

Comparison Between Different Compression Techniques using Various Encoding Methods

Angshuman Mazumdar¹, Rahul Saidha², Nitin Nayak³

^{1,2,3}B. Tech (ECE), Vellore Institute of Technology,
Vellore, Tamil Nadu, INDIA

¹ angshuman.mazumdar2015@vit.ac.in, ² rahul.saidha2015@vit.ac.in, ³ nitin.nayak2015@vit.ac.in

ABSTRACT

Abstract: *The paper discusses the different encoding techniques used in computer communication, and aims to draw comparison between them, by comparing how much the techniques compress images and text before transmission. The different techniques used to draw out the comparison are: Run Length Encoding (RLE), Huffman Encoding, Lempel-Ziv Encoding and Discrete Cosine Transform Encoding (DCT). Through the simulations, the various outputs are compared for their compression ratio, space savings and quality loss – and through the comparison, we try to arrive at a conclusion as to what effect the different encoding techniques have on the input data stream.*

Keywords: *Run Length, Huffman, Lempel Ziv, DCT, Compression Ratio, Space Savings, MATLAB, Bitmap, JPEG, PNG*

1. INTRODUCTION

The source of information can be analog or digital, e.g. analog: audio or video signal, digital: like teletype signal. The signal produced by source is converted into digital signal consists of 1's and 0's. This is done by the source encoder. Representation of a signal needs to be done in as few bytes as possible. This sequence of binary digits is called information sequence. The process of efficiently converting the output of analog or digital source into a sequence of binary digits is known as source encoding.

The process of reducing the size of a data file is often referred to as data compression. In the context of data transmission, it is called source coding; encoding done at the source of the data before it is stored or transmitted. Source coding is not the same as channel coding, for error detection and correction or line coding, the means for mapping data onto a signal.

2. FORMULAS AND ENCODING TECHNIQUES USED

2.1 Run Length Encoding (RLE)

- Run-length encoding is a data compression algorithm that is supported by most bitmap file formats.
- RLE is suited for compressing any type of data regardless of its information content, but the content of the data will affect the compression ratio achieved by RLE.
- Although most RLE algorithms cannot achieve the high compression ratios of the more advanced compression methods, RLE is both easy to implement and quick to execute, making it a good alternative to either using a complex compression algorithm or leaving your image data uncompressed.
- RLE works by reducing the physical size of a repeating string of characters. This repeating string, called a run, is typically encoded into two bytes. The first byte represents the number of characters in the run and is called the run count.
- The second byte is the value of the character in the run, which is in the range of 0 to 255, and is called the run value.

2.2 Huffman Coding

- The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority.
- The major steps are as follows:
 - Create a leaf node for each symbol and add it to the priority queue.
 - While there is more than one node in the queue:
 - Remove the two nodes of highest priority (lowest probability) from the queue
 - Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
 - Add the new node to the queue.
 - The remaining node is the root node and the tree is complete.
 - Traverse the constructed binary tree from root to leaves assigning and accumulating a '0' for one branch and a '1' for the other at each node. The accumulated zeros and ones at each leaf constitute a Huffman encoding for those symbols and weights.

