Usage of Optimal Compression Techniques for Data and Image Archiving

Neha S

*Department of Computer Science and Engineering,*

*Amrita School of Computing, Bengaluru,*

*Amrita Vishwa Vidyapeetham, India.* [bl.en.u4aie23022@bl.students.amrita.e](mailto:bl.en.u4aie23022@bl.students.amrita.edu) [du](mailto:bl.en.u4aie23022@bl.students.amrita.edu)

Swati V

*Department of Computer Science and Engineering,*

*Amrita School of Computing, Bengaluru,*

*Amrita Vishwa Vidyapeetham, India.* [bl.en.u4aie23031@bl.students.amrita.e](mailto:bl.en.u4aie23031@bl.students.amrita.edu) [du](mailto:bl.en.u4aie23031@bl.students.amrita.edu)

Penumarthi Hima Varshini *Department of Computer Science and Engineering,*

*Amrita School of Computing,*

Bengaluru*,*

*Amrita Vishwa Vidyapeetham, India.* [bl.en.u4aie23046@bl.students.amrita.e](mailto:bl.en.u4aie23046@bl.students.amrita.edu) [du](mailto:bl.en.u4aie23046@bl.students.amrita.edu)

Navya D

*Department of Computer Science and Engineering,*

*Amrita School of Computing, Bengaluru,*

*Amrita Vishwa Vidyapeetham, India.* [bl.en.u4aie23050@bl.students.amrita.e](mailto:bl.en.u4aie23050@bl.students.amrita.edu) [du](mailto:bl.en.u4aie23050@bl.students.amrita.edu)

Vineetha K V

*Department of Computer Science and Engineering,*

*Amrita School of Computing, Bengaluru,*

*Amrita Vishwa Vidyapeetham, India.*

[jain\_vineetha@blr.amrita.edu](mailto:jain_vineetha@blr.amrita.edu)

Ullas S

*Department of Computer Science and Engineering,*

*Amrita School of Computing, Bengaluru,*

*Amrita Vishwa Vidyapeetham, India.*

[s\_ullas@blr.amrita.edu](mailto:s_ullas@blr.amrita.edu)

***Abstract—*The project TinyHuff aims to develop a web application that will facilitate the compressing and decompressing of text and image files using the Huffman coding algorithm and Chroma Subsampling. The main objective is basically to show how effective Huffman coding can be in minimizing the sizes of files while preserving the content of the file. TinyHuff provides a user interface for the user to upload his/her files for compression and download in the compressed format. Apart from this, it also has the feature to decompress the respective human-readable format of the files from their compressed formats and restore them to their normal states. The project development of TinyHuff shall be based on the need for more storage- and data-transmission- efficient respondents' issues without losing quality or speed. TinyHuff provides an effective way to reduce the size of the file by using Huffman coding and thus is a relevant tool for end- users who need to store and manage their digital storage easily.**

***Keywords—Huffman coding, Chroma Subsampling, Encoding, Decoding, DCT, Binary files***

1. INTRODUCTION

Huffman coding is a fundamental data compression technique widely recognized for its efficiency in compressing text and image files while retaining all essential information. In text compression, Huffman coding reduces storage and transmission costs by assigning shorter codes to frequently occurring characters. Similarly, image compression, classifies pixel data by frequency, using shorter codes for common elements. This method proves invaluable in fields where data integrity is paramount, such as digital photography, remote sensing, video encoding, medical imaging, and image archiving. Huffman coding is particularly favored for lossless compression because it often achieves compression rates that exceed theoretical expectations, ensuring that vital details are preserved even after compression.

