2023 4th IEEE Global Conference for Advancement in Technology (GCAT)
Bangalore, India. Oct 6-8, 2023

2023 4th IEEE Global Conference for Advancement in Technology (GCAT) | 979-8-3503-0525-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/GCAT59970.2023.10353248

A Novel Hybrid Compression Scheme for Lossless Gray Scale Image Compression

Vishwanath S. Kamatar
Computer Science and Engineering
KLE Institute of Technology
Hubli, India
vishu.kamatar@gmail.com

Vishwanath P. Baligar
School of Computer Science and Engineering
KLE Technological University
Hubli, India
vpbaligar@kletech.ac.in

Abstract—**In today's world, digital image storage and transmission play an essential role, as images are mainly involved in data transfer. Digital images take more storage space and bandwidth for transmission, so image compression is important in data compression. This paper discusses a unique and novel hybrid lossless image compression approach. Exactly half of the image is encoded with one approach, and other half of image is encoded with other approach. The method uses a block approach. In the first approach, the pixels of the block are transformed with a novel mapping transform. Each pixel nibbles are mapped as a single bits in a transform table generating more zeros, which helps achieve compression. Later, the inverse transform is applied in the reconstruction, and a single bit value from the table is remapped into pixel nibbles. With these nibbles, pixel values of the block are reconstructed without any loss. The block differential encoding approach is used for compression of other half of the image. This approach achieves lossless compression of images.**

Keywords— image compression, lossless, mapping transform, JPEG-LS, Differential encoding

I. Introduction

This Digital communication is now everyone's top priority, and digital photos are an essential component of the information shared in communications. The digital images require a larger amount of storage space, a lot of bandwidth when transmitting across channels, and more time to communicate. So, image compression is really necessary.

Due to redundancies in the data of digital images, the compression of digital images is possible[1,2]. It is indispensable that the reconstructed image is the same as the original without deviations in the pixels of the image in the lossless compression [3,4,5].

There are two main types of image compression: lossless and lossy compression. Lossless compression methods aim to preserve all the original image data, allowing for perfect reconstruction of the image. These algorithms exploit statistical properties and mathematical techniques to efficiently represent the image without any loss of information. Lossless compression is typically used in scenarios where exact pixel values are crucial, such as medical imaging or scientific data.

On the other hand, lossy compression techniques achieve higher compression ratios by selectively discarding image data that is considered less important or imperceptible to the human eye. These algorithms exploit visual and psychovisual properties to remove redundant and irrelevant information, resulting in a smaller file size. Although lossy compression leads to a loss of image quality, the trade-off is often acceptable in applications like digital photography, online streaming, and web content.

Various image compression standards and algorithms have been developed over the years. Some popular standards include JPEG (Joint Photographic Experts Group)[6], PNG (Portable Network Graphics), and HEVC (High-Efficiency Video Coding). Each standard employs different compression techniques and optimization strategies tailored to specific requirements and constraints.

Here, the work proposed is lossless image compression for generic images using a novel hybrid approach. Novel mapping transform is applied to half of the number of pixels in the image, and the transformed image data is encoded using a Huffman encoder. The other half of the number of pixels in the image are encoded using blockwise differential encoding The results are compared with the results of JPEG-LS.

The literature survey outlines different image compression methods in Section 2. Section 3 describes the proposed work and algorithms. In section 4, results are discussed. Section 5 gives the conclusion.

II. Related work

This section provides the facts and the outcomes of some image compression techniques which are lossy and lossless.

Weinberger et al., 1996 have proposed an image compression algorithm with low complexity (LOCO-I) and became a core part of the new ISO/ITU standard. The algorithm achieves a high compression ratio with low complexity. This is one of the best available methods and is standardized into JPEG-LS [5].

Faisel Ghazi Mohammed and Hind Moutaz Al-Dabbas 2018 have proposed image compression based on Wavelet Difference Reduction and using three different wavelet codecs.[7]

Kumar S. N. et al., 2021 have proposed a medical image compression method. The coefficients obtained from polynomial approximation were quantified using Lloyds, and an arithmetic encoder was used to encode[8].

Rui Lia et al., 2021 have developed a method to construct core tensor and factor matrices called correlation-based Tucker decomposition that can be used in any Nth order tensor based on TD[9].