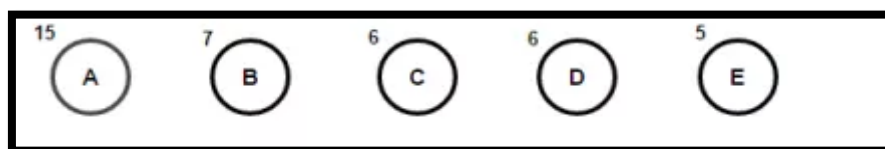


Figure 1. Example Input

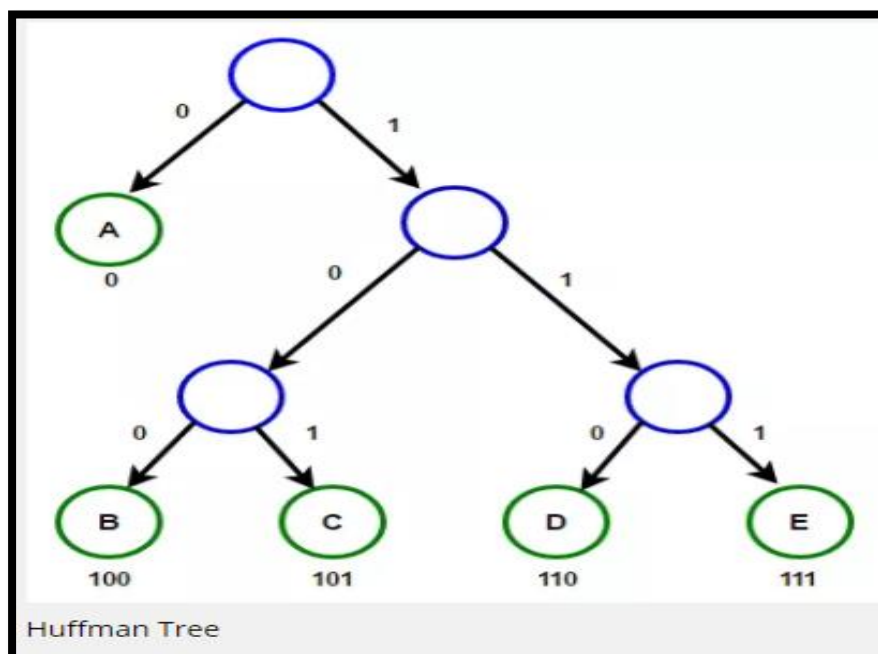


Figure 2. Huffman Tree for the example input

2.3 *Lempel-Ziv Encoding*

- Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch.
- The algorithm is simple to implement and has the potential for very high throughput in hardware implementations.
- It is the algorithm of the widely used Unix file compression utility compress and is used in the GIF image format.
- LZW compression uses a code table, with 4096 as a common choice for the number of table entries. Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
- When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks. Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.

2.4 *Discrete Cosine Transform Encoding (DCT)*

- Discrete Cosine Transform is used in lossy image compression because it has very strong energy compaction, i.e., its large amount of information is stored in very low frequency component of a signal and rest other frequency having very small data which can be stored by using very less number of bits (usually, at most 2 or 3 bit).
- To perform DCT Transformation on an image, first we have to fetch image file information (pixel value in term of integer having range 0 – 255) which we divide in block of 8x8 matrix and then we apply discrete cosine transform process (such as shifting, quantization and zig-zag scanning) on that block of data.
- After applying discrete cosine transform, we will see that its more than 90% data will be in lower frequency component.

2.5 *Compression Ratio*

- There are two types of compression ratios:

$$\text{Data Compression Ratio} = \frac{\text{Uncompressed Image Size}}{\text{Compressed Image Size}} \quad \dots (1)$$

$$\text{Savings Data Compression Ratio} = \frac{\text{Compressed Image Size}}{\text{Uncompressed Image Size}} \quad \dots (2)$$

- The Space Savings is usually expressed in percentages as:

$$\text{Space Savings} = \left\{ 1 - \left(\frac{\text{Compressed Image Size}}{\text{Uncompressed Image Size}} \right) \right\} * 100 \% \quad \dots (3)$$

- Smaller compression ratio leads to better space savings, however the quality (in case of images) doesn't always need to be better than the uncompressed one.

3. ALGORITHM

- Major method of coding used here is GUI based using MATLAB's built in app-designer called guide
- Two separate .m files made: one for text encoding and one for image encoding
- The text compression window has 3 options for **RLE**, **Huffman** and **LZW** encoding techniques
- The image encoding window also has 3 techniques: **RLE**, **Huffman** and **DCT** encoding techniques.
- The individual files contain the codes for the 3 techniques in the form of functions that are bound to buttons
- All the images are shown and encoded real time
- Also, there is the capability to check the information of each image: source and output and compare the metadata.