1. LITERATURE REVIEW

The paper by M. A. Rahman et.al. has several good advantages. First, the technique improves efficiency in compression by using binary sequences with lesser bits that assign the probability to the image data. This will result in better compression ratios when compared to the older methods, hence less storage needed and allows data to be sent faster. First and foremost, the quality of the original image is retained because it is lossless in its compression. This may be very important in the case of medical imaging or archives, where any reduction in data is just not acceptable. You can be assured that absolutely nothing will go astray. And finally, this technique is pretty versatile; it can hence work with quite a number of picture types, making it useful in many situations and industries. Now, some disadvantages from the same paper are as follows: First of all, sometimes this technique may be a bit tricky to compute as compared to other easy techniques like RLE. This technique can result in more processing time along with extra resources as well. With all this complexity, the overall efficiency may get hurt during the compression process. Also, special software or hardware may be needed for this type of coding technique that can increase development cost as well. Also, being relatively new, it has not been tested on too many image datasets or real-world scenarios. Thereby, surprises or performance hiccups that would then affect how widely it can be used and relied on might occur. On the whole, while the new lossless coding technique is very promising in making image compression better without losing data, it might add some complexity and costs.[1]

The paper by A.H.M. Zadidul Karim describes a number of remarkable advantages. In this method, compression efficiency is conveyed through the selective compression of color segments. This gives better ratios even compared with standard Huffman coding, especially on images with large areas of solid color. Besides, piecewise selection keeps relevant color details of parts of the image intact, which comes very handy if someone needs color accuracy. It is also versatile, as it can be used for different types of images.

Meanwhile, there are also disadvantages. First of all, the partial/piecewise color selection can get pretty complex. It needs more advanced algorithms and uses more computing resources than regular Huffman coding would require. Next, there's the time factor—selecting & compressing those color segments could slow down processing. This would not work so well for tasks needing real-time action. Lastly, we have to save extra data for those color blocks and the compression performed. That extra bit adds up to create overhead and sometimes actually subtracts a few of the benefits from the improved compression ratios attained. So, all in all? It makes significant enhancements within certain conditions, but at the cost of more resources and tricky steps in the implementation process.[2]

The research paper by Qusay Kanaan Kadhim has some major advantages. On image compression, discrete cosine transform does wonders by changing data from the spatial domain into the frequency domain. This change largely reduces redundancy and produces high compressions—pretty nice. Since most of the significant information ends up being in just a few low- frequency components, we can easily dispose of higher frequency parts without losing much quality. Not only does DCT play nicely with JPEG, a big standard for image compression, but it also makes DCT-compressed images quite easy to handle among different devices and software. Not only that—DCT generally introduces fewer blocking artifacts compared to other relevant methods like Discrete Fourier Transform. Fewer blocking artifacts are, of course, related to fewer visual distortions at the edges of blocks. This can be interpreted to mean there will always be more clarity in the images. Furthermore, it is apparent that the algorithm of the DCT technique has remained very easy and computationally efficient. This brings out the good side of both hardware and software options.[3]

The research paper by Keyur D. Joshi has a lot of key benefits. First, it is about the reduction in compression time for images, which actually turns out to be realized from the improvement of the efficiency of the algorithm for DCT. It realizes faster processing times; this function is very important in real-time applications and large-scale image data processing. The improved DCT algorithm has significantly better image quality while still having improved compression characteristics; thus, it will be very useful for both speed- critical and quality-critical applications such as video streaming and teleconferencing. This reduces the computational complexity of the algorithm, making it quite suitable for low-processing-power devices like mobile and embedded systems. This reduction in complexity also contributes to lower energy consumption, thereby clearly showing advantages for battery-operated devices. Finally, the improved DCT algorithm is scalable, and with its help, different sizes and resolutions process various digital images, ranging from small icons to high- resolution digital photographs, making it versatile for many applications. [4]

Some disadvantages are also mentioned in this research paper. Even though the algorithm takes care of preserving the quality of the image, the more detailed images or images that have a high fidelity may lose some of the finer details. The changes that have been provided in the standard DCT may involve increased complexity in knowing and implementing the algorithm, and thus might prove to be more difficult for a

developer who has not acquired sufficient expertise and training. Incompatibility issues can arise because the improvised algorithm may not implement perfectly in already existing systems and standards using the traditional DCT methods and therefore might require changes in the hardware or software infrastructure. The algorithm may also not work very well in certain special cases or on images of certain types. Handling these edge cases might be really hard. In addition, this would require a huge amount of time and resources to test and validate the correctness and reliability of the algorithm in many different real-world conditions. Lastly, if the algorithm uses any proprietary techniques, there will be legal and financial considerations about patenting and licensing that may affect its acceptance for adoption and deployment.[4]