Faisel Ghazi Mohammed and Hind Moutaz Al-Dabbas, 2018 have proposed a method for compressing images bywavelet transform filters based on wavelet difference reduction WDR on the color images[10].

Catching Ding et al., 2020 have proposed a method based on partial differential equations. The image was compressed by segmenting the image with the quadtree approach with encoding and transmitting some pixels[11].

Although references discussed above achieve good results in different application domains, there is still a need for generic compression algorithms that can be used on any image irrespective of application domains.

III. Proposed work

A hybrid lossles image compression algorithm is presented. Figure 1 shows the block diagram of the proposed methodology. The algorithm is developed to compress grayscale images. The proposed image compression algorithm compresses an image in terms of blocks. Exactly 50% of image is compressed using lossless mapping transform, and rest 50% image is compressed using block differential encoding.

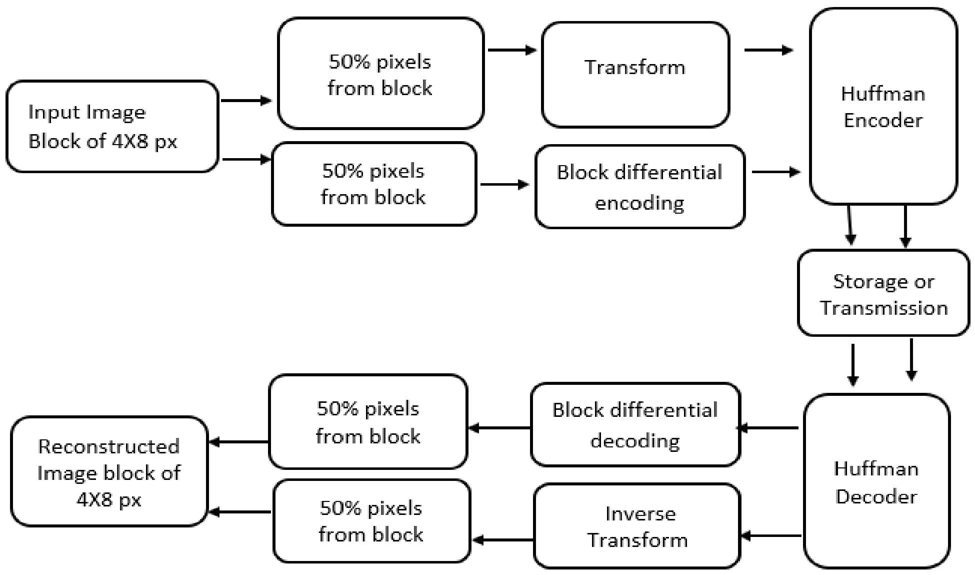

Fig. 1. Proposed methodology

A. Compression

The input image is divided into blocks of 4×8 size. The 16 pixels out of 4× 8 size block, used to compress using mapping transform are shown as asterisks in Figure 2. The other 16 pixels not shown are compressed using block differential encoding method.

*		*		*		*	
	*		*		*		*
*		*		*		*	
	*		*		*		*

Fig. 2. Context of pixels from the block

1) Mapping transform: A table of 16X16 is created and referred as a transform table to transform the pixels. For each pixel from the block, two entries are made in the transform table. Each entry in the transform table represents either the MSB nibble value or the LSB nibble value. When making entry of MSB in transform table, column indices represent MSB nibble values, and row indices represent the pixel order number. When making entry of LSB in transform table, row indices represent LSB nibble values, and column indices represent pixel order number. For the fifth pixel's MSB value of the block, an entry of '1' is made in Transform_Table[5][MSB] position. For the same pixel, entry of '1' for LSB value is made in Transform_Table[LSB][5] position. In the same way, MSBs and LSBs of other pixels entries are made in subsequent rows and columns. The transform is formulated as in Eq 1.

$$F' = \bigcup_{u=0}^{4} \bigcup_{v} \left[F\left[[i, f(u,v)], [g(u,v), i]\right]\right] \qquad (1)$$

v=0,2,4,6 when u is even
v=1,3,5,7 when u is odd
i=0,1,2,3,…,16
where , $f(u,v)$ is MSB nibble and $g(u,v)$ is LSB nibble

There are sixteen pixels in the block, so there will be sixteen MSB nibbles and LSB nibbles, which constitute a total of thirty-two nibbles. There may be chances that some entries in the transform table get overlapped because some MSB and LSB values may fit into the exact positions of the transform table, and hence the entries in the transform table will be less than or equal to thirty-two. So a maximum of thirty-two entries of ones and a minimum of two hundred and twenty-four zeros can be there in the transform table for sixteen pixels of the input block. These transform table values are considered as bits so that every transform table gives 256 bits. These 256 bits of the transform table are stored as 4X8 pixels transformed block, each pixel with 8 bits. Every 4X8 pixels block of the input image generates a transformed block of size 4X8 pixels. After the input image is processed block-wise and transformed, a transformed image is generated. This transformed image is encoded using Huffman encoding to generate a compressed bitstream.