4. SIMULATION WINDOWS

4.1 Text Encoding Window

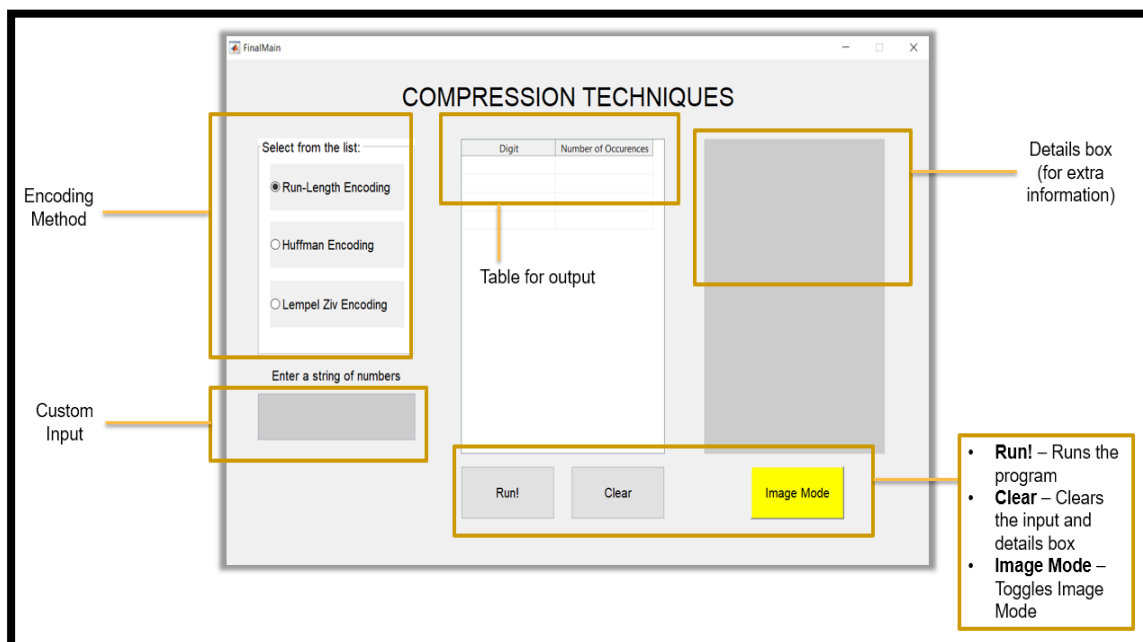


Figure 3. The "Text Encoding" Window

4.2 Image Encoding Window

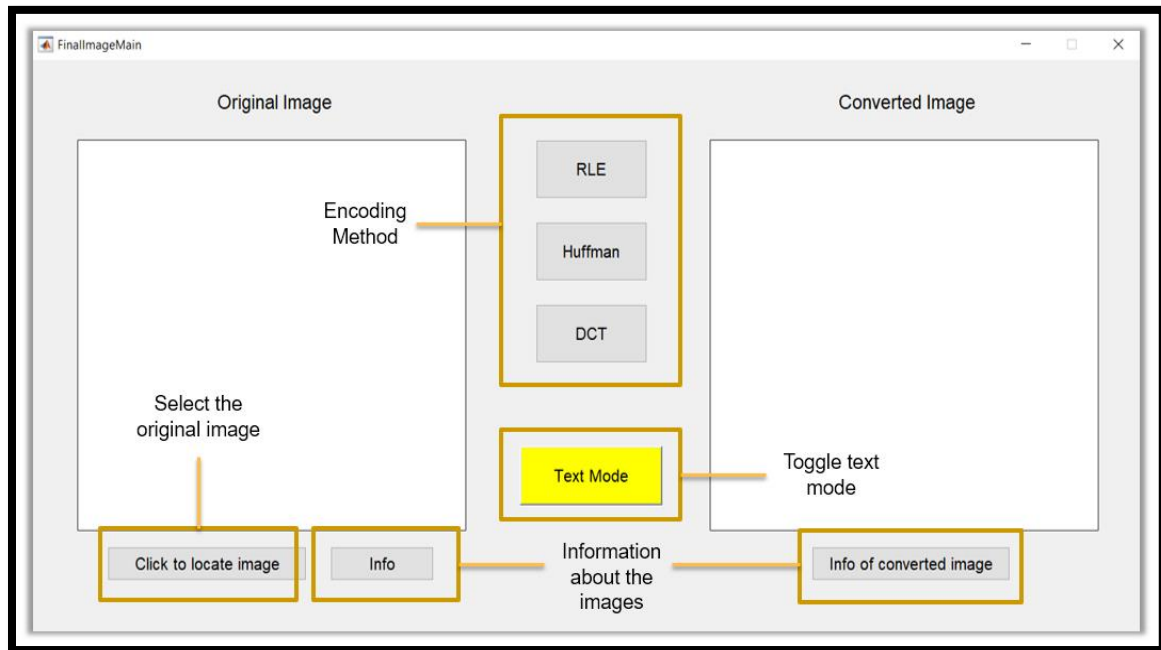


Figure 4. The “Image Encoding” Window

5. RESULTS

5.1 Text Encoding

5.1.1 Run Length Encoding

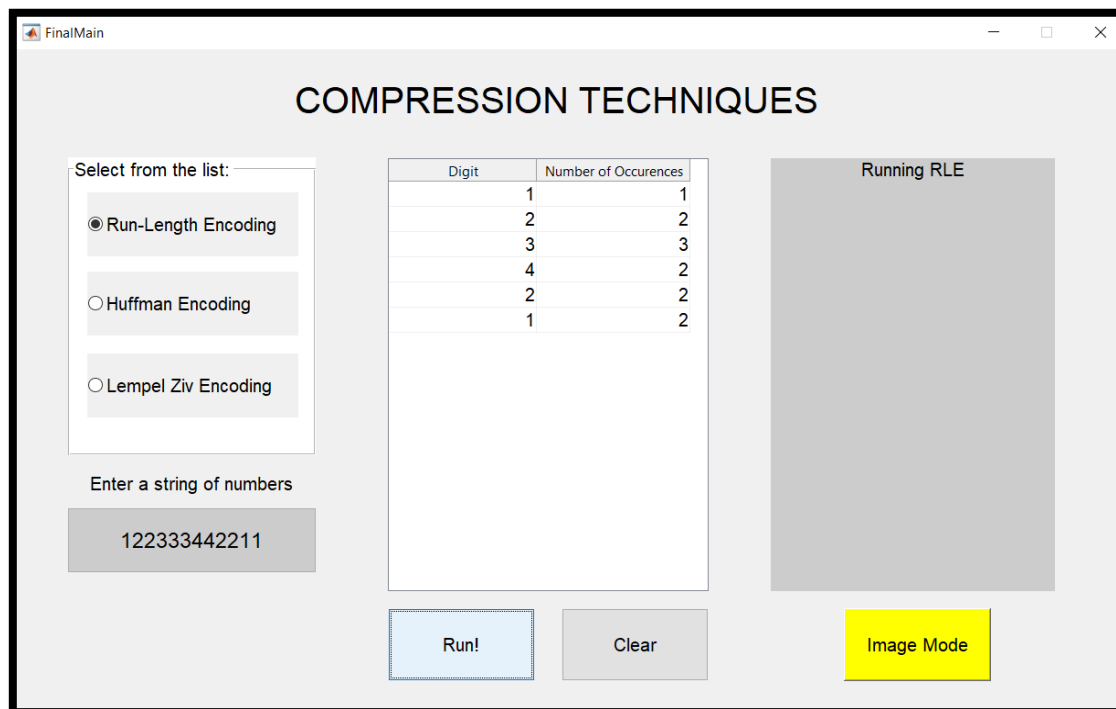


Figure 5. RLE on given input

5.1.2 Huffman Encoding

The screenshot shows the 'COMPRESSION TECHNIQUES' window. On the left, under 'Select from the list:', 'Huffman Encoding' is selected with a radio button. Below it, a text box shows 'Enter probabilities as []:' followed by '[0.5 0.1875 0.1875 