The research paper by Dinesh P. and G. Uganya presents several advantages. The principle advantage of the paper is the considerable reduction of the image compression time. It has been able to offer faster processing times for apps that require real-time processing and huge amounts of image data due to the efficiency of the DCT algorithm employed. The improvised DCT algorithm provides faster compression without substantial loss in image quality, hence striking a balance in speed and quality for applications that are time- critical and require quality, like video streaming and teleconferencing. This has also reduced the complexity of the algorithm in computation, hence making it quite suitable for devices with limited processing power and beneficial for mobile devices and embedded systems that want efficient image compression without consuming many resources. Besides that, with reduced computation time and complexity, this algorithm shall reduce energy consumption, thus efficiently applicable for battery-operated devices. Still, the added advantage of the improved DCT algorithm is that it shall be very versatile and applicable in all aspects—veryeasy to scale with respect to different applications of image size and resolution, from a small icon image to a high-resolution photograph.[5]

It also enumerates various disadvantages in the paper. Where the improvised algorithm is trying to sustain image quality, there could be a loss of details, especially in extremely detailed images or when high-fidelity images are needed. The changes to the standard DCT will add more complications to the comprehension and implementation of the algorithm, hence demanding more skills and training for the developer. Also, the such improvised algorithm may not be that much compatible with existing systems or standards using the traditional DCT method. That means changes in hardware or even software infrastructure could be necessary. The algorithm can have edge cases and anomalies, where it performs suboptimally under certain scenarios or with a certain type of images. Quite a challenge then to identify these issues and improve on them. Furthermore, it requires a lot of testing and validation to ensure that the improvised working of the algorithm is correct and reliable under various real-life conditions, which can again be time-consuming and resource- intensive. For instance, if the algorithm involves any proprietary techniques, there will be legal and financial issues related to its patenting and licensing that will influence its acceptance and fielding. [5]

1. IMPLEMENTATION

If the source file is an image, then chroma subsampling is performed to reduce the resolution of color information while preserving brightness. Reducing an image size this way means that no actual perceived degradation in its quality would take place. The subsampled image will then be transformed into text using the `imageToTextService`. This step will allow applying text-based compression techniques. This text file then undergoes Compressor Service. If the input file is a text, then it will undergo compression by the Compressor Service.

1. *Chroma Subsampling:*

Chroma subsampling is a technique that reduces the amount of data used in representing color information in an image. It is usually applied to formats like JPEG image files. Here’s a detailed methodology:

* 1. *Separate Luminance and Chrominance:* The image will be converted from the RGB color space to the YCbCr color space, where Y represents luminance and Cb and Cr represent chrominance.
  2. *Reduce Chrominance Resolution:* Basically, this means that the resolution for the chrominance channels is reduced. In 4:2:0 subsampling, for instance, the chrominance channels are halved in horizontal and vertical resolution*.*

This subsampling relies on the phenomenon of the human eye being less sensitive to changes in color than in brightness.

* 1. *Compression:* Luminance and subsampled chrominance channels are compressed separately. Typically, luminance would be maintained at full resolution for the sake of image quality.

1. *Compressor Service:*

The compressing technique used is Huffman coding. This technique compresses data by assigning the shorter binary codes for the more frequently occurring symbols and longer codes for less frequently occurring symbols. Here is the detailed, step-by-step methodology:

* 1. *Frequency Analysis:* Calculate the frequency of every symbol in the input data. This base of frequency information shall be used for creating the Huffman tree.
  2. *Construction of the Huffman Tree:* A priority queue is used in the construction of a Huffman tree. Here, a min-heap is utilized for this purpose. All the symbols are initially defined as leaf nodes, and then the nodes are prioritized by their frequencies. Extract the two nodes with the lowest frequencies from the queue. Create a new node with these two nodes as children. This new node takes as its frequency the sum of the two child nodes. This newly created node is, in turn, inserted back into the priority queue. This whole process keeps on repeating until only one node is left in the queue. That node is taken as the root of the Huffman tree.
  3. *Generation of Huffman Codes:* Next, binary codes for each symbol have to be generated. To do so, one needs to traverse the Huffman tree generated from the root node down to every leaf node. A '0' is added to the code when moving left, and a '1' when moving right. These codes are stored in a dictionary in which every symbol is mapped to its binary code.
  4. *Data Encoding:* The input data is then encoded by replacing every symbol with its binary code as given in the

dictionary. This will result in a binary stream representing the compressed data.