2) Block differential encoding: The other 16 pixels of the block are stored in a array as a block of 4X4 pixels. The algorithm reads pixel values of the block and finds the maximum and minimum pixels to compress the image block. The algorithm finds the difference between the maximum and minimum pixel values within the block. The algorithm uses a predefined threshold value for the maximum difference value. If the difference between the maximum pixel and minimum pixel is less than the threshold value, then the block is encoded; otherwise, block is not encoded. The predefined threshold value is computed by $2n - 1$, where n is the number of bits that can be used to represent pixels. In the proposed method n value is 7 so threshold value is 127. Depending on the maximum difference value of the block, the number of bits required to represent other pixel differences within the block is found. The difference between the maximum pixel value and every block pixel is computed. Further, these differences are stored in another array called the difference block. The difference values from the difference block are encoded with fixed-length codes. A compressed bitstream is generated for a block using four components those are the overhead bit to represent whether the block is encoded or non-encoded block, the maximum pixel value of the block, the number of bits required to represent each difference value, and all difference values of the difference block. The same procedure is repeated for all blocks of the image. After converting all blocks into a bitstream, the bitstream is written to a binary data file. Further this binary file is compressed using Huffman coding.

B. Decompression

1) Inverse mapping transform: During the decompression of the image, the Huffman encoded bitstream is decompressed using the Huffman decoder to reconstruct the transformed image. The algorithm divides the transformed image into 4X8 size blocks. Each transformed image block is read, and the transformed table is reconstructed. In the reconstruction of the transformed table, a table of size 16X16 is created. Each pixel value of the transformed block is read, and the pixel value is converted into bits, and these bits (0’s or 1’s) are stored in a transformed table and the transformed table is regenerated.

The transform table column indices represent MSB nibble values, and row indices represent LSB nibble values. Initially, all MSB nibble values are reconstructed. The row sum of the transform table is computed. If the sum of the row is one, then the scan is made in that row to find the position of 1 (one). If the position of the transform table contains 1, then the column index is retrieved. This column index value is the MSB nibble value.

In the same way, all other MSB nibble values are retrieved for the rows where the row sum is one. If the single row contains more numbers of 1's, then only one will represent MSB, and others represent LSB's. If the row sum is more than one, there will be conflict in choosing the correct MSB value. Hence to overcome this conflict, each column sum is computed. The column which has the maximum sum is considered. If there are two 1’s in the same row, then the position that contains 1 and is also nearer to a column with maximum sum is considered. The column index of the position is retrieved as an MSB nibble value for the pixel. Similarly, MSB nibble values for other pixels of the block are reconstructed.

Similarly, after reconstruction of MSB nibble values, LSB nibble values are reconstructed. The index of the row is retrieved as an LSB nibble value. This method is considered as the first prediction of LSB nibble value. Also, to get the exact value of the LSB nibble, the MSB nibble value is retrieved for the currently reconstructing pixel. This MSB nibble value is compared with the MSB nibble value of other reconstructed pixels within the block. If the comparison results in a match, then the LSB nibble value of the matching pixel is retrieved. This method is considered as the second prediction of LSB nibble value. If the LSB nibble value obtained in both prediction methods is the same, the value is considered the reconstructed LSB nibble value for the pixel. If the LSB nibble value obtained differs in both prediction methods, the LSB nibble value obtained in the second prediction method is considered. Similarly, LSB nibble values for other pixels of the block are reconstructed. Using the reconstructed MSB and LSB nibbles, the pixels are reconstructed. These reconstructed 16 pixels are stored in the alternative positions of the block of 4X8 pixels. This block is called a partially reconstructed block (PRB) because out of 32 pixels of the block, only 16 pixels are reconstructed.