0.125]'. In the center, there is an empty table with two columns: 'Digit' and 'Number of Occurrences'. On the right, a large gray area displays the binary code for the digits: 1 (011), 0 (00), and 0 (010). At the bottom, there are three buttons: 'Run!' (blue), 'Clear' (gray), and 'Image Mode' (yellow).

Figure 6. Huffman on given probabilities

5.1.3 Lempel-Ziv Encoding

The screenshot shows the 'COMPRESSION TECHNIQUES' window. On the left, under 'Select from the list:', 'Lempel Ziv Encoding' is selected with a radio button. Below it, a text box shows 'Character string (with ! at end):' followed by 'abbcdabdbbbabbaccbd!'. Below that, another text box shows 'Enter string elements (letters):' followed by 'abcd'. In the center, there is a table with two columns: 'Digit' and 'Number of Occurrences'. The 'Digit' column contains a list of strings: a, b, c, d, ab, bb, bc, cd, db, ba, abd, dbb, bab, bba, ac, and a scrollbar is visible on the right. On the right, a large gray area displays the Lempel-Ziv codes for the strings: 1 (a), 2 (b), 2 (c), 3 (d), 4 (ab), 2 (bb), 5 (bc), 9 (cd), 10 (db), 6 (ba), 1 (abd), 3 (dbb), 3 (bab), 2 (bba), 4 (ac), and 0 (empty string). At the bottom, there are three buttons: 'Run!' (blue), 'Clear' (gray), and 'Image Mode' (yellow).

