ISSN: 0067-2904

Image Compression Using Mapping Transform with Pixel Elimination

Vishwanath S. Kamatar[1]*, Vishwanath P. Baligar[2]
[1]*Department of Computer Science & Engineering, KLE Institute of Technology, Karnataka, India*
[2]*School of Computer Science, KLE Technological University, Karnataka, India*

Received: 12/8/2021 Accepted: 24/6/2022 Published: 30/9/2023

Abstract

In today's world, digital image storage and transmission play an essential role,where images are mainly involved in data transfer. Digital images usually take large storage space and bandwidth for transmission, so image compression is important in data communication. This paper discusses a unique and novel lossy image compression approach. Exactly 50% of image pixels are encoded, and other 50% pixels are excluded. The method uses a block approach. Pixels of the block are transformed with a novel transform. Pixel nibbles are mapped as a single bit in a transform table generating more zeros, which helps achieve compression. Later, inverse transform is applied in 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 average method is used in reconstruction of excluded pixels. This approach achieves better quality in reconstructed test images at lower PSNR values ranging from 33dB to 44dB. Compression ratio achieved is more than 2. Correctness ratio achieved by proposed method is more than 0.5.

Keywords: pixel elimination, image compression, lossless, mapping transform, lossy, JPEG

1. Introduction

In today's world, digital communication has a high priority in everyone's life, and digital images have become an inevitable part of information in communication. Digital images usually take large storage space, require high bandwidth while transmitting through channels, and require more time to transmit. So it is very indispensable to compress images.

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

In lossy compression, high-quality images are reconstructed, but visual system makes negligible with a loss of certain information. This approach uses transforms like Discrete Cosine Transform [6].

Here, the work proposed is image compression for generic images using a novel transform. Transform is applied to half the number of pixels in the image, and the transformed image data is encoded using a Huffman encoder. Results are compared with results of JPEG compression.

*Email: vishwanath.k@kleit.ac.in

Literature survey outlines different image compression methods in Section 2. Section 3 describes the proposed work and algorithms. In section 4, a new metric called correctness ratio is discussed. In section 5, results are discussed. Section 6 gives conclusion.

2. Related work

This section provides facts and 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. 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 by wavelet 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. Image was compressed by segmentation with quadtree approach, 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.