1. *Writing and Reading into Binary Files:*

Writing and reading into binary files are the essential parts of compression and decompression processes, respectively. Here's the detailed methodology:

* 1. *Writing Compressed Data:* First, a compressed data stream is converted into binary form. Then this binary data is written onto a binary file. Unwanted space is not wasted, and the data is stored compactly therein.
  2. *Reading Compressed Data:* When the data has to be decompressed, the binary file is read, and the binary data are retrieved back. This binary data is then decoded again into its original form by the proper decompression algorithm.

1. *Zip File Creation:*

After the compression, the compressed files are collated to a ZIP file for download. Below are the steps:

* 1. *Zip File Creation:* The ZIP file is created using ZipOutputStream. Every compressed file is written to the ZIP file as a ZipEntry. It creates the `file data reading' and writes the same to `ZIP Output Stream.'
  2. *File Cleanup:* Once added successfully, files in a ZIP file, the original compressed files have to be removed to save space.

1. *File Decompression*:

The `decompress` method is responsible for decompressing the uploaded ZIP file. The step-by-step process is as follows:

* 1. *Unzippingthe File:* The unzipped ZIP file is opened, and the single files inside it are methodically extracted. These files are then identified based on their unique extensions such as huff, .ser, and .txt, which would be important for the succeeding process of decompression.
  2. *Decompression:* Files that are identified are passed to ‘Decompressor Service’. This service reconstructs the original file with the decompressed data.
  3. *File Cleanup:* Temporary files created while unzipping and decompressing are removed for effective usage of resources.

1. *Image Decompression:*

The `Decompressor Service` is used to decompress ZIP files containing compressed image data. The steps involved in this process are as follows:

* 1. *Unzipping and Identification:* The ZIP file is unzipped as in the general case of decompression and the individual files are identified. The files are decompressed using the

`DecompressorService` to get backtheoriginaltext file.

* 1. *Text to Image Conversion:* This decompressed text file is then reconverted into an image by the

# `TextToImageService`.

* 1. *File Cleanup:* Temporary files created during the process are cleaned to keep the system clean and trim.