Figure 7. Lempel-Ziv on given input

5.2 Image Encoding

5.2.1 Run Length Encoding

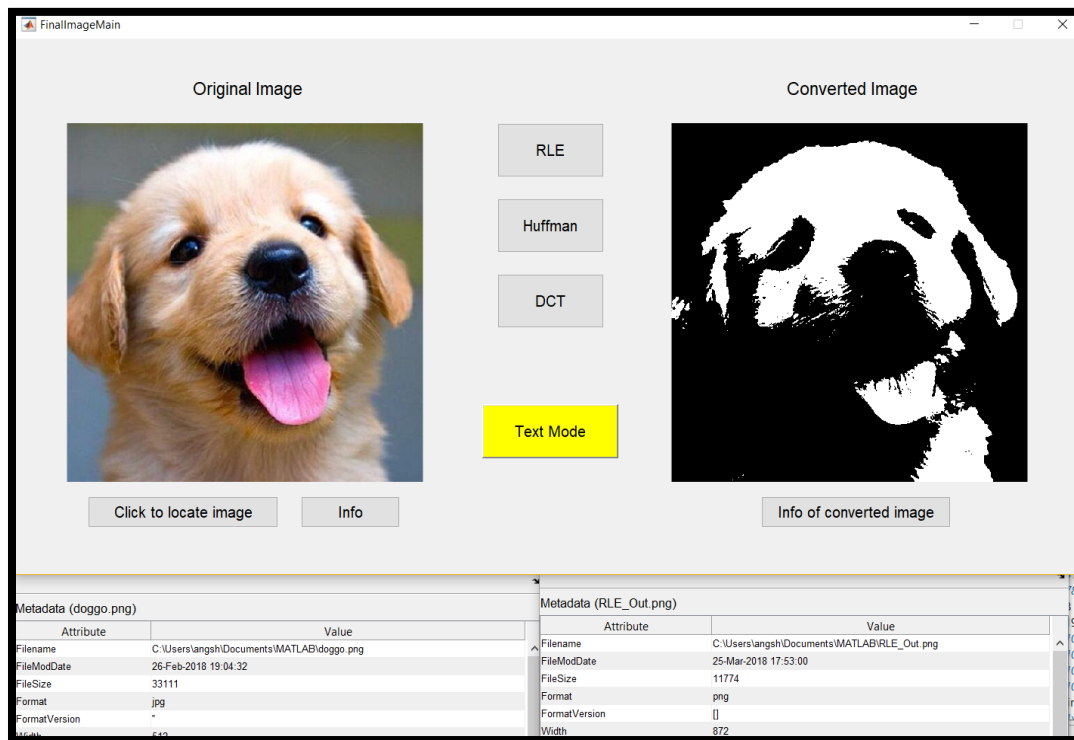


Figure 8. RLE on Image

5.2.2 Huffman Encoding

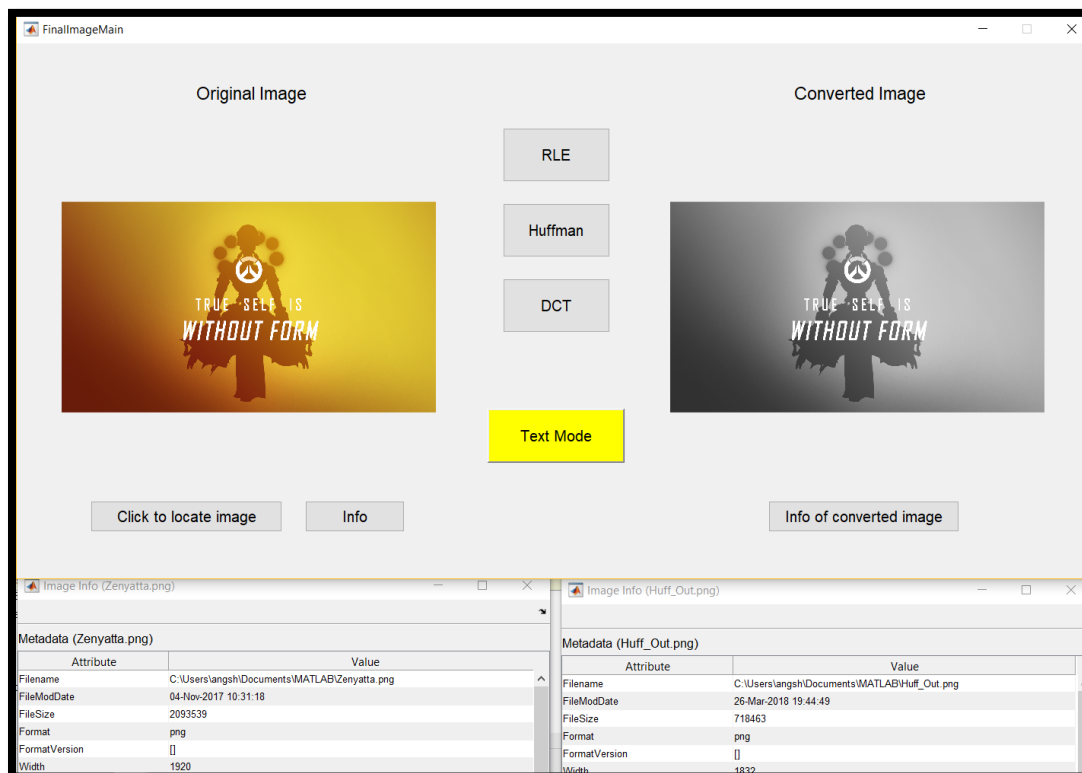


Figure 9. Huffman on Image

5.2.3 Discrete Cosine Transform Encoding

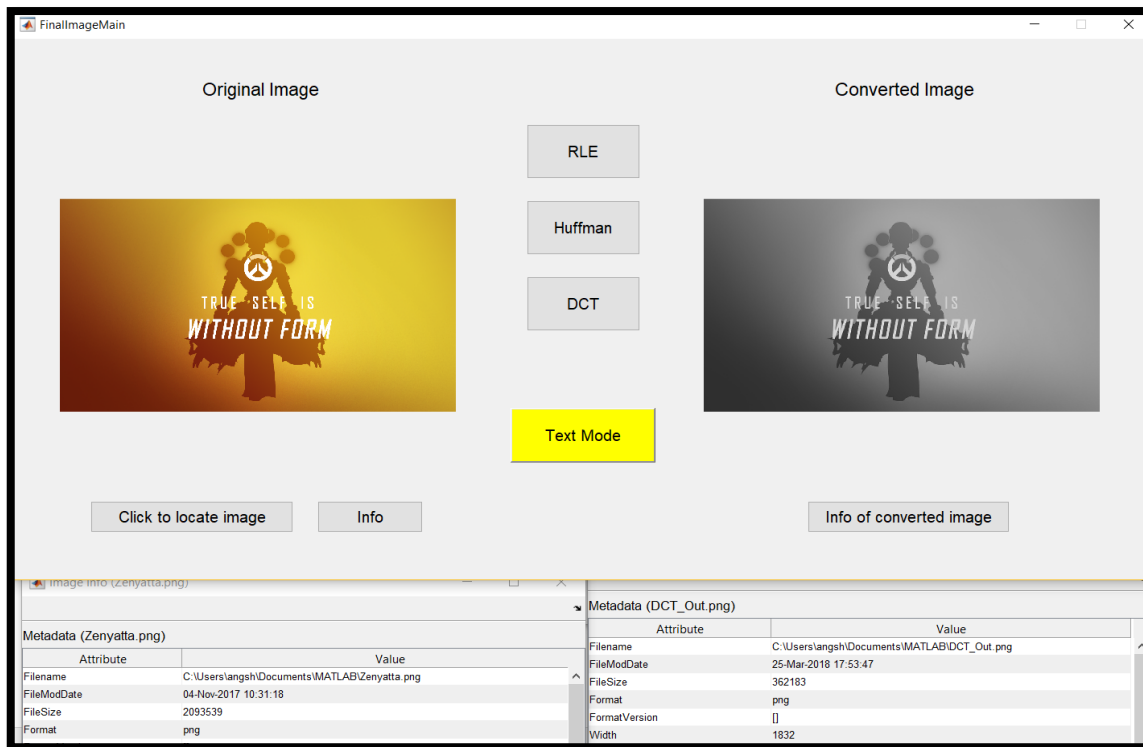


Figure 10. DCT on Image

The final result table for the Image Encoding part can be formulated as follows, which will show the different comparison parameters and results:

Table 1.

Comparison of different parameters

Method	Original Image Size (kB)	Compressed Image Size (kB)	Savings Compression Ratio (eq. 2)	Space Savings (eq. 3)
	(1)	(2)	(3)	(4)
RLE	33.1	11.7	0.353	64.7%
Huffman	2100	718	0.342	65.8%
DCT	2100	362	0.172	82.8%

5.3 Discrepancies

- While Huffman and DCT techniques work very well for large files, for smaller files it actually increases the size of the encoded image.



	1DCT_Out_doggo.png	25-Mar-18 4:47 PM	PNG File	175 KB
	1doggo.png	14-Jan-18 8:29 PM	PNG File	33 KB

Figure 11. DCT increases size of very small images

- Also, when RLE is applied to large images, even though there is a significant decrease in size, however much of the information is lost, and while decoding, very less usable data is obtained