3. Proposed methodology

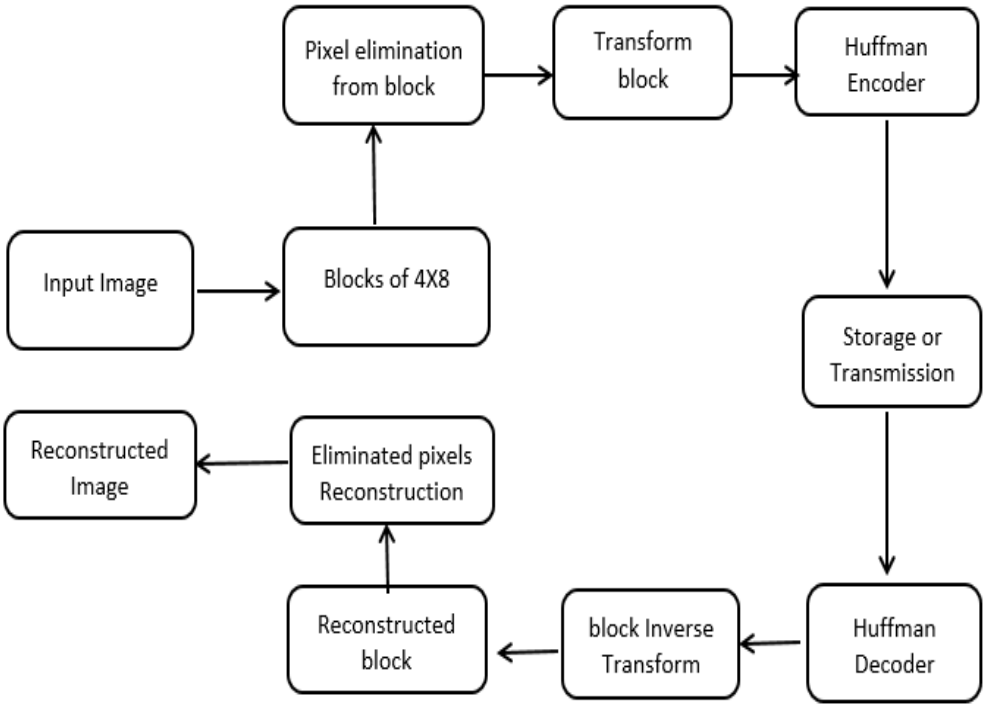

Figure 1: Proposed methodology

A simple lossy image compression algorithm is presented. Figure 1 shows block diagram of the proposed methodology. An algorithm is developed to compress grayscale images. The proposed image compression algorithm compresses an image in terms of blocks. Exactly 50% of pixels are eliminated from each block, and compression is achieved with the remaining 50% of pixels.

3.1 Compression

Input image is divided into blocks of $M \times N$ size. From the block, predefined alternatively positioned pixels are eliminated. The $\frac{M \times N}{2}$ pixels out of $M \times N$ size block with M=4 and N=8, used to compress, are shown as asterisks in Figure 2.

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

Figure 2: Context of pixels from the block

A table of $\left(\frac{M\times N}{2}\right)\times\left(\frac{M\times N}{2}\right)$ is created and referred to as a transform table to transform the pixels. For each pixel 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. MSB and LSB nibble values of the pixel are computed by Eq. 1 and Eq. 2.

$$MSB\ nibble\ value = \frac{P[M,N]}{16} \quad (1)$$

$$LSB\ nibble\ value = P[M,N]\ mod\ 16 \quad (2)$$

The two entries added to the transform table represent pixel's MSB and LSB nibble values. 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 first pixel's MSB value of the block, an entry of '1' is made in *Transform_Table[0][MSB]* position. For the same pixel, entry of '1' for LSB value is made in *Transform_Table[LSB][0]* 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 3.

$$F' = \cup_{u=0}^{4} \cup_{v} \left[F\left[[i, f(u,v)], [g(u,v), i\right]\right] \quad (3)$$

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

To illustrate how the entries are made in the transform table, an example block is considered and shown in Figure 3. In a block, pixels in alternative positions are eliminated, and the pixels used to compress the block 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

Figure 3: Example block

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

Figure 4: Sixteen pixels used to compress a block

From the example block, first pixel value 162 is considered. For the pixel, MSB and LSB nibbles are computed using Eq. 1 and Eq. 2.
MSB of the pixel = 162 / 16 = 10 (MSB)
LSB of the pixel = 162 mod 16 = 2 (LSB)

These MSB and LSB nibble values entries will be inserted into the transform table. So a table of size 16X16 is created with all positions as zeros. MSB nibble value entry is made in the transform table in position *Ttransform_Table[0][10]*. LSB nibble value entry is made in the transform table in position *Ttransform_Table [2][0]*. MSB and LSB nibble values entries in the transform table are as shown in Figure 5. Similarly, other pixels are processed for making an entry in the transform table. The transform table after making all entries is shown in Figure 6.

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

Figure 5: Transform table with an entry of MSB and LSB nibble values

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	

Figure 6: Complete transform table for the example block

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 a chance that some entries in the transform table get overlapped because some MSB and LSB values may fit into exact position of the transform table, and hence 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 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. Algorithm 1 provides details to generate a transformed block from the input image block.

Algorithm 1: Generating transformed block

Input	Input image block of size 4X8
Output	Transformed block of size 4X8

Start
Step1: Read input block and eliminate alternatively positioned pixels from the block.
Step 2: For sixteen pixels of the block compute MSB and LSB nibble values.
Step 3: Make MSB entry in transform table as *Transform table[i][MSB]*=1 and LSB entry
in transform table as *Transform table [LSB][i]*=1
Step 4: Read eight bits of the Transformed table in a column-major manner and store the
pixel value in the transformed block.
Step 5: Save the transformed block with size 4X8.
Stop.

Bits in the transformed table shown in Figure 6 are converted into a transformed block of 4X8 pixels by converting bits into pixel values by combining the bits in the column-major method. The transformed block is shown in Figure 7.

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

Figure 7: Transformed block for the example block

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. Algorithm 2 provides details to generate Huffman compressed bitstream from the input image.