2) Block differential decoding: The other 16 pixels of the partially reconstructed block are reconstructed by using Block differential decoding. Initially, the Huffman compressed file is decompressed then, proposed method reads the decompressed file to get the first overhead bit to check whether the image block is encoded. If the overhead bit is '0', then the block is not encoded, all sixteen values are read with eight bits each, and the difference block is reconstructed. If the overhead bit is '1' then block is encoded, so the next eight bits are read from the compressed file to get the maximum pixel value of the image block. After reading the maximum pixel value, the proposed method reads the next three bits to find out the number of bits used to represent the difference values of the difference block. By reading the specific number of bits every time, the proposed method gets all sixteen difference values of the difference block. These values are stored in an array as a reconstructed difference block. From the maximum pixel value, every difference value of the reconstructed difference block is subtracted to reconstruct every pixel of the image block. These reconstructed pixels are stored in partially reconstructed block in the order and all pixels of the block are reconstructed. The same procedure is repeated to reconstruct all the blocks of the image.

IV. METHODOLOGY

This section of the paper gives proposed encoding and decoding algorithms and also working of proposed algorithms with example.

A. Encoding

Algorithm 1 provides the steps for proposed compression algorithm.

Algorithm 1:

Input: Input image

Output: Huffman compressed bitstream

```
Start
Step 1: Divide the input image into blocks of 4X8 size.
Step 2: Read blocks in a raster scan manner.
Step 3: Convert each block into the transformed block using mapping transform.
Step 4: Combine all transformed blocks and save the image as a transformed image.
Step 5: Apply Huffman encoding to compress the transformed image into the bitstream.
Step 6: From the 16 pixels of 4X8 block, create block of 4X4 pixels.
Step 7: Find the difference block.
Step 8: Convert difference block into bitstream using variable length coding.
Step 9: Repeat step 8 for all the blocks of the image.
Step 10: Compress the bitstream using Huffman encoder.
Step 11: Save the two Huffman compressed files.
Stop.
```

B. Decoding

Algorithm 2 provides the steps for proposed decompression algorithm.

Algorithm 2

Input Huffman compressed bitstream

Output Reconstructed Image

```
Start
Step 1: First Huffman Compressed file is decompressed using Huffman decoder, and the transformed image is reconstructed.
Step 2: Divide transformed the image into blocks of size 4X8.
Step 3: Transform table for each block is regenerated.
Step 4: Partial block is reconstructed from each transform table.
Step 5: Steps 3 and 4 are repeated for all blocks.
Step 6: Reconstructed blocks are merged to form a partially reconstructed image.
Step 7: Decompress the second file using Huffman decoder and read Bit stream.
Step 8: Convert the bitstream into difference block.
Step 9: From the difference block reconstruct the actual 4X4 block.
Step 10: Pixels from the 4X4 block are stored in blank positions of partially reconstructed block.
Step 11: Repeat steps 7 to 9 for each block and reconstruct complete image
Stop.
```

C. Example

1) Compression: The working of the proposed algorithm is illustrated with an example. An example block considered is shown in Figure 3. In the block, pixels used in mapping transform are shown in Figure 4.

162	162	162	162	161	157	163	163
162	162	162	162	161	157	163	163
162	162	162	162	161	157	163	163
162	162	162	162	161	157	163	163

Fig. 3. Example block

162		162		161		163	
	162		162		157		163
162		162		161		163	
	162		162		157		163

Fig. 4. Sixteen pixels used to compress a block using mapping transform

From the example block, first pixel value 162 is considered. For the pixel, MSB and LSB nibbles are computed. These MSB and LSB nibble values entries are to be made in the transform table. MSB nibble value entry is made in the transform table in position Transform_Table[0][10]. LSB nibble value entry is made in the transform table in position Transform_Table [2][0]. Similarly, other pixels are processed for making an entry in the transform table. The transform table after making all entries is as shown in Figure 5.

The bits in the transformed table shown in Figure 5 are converted into a transformed block of 4X8 pixels by converting the bits into pixel values by combining the bits. The transformed block is shown in Figure 6. The transformed block is further compressed using Huffman encoder.

The other 16 pixels of the example block are compressed using block differential encoding. These pixels are stored using special arrangement in array as shown in Figure 7. The difference block is computed for the block shown in Figure 7 and is as shown in Figure 8. From the difference block, the bitstream is generated. The complete bitstream is compressed using Huffman encoder.