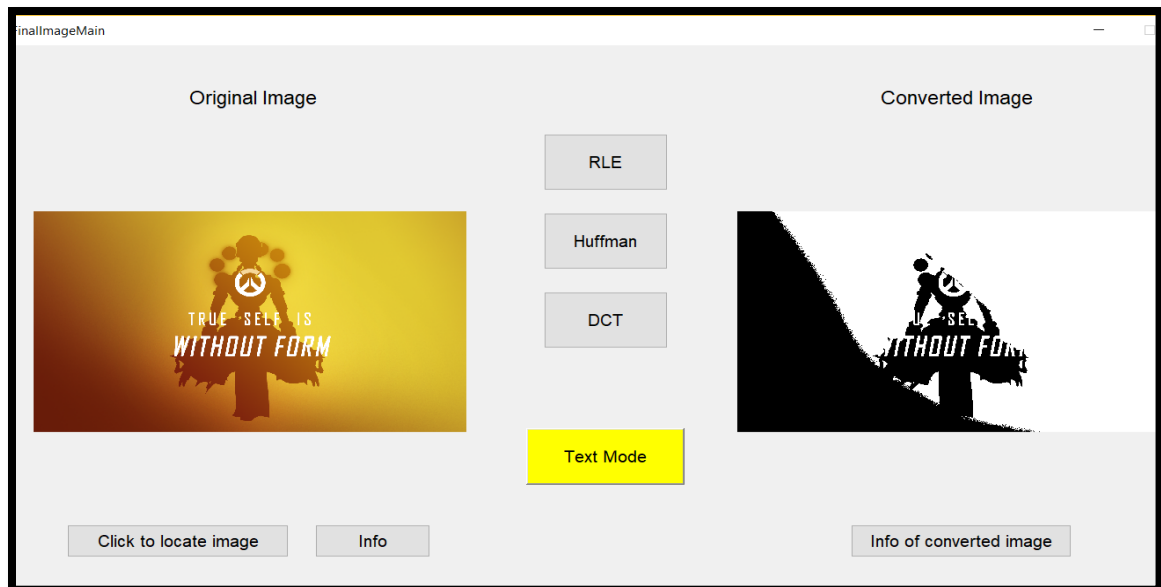


Figure 12. RLE causes image losses on some large images

6. CONCLUSIONS

From the simulations, we can conclude that first, for text encoding, it was observed that all the three methods more or less do the same amount of data compression; which depends on the length and no. of dictionary elements of the initial input data

In image encoding, it was seen that for large sized images, DCT was the best option for encoding (and compression). However, below a certain file size, RLE is still the best method. Huffman encoding is slightly of the “in-the-middle” type technique, however it has the most variations and can be easily re-coded to give even better compressibility ratio.

Finally, after a few runs, it could be observed that the compressibility of an image depends majorly on three factors: **initial file type**, **initial size** and **method used**; which is contrary to the belief that the compressibility depends on only the mode of encoding.

REFERENCES

- [1] Jan Kybic, Vaclav Hlavac, Tomas Svoboda, *“Image Processing, Analysis, and Machine Vision, A MATLAB Companion”*, Dec. 2010
- [2] Kamalpreet Kaur, Jyoti Saxena and Sukhjinder Singh, *“Image Compression Using Run Length Encoding (RLE)”*, Int’l Journal on Recent and Innovation Trends in Computing and Communication, vol. 05, no. 05, May 2017
- [3] Catherine Holloway, *“JPEG Image compression: Transformation, Quantization and Encoding”*, April 2008
- [4] Anitha S, *“LOSSLESS IMAGE COMPRESSION AND DECOMPRESSION USING HUFFMAN CODING”*, Int’l Research Journal of Engineering and Technology (IRJET), vol. 02, no. 01, April 2015
- [5] Nilkesh Patra, Sila Siba Sankar, *“DATA REDUCTION BY HUFFMAN CODING AND ENCRYPTION BY INSERTION OF SHUFFLED CYCLIC REDUNDANCY CODE”*, Technical Thesis (NIT Rourkela), 2007
- [6] Won Y. Yang (2013), *MATLAB/Simulink for Digital Communication*, Hongreung Science Publishing Co; 2nd edition (2013)