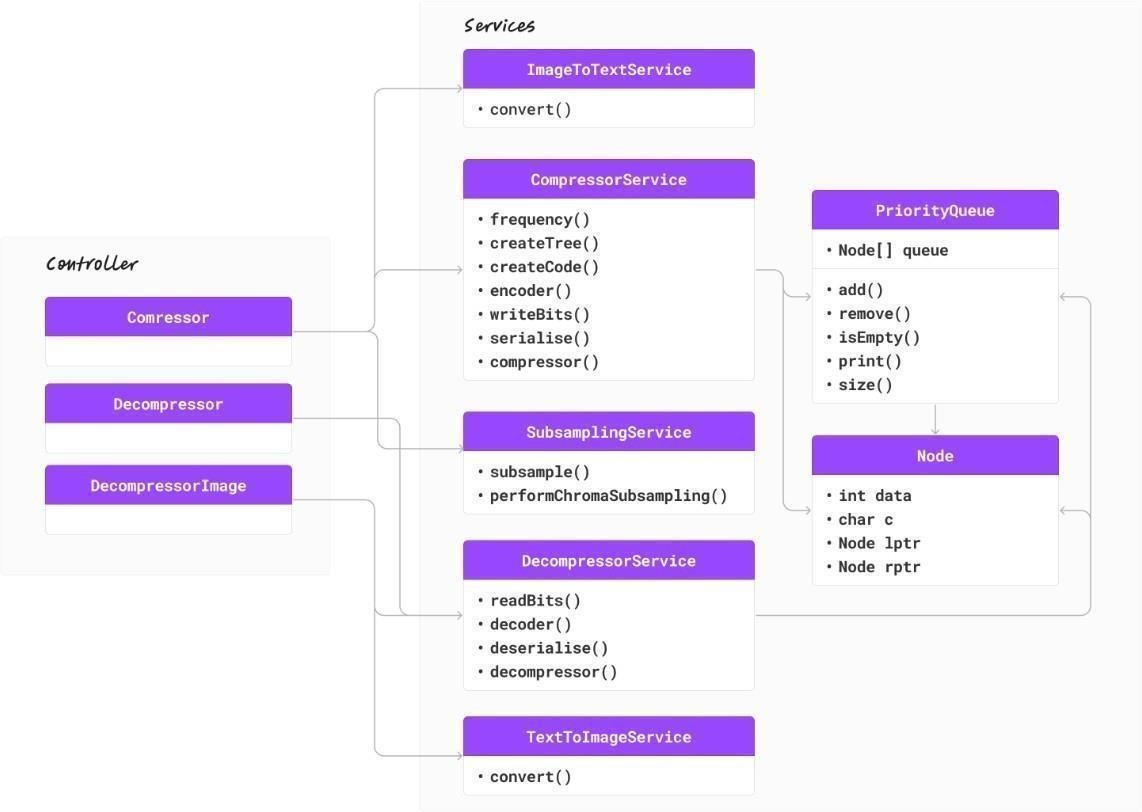


Fig. 1. UML Diagram

1. *Decompressor Service:*

# Decompression involves Huffman decoding to get back the data which was encoded using the Huffman algorithm. It involves resuming the binary stream again into the form of original symbols with the help of the same Huffman tree. Here's how it works:

* 1. *Reconstructing the Huffman Tree:* The same Huffman tree used during encoding is necessary to be available for decoding. Such a tree can be reconstructed with the aid of the frequency information, or it can directly be stored and retrieved if the data format supports it.

# *Decoding the Binary Stream:* It traverses left from the root node for a '0' and right for a '1' during the bit-by-bit traversal of the encoded binary stream. The symbols are translated according to the reached leaf node. The symbol is added to the output, and the tree is navigated from the root for another lot of bits.

* 1. *Output the Decoded Data:* The procedure is iterated, or more precisely, incrementally executed on the following step as many times as needed until the full stream of binary data passes through the box lengthwise and each and every symbol is decoded, thus reconstructing the original uncompressed data. The process iterates or more precisely continues forward on

# each following step as many times as necessary until the full stream of symbols is gained.

1. *File Management:*

# All the way in the process, temporary files that have been created due to compression or decompression are deleted once used to ensure maximally efficient storage and to avoid clutter.

1. FUTURE SCOPE

While Huffman coding works fine in many scenarios, there indeed are limitations to its use, especially in situations where JPEG images either cannot be used or are not preferred. One of the major disadvantages is that the statistical distribution of symbols in input data is quite sensitive. It works best when there are huge disparities in the frequency of the symbols so that it can allot shorter codes to the common symbols. However, if the frequencies of the symbols are relatively uniform, Huffman coding's compression efficiency is reduced. That is, if the symbol frequencies are pretty much close to one another, the possible compression in such a case will be very low, compared with cases having apparent imbalance in the symbol frequencies.

The second limitation has to do with the fact that Huffman coding processes data one symbol at a time; hence, it is inherently unable to capture higher order correlations or

complex patterns within the data. Thus, in turn, it limits its capacity to perform effective compression of datasets where there are intricate interdependencies between symbols. While compressing data with complicated patterns or interdependencies, other techniques, arithmetic coding or context-based modeling, are likely to provide far superior results. This can be attributed to the fact that these advanced methods will define the structure of this data more exactly and hence guarantee a higher compression standard. This is why, when working with large datasets, it often helps a lot to consider other compression strategies for optimum output.

1. RESULT AND ANALYSIS

To analyze the optimization of the implemented algorithm, we used text samples of different sizes from the Canterbury Corpus. [7] To look at the image compression ratio, we downloaded images of all sizes and colors from stock images and compiled them into a repository. [8]