2) Decompression

In the decompression phase, the transformed table regenerated for the example block with row sum and column sum is same as shown in Figure 5. MSB nibble value for the first block pixel is computed by scanning the row with index 0(zero). As the row sum is '1', the MSB nibble value is reconstructed directly by retrieving the column's index, where '1' is present. '1' is present in the column with index 10, so the first pixel's MSB value is 10. The MSB value of the second pixel is computed by scanning the row with index 1(one). As the row sum is two, there is a conflict in choosing the column's index where one is present. Since position one in the column with index 10 is nearer to the column with a maximum sum, index 10 is retrieved as the MSB nibble value of the second pixel.

MSB nibble values

LSB nibble values

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Row sum
0											1						1
1			1								1						2
2	1	1			1	1			1	1	1		1	1			9
3				1				1			1	1				1	5
4											1						1
5											1						1
6										1							1
7											1						1
8											1						1
9											1						1
10											1						1
11											1						1
12											1						1
13							1				1				1		3
14										1							1
15											1						1
Col sum	1	1	1	1	1	1	1	1	1	3	14	1	1	1	1	1	

Fig. 5. Complete transform table for the example block

32	0	32	0	64	0	16	0
32	0	32	0	0	4	16	0
32	0	34	2	253	253	16	0
32	0	32	0	0	4	32	0

Fig. 6. Transformed block for the example block

162	162	157	163
162	162	161	163
162	162	157	163
162	162	161	163

Fig. 7. Other 16 pixels block for the example block

LSB nibble value for the first block pixel is computed by scanning the column with index 0(zero). As the column sum is one, the LSB nibble value is reconstructed directly by retrieving the row index of the position where one is present. The row index 2 contains the one, so the LSB nibble value of the first pixel is 2. The LSB nibble value for the second pixel of the block is computed by scanning the column with index 1(one). Here, column sum is one, and one is present in the row with index 2, so index 2 is retrieved as LSB nibble value for the second pixel. Pixel values are computed by using the MSB and LSB nibble values.

Similarly, 16 pixels of the block are reconstructed and stored in alternative positions of the block. The partially reconstructed block will be the same as shown in Figure 4.

Other pixels of the block are reconstructed using the block differential decoding method. Initially, the compressed bitstream is decompressed by Huffman decoder. Then, the decompression method reads the overhead bit as '1' and comes to know that the block is encoded. Then the next eight bits are read to get 163, the maximum pixel value for the block. After that, the following three bits value is read as '3'. So this value '3' represents that group of three bits from the bitstream make each difference value. So the algorithm reads three bits every time to get sixteen difference values of the difference block. These sixteen difference values are arranged in order and stored in an array as reconstructed difference block is same as shown in Figure 8.

1	1	6	0
1	1	2	0
1	1	6	0
1	1	2	0

Fig. 8. Difference block

The 16 pixels are reconstructed using the reconstructed difference block and the maximum pixel value of the block. The reconstructed 16 pixels block will be same as shown in Figure 7. Thus, these pixels are stored in blank positions of block as shown in Figure 4. The completely reconstructed block for the example is shown in Figure 9.

162	162	162	162	161	161	163	163
162	162	162	162	161	157	163	163
162	162	162	162	161	161	163	163
162	162	162	162	161	157	163	163

Fig. 9. Completely reconstructed block

V. RESULTS AND DISCUSSION

The proposed algorithms are implemented using C++ and Opencv 2.4.10. The proposed algorithm is developed to work on grayscale images. Proposed algorithm performance is evaluated using different grayscale test images of size 512X512 pixels. As the algorithm reads the image and divides the image into 4X8 pixels block, the width of image must be multiple of 8 and height must be multiple of 4. If the image size is not multiple of above said values, then replicate the required number of last columns or rows of the image to become multiple of the above said values. By this modification, proposed algorithm can be made to work on unequal size images.

The test images used Girl, Hill, monument1 monument2, Sea, and Wall are shown in Figure 10. These images are used as test images as these images contain both low and high frequency components. During the compression, the transformed image generated for the test image, and reconstructed image are shown in Figure 11.