Algorithm 2: Generating Huffman compressed bitstream

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 Algorithm 1. 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. Stop	

3.2 Decompression

During decompression of the image, Huffman encoded bitstream is decompressed using Huffman decoder to reconstruct the transformed image. The algorithm divides the transformed image into 4X8 size blocks. The blocks of the transformed image are processed in a raster scan manner. 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 in the column-major method and the transformed table is regenerated. The steps to regenerate the transform table is given in Algorithm 3.

Algorithm 3: Reconstruction of transform table

Input	Transformed block of 4X8
Output	Transform table of size 16X16
Start	

Step 1: Create a transform table of size 16X16.
Step 2: Read the pixel from the block and convert it into 8 bits.
Step 3: Store the converted bits in the transform table using the column-major method.
Step 4: Repeat steps 2 and 3 for all pixels of the transformed block.
Step 5: Transform table is reconstructed.
Stop.

The transform table column indices represent MSB nibble values, and row indices represent LSB nibble values. Initially, all MSB nibble values are reconstructed. Row sum of the transform table is computed. If the sum of the row is one, then a scan is made in that row to find 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. The considered column will contain the maximum number of MSBs as the nearby pixel values of the block are almost the same. So 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 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. Algorithm 4 provides the details in the reconstruction of the PRB from the transform table. Steps in the reconstruction of the partial image are provided in Algorithm 5.

Algorithm 4: Partial Reconstruction of block

Input	transform table of size 16X16
Output	Partially Reconstructed Block of size 4X8

Start
Step 1: Read the transform table and search for entry one in a row where nearby column sum is more than other column sums.
Step 2: Extract the column index of that entry as MSB value and row index as pixel order

number.
Step 3: Repeat Step 1 and Step 2 to reconstruct MSB values for all 16 pixels of the block.
Step 4: Read the transform table and search for entry one in a column where nearby row sum is more than other row sums.
Step 5: Extract the row index of that entry as LSB value and column index as pixel order number.
Step 6: Repeat Step 4 and Step 5 to reconstruct LSB values for all 16 pixels of the block.
Step 7: Using MSB and LSB values all 16 pixel values of the block are reconstructed.
Step 6: Reconstructed 16 pixels are arranged stored in alternative positions block.
Stop.

Algorithm 5: Partial Image reconstruction

Input	Huffman Compressed file
Output	Partially Reconstructed Image

Start
Step 1: 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 using algorithm 1.
Step 4: Partial block is reconstructed from each transform table using algorithm 2.
Step 5: Steps 3 and 4 are repeated for all blocks.
Step 6: Reconstructed blocks are merged to form a partially reconstructed image.
Stop.

An average method is used to reconstruct the complete image using partially reconstructed image. In partially reconstructed image, reconstructed pixels present alternative positions. The image's 50% non-reconstructed pixel positions are surrounded by four reconstructed pixels in the top, right, bottom, and left positions. The average value of four surrounding pixels is computed. Average value is compared with surrounding four pixels. Pixel value nearer to computed average value is considered a reconstructed pixel value. Similarly, other pixels in the partially reconstructed image are reconstructed. Algorithm 6 gives the steps to reconstruct the complete image.

Algorithm 6: Complete Image reconstruction

Input	Partially reconstructed image
Output	Reconstructed Image

Start
Step 1: Partially reconstructed image is read.
Step 2: For any non-reconstructed pixel $P[x, y]$ compute the average.
Step 3: $Average = \frac{1}{4}(P[x-1, y] + P[x, y-1] + P[x+1, y] + P[x, y+1])$
Step 4: $P[x, y] = Pi$ (if $Pi \approx$ average and Pi is any pixel among four surrounding pixels).
Step 5: Steps 2 to 4 are repeated for all non-reconstructed pixels.
Step 6: Image is saved as reconstructed image.

Stop.

For the above example, the transformed block is shown in Figure 7. Figure 6 shows the transformed table for the block with same row and column sum in decompression phase. 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 equals two, there is a conflict in choosing 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.

LSB nibble value for the first block pixel is computed by scanning the column with index 0(zero). As the column sum equals 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 equals 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 Eq. 4 using the MSB and LSB nibble values.