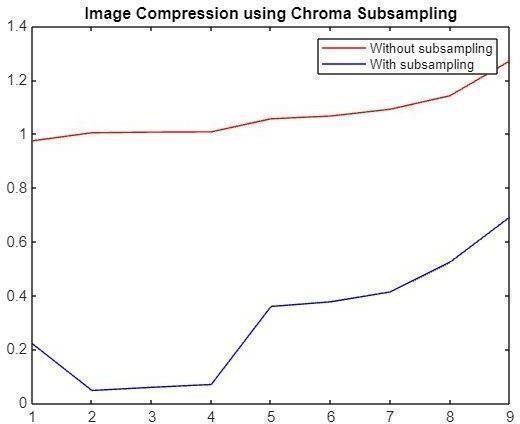


Fig. 2 Image Compression using Chroma Subsampling

Fig. 2 is the relationship between the Shannon score and the compressed rate. One axis represents the Shannon score, but the other shows entropy or average quantity of information per symbol amongst the data set. The compressed ratio is plotted against the y-axis to depict efficiency; better efficiency is depicted by a high value of this compression ratio. From this graph, we find that there is an increase in the compressed ratio as the Shannon score raises from 4.4 to 4.8; it directly tells that better compression is upon higher entropy.

After reaching the most compressed ratio at roughly about

3.6 at Shannon score 4.8, the graph takes a steep fall. That is to say, better entropy is not going to help much in increasing the level of compression, past a certain point. Instead, literally, it might mean that the data has more randomness, so compression would not be easy to do effectively. All the way up to 6, the Shannon score is decreasing slowly; in simple terms, that means the compressed ratio is also decreasing slowly, which shows how there are diminishing returns for compression with growing entropy. On a general basis, the graph shows that with increased entropy, there is better efficiency in compression, but that reaches a certain level after which it flattens off. This is kind of typical of most or all of the compression algorithms available, as in the case of Huffman coding. So effective performance to a moderate entropy level is the maximum that the algorithms could strive for. Past this stage, the complexity in data surpasses the efficient compression of the algorithm and thus compresses ratios reduce.

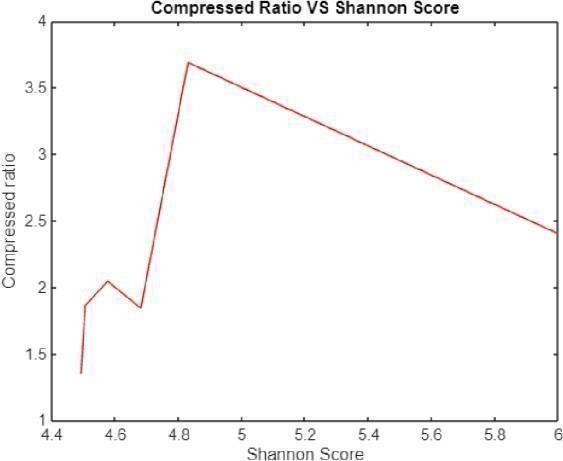


Fig.3 Image Compression using Chroma Subsampling

Images store a lot of information including color details. In Fig. 3, the X-axis shows compression level (how much the image is shrunk) and the Y-axis represents image quality. There are two lines on this graph: red for "no chroma subsampling" and blue for "with chroma subsampling."

A picture with chroma subsampling has less color data recorded in it. This reduces the file size, but it may also result in a minor loss of color detail. As compression increases, the blue line (with chroma subsampling) exhibits a sharper decline in quality since color information is given up in favor of lower size. Better color quality is indicated even at greater compression levels when the red line (no chroma subsampling) remains higher, however this also leads to bigger file sizes.

In essence, choose without subsampling when image quality is crucial and file size isn't a big deal, like for detailed photos. Choose subsampling when file size is important and some color loss is acceptable, like for icons or graphics.

1. CONCLUSION

Clearly, Huffman coding is very useful for data archiving. It reduces the file size tremendously without losing vital information, which is noted from the graphs in Figures 2 and 3. This is based on the fact that shorter codes are used with frequently occurring symbols to attain the best rate of compression rate without loss of integrity of data.

Moreover, the integration of chroma subsampling improves performance, especially in image compression. Chroma subsampling optimizes compression ratios during lossy compression without losing visual quality by reducing the resolution of color information while resolving the brightness. These are factors that underline the effectiveness of Huffman coding in data archival and image compression, thus proving its essential place in efficient storage and transmission where preservation of the quality of data is a must.