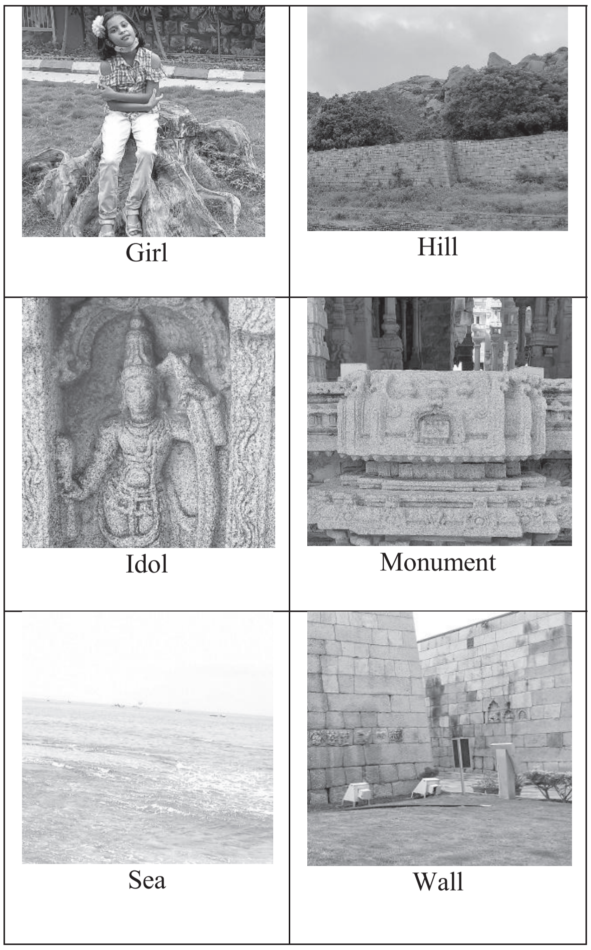

Fig. 10. Test Images

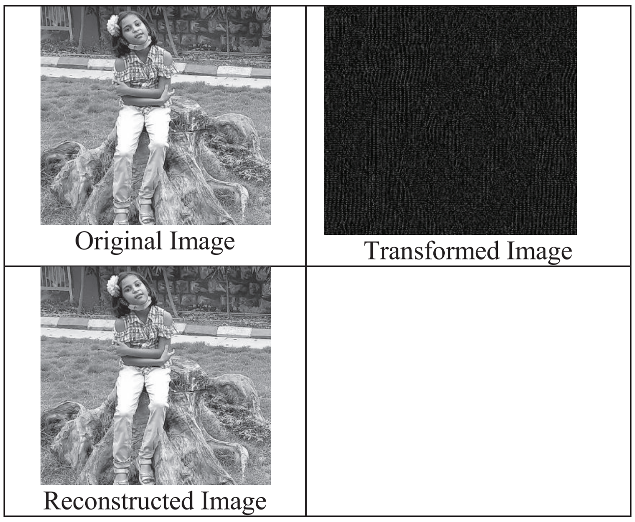

Fig. 11. Original Girl image, transformed image, and reconstructed image

The compressed image sizes and the compression ratios are computed for performance evaluation are depicted in Table 1.

TABLE I. COMPRESSION RATIO, PSNR FOR THE PROPOSED METHOD

Image	Original Image size in bytes	Compressed Image Size in bytes	Compression Ratio
Girl	263222	145538	1.81
Hill	263222	141662	1.86
Idol	263222	152437	1.73
Monument	263222	151818	1.73
Sea	263222	137660	1.91
Wall	263222	142777	1.84

The Figure 12 shows the comparison of proposed method with JPEG-LS. From the figure it can be drawn that the proposed method results are comparable with JPEG-LS but the compression ratio achieved by proposed method is lesser compared to JPEG-LS.

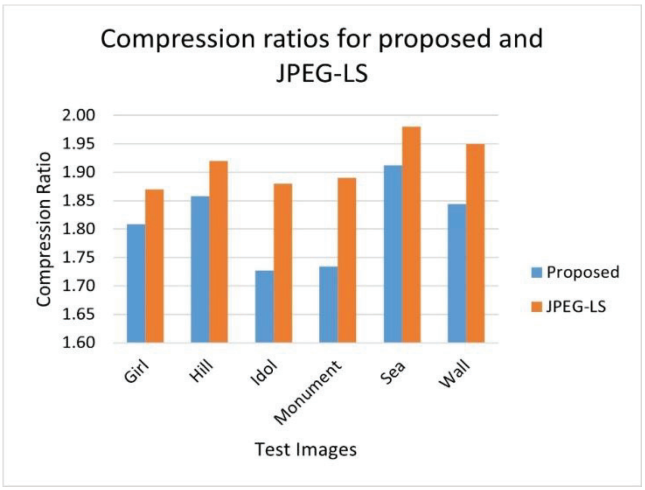

Fig. 12. Comparison of compression ratios of proposed and JPEG-LS method.

