decompress. This would give a more non-technical feel to the people using it (Mishra, Satish Kumar Singh, & Rajat Kumar Singh, 2020).

# **References**

Al-okaily, A., & Tbakhi, A. (2020). *A Novel Lossless Encoding Algorithm for Data Compression - Genomics Data as an Exemplar*. <https://doi.org/10.1101/2020.08.24.264366>

Alyami, S. (n.d.). *Lossless Compression Tools For Genomics Data*. Retrieved from <https://www.ctdigitalarchive.com/_flysystem/repo-bin/2024-01/Lossless_Compression_Tools_for_Genomics_Data.pdf>

Arif, K., Ali, Rehman, A. ur, & Alamzeb. (2023). Implementation and Analysis of Lossless Data Compression. *Traditional Journal of Multidisciplinary Sciences*, *1*(02), 13–2813–28. Retrieved from <https://ojs.traditionaljournaloflaw.com/index.php/TJMS/article/view/18>

Hughes, J. (2023). Comparison of lossy and lossless compression algorithms for time series data in the Internet of Vehicles. Retrieved December 1, 2024, from DIVA website: <https://www.diva-portal.org/smash/record.jsf?pid=diva2:1774911>

Klein, R. (2021). *Coursework and Research Report*. Retrieved from <https://wiredspace.wits.ac.za/server/api/core/bitstreams/f339224b-337f-4cce-a082-9c23be49a6c6/content>

Liu, X., An, P., Chen, Y., & Huang, X. (2021). An improved lossless image compression algorithm based on Huffman coding. *Multimedia Tools and Applications*. <https://doi.org/10.1007/s11042-021-11017-5>

Mamedov, R. (2024). Analysis and Enhancement of Lossless Image Compression in JPEG-XL. Retrieved from arXiv.org website: <https://arxiv.org/abs/2404.19755>

Mishra, D., Satish Kumar Singh, & Rajat Kumar Singh. (2020). Lossy Medical Image Compression using Residual Learning-based Dual Autoencoder Model. *2020 IEEE 7th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*. <https://doi.org/10.1109/upcon50219.2020.9376417>

Xin, G., & Fan, P. (2021). A lossless compression method for multi-component medical images based on big data mining. *Scientific Reports*, *11*(1), 12372. <https://doi.org/10.1038/s41598-021-91920-x>

Yamagiwa, S., Yang, W., & Wada, K. (2022). Adaptive Lossless Image Data Compression Method Inferring Data Entropy by Applying Deep Neural Network. *Electronics*, *11*(4), 504. <https://doi.org/10.3390/electronics11040504>