ACKNOWLEDGEMENT

We would like to thank all our supporters who gave their time, helpful advice, and encouragement throughout this research project. We extend our gratitude to technical people who helped us in running the studies. Your help minimized the hassle of dealing with complex parts. Your insights greatly improved our work.

REFERENCES

1. Rahman, Md Atiqur, et al. "A novel lossless coding technique for image compression." 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR). IEEE, 2018.
2. Karim, AHM Zadidul, et al. "Image Compression using Huffman Coding Scheme with Partial/Piecewise Color Selection." 2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON). IEEE, 2021.
3. Kadhim, Qusay Kanaan. "Image compression using discrete cosine transform method." International Journal of Computer Science and Mobile Computing 5.9 (2016): 186-192.
4. Joshi, Keyur D., and Vishal Saha. "Reducing Image Compression Time using Improvised Discrete Cosine Transform Algorithm." 2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT). IEEE, 2022.
5. Dinesh, P., and G. Uganya. "Enhancing the Efficiency and Sensitivity in Discrete Cosine Transform based Image Compression and Comparing with Discrete Wavelet Transform." 2022 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS). IEEE, 2022.
6. Barbhuiya, AHM Jaffar Iqbal, Tahera Akhtar Laskar, and K. Hemachandran. "An approach for color image compression of JPEG and PNG images using DCT and DWT." 2014 International Conference on Computational Intelligence and Communication Networks. IEEE, 2014.
7. Benchikh, Salam, and Michael Corinthios. "A hybrid image compression technique based on DWT and DCT transforms." (2011): 11- 11.
8. Thery, Laurent. Formalising Huffman's algorithm. Diss. Università degli Studi dell'Aquila, 2004
9. A. Gopinath and M. Ravisankar, "Comparison of Lossless Data Compression Techniques," 2020 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 2020.
10. G. Spasova and I. Boychev, "A Method of Color Images Compression," 2021 International Conference on Biomedical Innovations and Applications (BIA), Varna, Bulgaria, 2022. R. Stasinski and G. Ulacha, "Mixed Huffman codes for on-line and off- line applications," 2022 Data Compression Conference (DCC), Snowbird, UT, USA, 2022.
11. S. More, V. Sanivarapu, Y. Sharma, V. M. Thigale and M. Suguna, "Enhancing Data Compression: A Dynamic Programming Approach with Huffman Coding and Burrows-Wheeler Transform," 2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2023
12. Kamatar, Vishwanath S., and Vishwanath P. Baligar. "A Novel Hybrid Compression Scheme for Lossless Gray Scale Image Compression." 2023 4th IEEE Global Conference for Advancement in Technology (GCAT). IEEE, 2023.
13. Ajay, Lekshmi S., and Sreenidhi Prabha Rajeev. "Comparative Analysis of Data Compression using Canonical Huffman and Golomb Rice Encoding in Verilog HDL and Implementation in FPGA." 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2023.
14. Radhakrishnan, Meera, R. Ramya, and Mahesh As. "Data Security Using Compression Methods in the Area of Plumbing Problems in a Cloud Environment." 2023 9th International Conference on Smart Computing and Communications (ICSCC). IEEE, 2023.
15. Das, Kunal, and D. Radha. "Implementation and Analysis of Image Compression Using Ramanujan’s Sum." 2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2022.
16. Bhuvaneswari, R., and P. R. Tharaniesh. "Exploring ChatGPT for Email Content Compression and Summarization." 2023 4th

International Conference on Communication, Computing and Industry 6.0 (C216). IEEE, 2023.

1. Ramachandran, Raji, Arpit Aggarwal, and Kanak Varshney. "Text Summarization and Image Decomposition of Medical Documents." 2023 4th IEEE Global Conference for Advancement in Technology (GCAT). IEEE, 2023..
2. Varthakavi, Sai Sasank, et al. "A steganographic analysis using batch steganography." 2020 IEEE International Conference for Innovation in Technology (INOCON). IEEE, 2020.
3. Sha, Akhbar, et al. "EzStream: Video Compression and Textual Representation for Multimedia Sensor Networks." 2023 7th International Conference on Electronics, Communication and Aerospace Technology (ICECA). IEEE, 2023.
4. <https://corpus.canterbury.ac.nz/>
5. <https://github.com/swativ5/datasetForHuffman>