The evaluation of the performance of the proposed algorithm in terms of execution time is carried out with different test images. The proposed algorithm is implemented using C++ and Opencv 2.4.10 and is tested on Intel Core i5 with a 2.1 GHz processor, 4GB RAM, and Windows 10 Operating System machine. The results are shown in Figure 13.

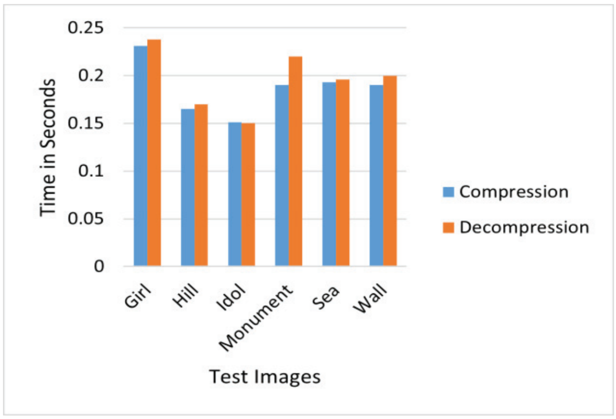

Fig. 13. Comparison of execution time for compression and decompression of proposed method

VI. CONCLUSION

The proposed image compression algorithm has low complexity. To achieve high compression of images, 50% of an image is compressed using mapping transform and Huffman codes and other 50% of image is compressed using block differential encoding method. Mapping of nibbles into a single bit within the table helps generate more zeros to compress. The novel transform is lossless and accurately reconstructs the pixels without any loss. The block differential encoding method is also lossless. The results show that the proposed method results are comparable with JPEG-LS. The algorithm takes less computational time.

REFERENCES

[1] R. Kaur and P. Choudhary, “A Review of Image Compression Techniques,” Int. J. Comput. Appl., vol. 142, no. 1, pp. 8–11, 2016.

[2] Anju and A. Ahlawat, “Performance analysis of image compression technique,” International Journal of Recent Research Aspects, vol. 3, no. 2, pp. 2349–7688, 2016.

[3] R. C. Gonzalez and P. Wintz, Digital Image Processing. Boston, MA: Addison-Wesley Educational, 1978.

[4] V. P. Baligar, L. M. Patnaik, and G. R. Nagabhushana, “High compression and low order linear predictors for lossless coding of grayscale images,” Image Vis. Comput., vol. 21, no. 6, pp. 543–550, 2003.

[5] M. J. Weinberger, G. Seroussi, and G. Sapiro, “LOCO-I: A low complexity, context-based, lossless image compression algorithm,” 1996, pp. 140–149.

[6] G. K. Wallace, "The JPEG still picture compression standard," in IEEE Transactions on Consumer Electronics, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992, doi: 10.1109/30.125072.

[7] Mohammed, F. G., & Al-Dabbas, H. M. (2018). Application of WDR Technique with different Wavelet Codecs for Image Compression. Iraqi Journal of Science, 59(4B), 2128–2134.

[8] S. N. Kumar, A. Ahilan, A. K. Haridhas, and J. Sebastian, “Gaussian Hermite polynomial based lossless medical image compression,” Multimed. Syst., vol. 27, no. 1, pp. 15–31, 2021.

[9] R. Li, Z. Pan, Y. Wang, and P. Wang, “The correlation-based tucker decomposition for hyperspectral image compression,” Neurocomputing, vol. 419, pp. 357–370, 2021.

[10] Mohammed, F. G., & Al-Dabbas, H. M. (2018). “The Effect of Wavelet Coefficient Reduction on Image Compression Using DWT and Daubechies Wavelet Transform.” Science International. 30 (5),757-762, 2018 ISSN 1013-5316

[11] C. Ding, Y. Chen, Z. Liu, and T. Liu, “Implementation of grey image compression algorithm based on variation partial differential equation,” *Alex. Eng. J.*, vol. 59, no. 4, pp. 2705–2712, 2020.