Pixel value = pixel MSB value * 16 + pixel LSB value (4)

First pixel = 10 * 16+ 2 = 162

Second pixel = 10 * 16+ 2 = 162

Similarly, 16 pixels of the block are reconstructed and stored in alternative positions of the block. The partially reconstructed block will be same as shown in Figure 4. Other block pixels are reconstructed using the average method, and a completely reconstructed block for the example considered is shown in Figure 8.

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

Figure 8: Completely reconstructed block

4. Correctness ratio as a new quality metric

MSE and PSNR are mainly used metrics in assessing quality of reconstructed images. The PSNR is computed by Eq. 5, and MSE is computed by Eq. 6. The input image is represented as $I_{0,}$ and the reconstructed image is represented as $I_{1.}$

$$\text{PSNR}= 10log\left(\frac{255^2 I_0 I_1}{\|I_0-I_1\|^2}\right) dB \quad (5)$$

$$\text{MSE} = \frac{\|I_0-I_1\|}{I_0 I_1} \quad (6)$$

However, here is a new metric that provides a better method in testing quality of reconstructed images. It is called correctness ratio and is computed by Eq. 7.

$$Correctness\ Ratio\ = \frac{\text{accurately reconstructed pixels count in reconstructed image}}{\text{original image pixels count}} \quad (7)$$

Accurately reconstructed pixels count in the reconstructed image is directly proportional to the correctness ratio. The correctness ratio increases when the reconstructed image's accurately reconstructed pixels count increases.

5. Results and discussion

The proposed algorithms described in previous section 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. However, it is possible for proposed algorithm to work on images with unequal sizes. As the algorithm reads an image and divides the image into 4X8 pixels block, 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 work on unequal size images.

The example test image used is Barbara, and during the compression, the transformed image generated for the test image, partially reconstructed image, and reconstructed image are shown in Figure 9. The histograms for original and reconstructed images are shown in Figures 10 (a) and (b), respectively. The reconstructed image histogram is almost similar to the original image histogram as more than 50% of the pixels are accurately reconstructed.

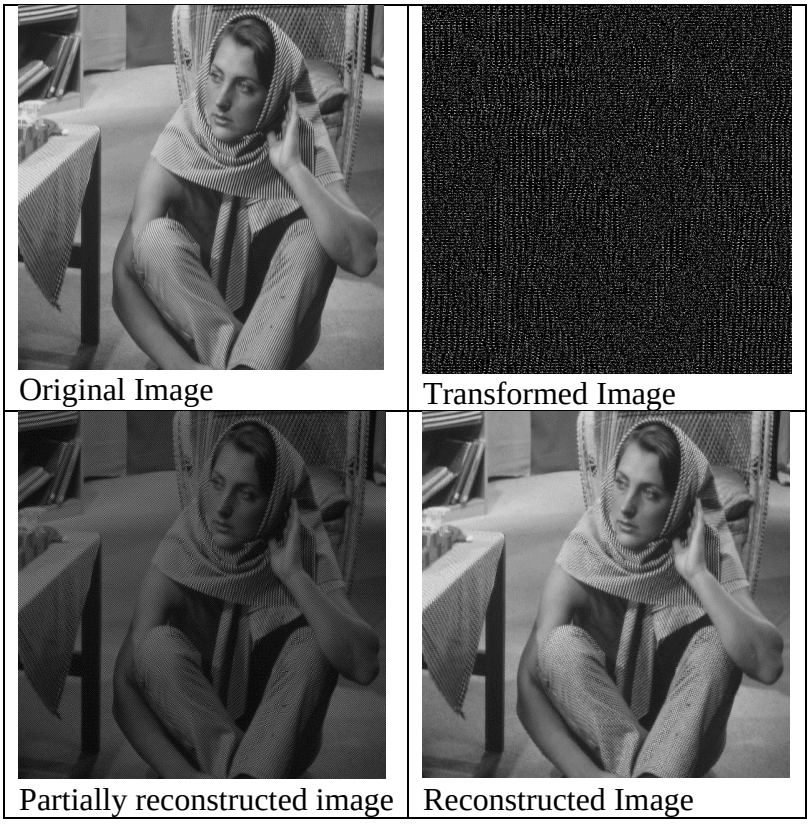

Figure 9: Original Barbara image, transformed image, and reconstructed image

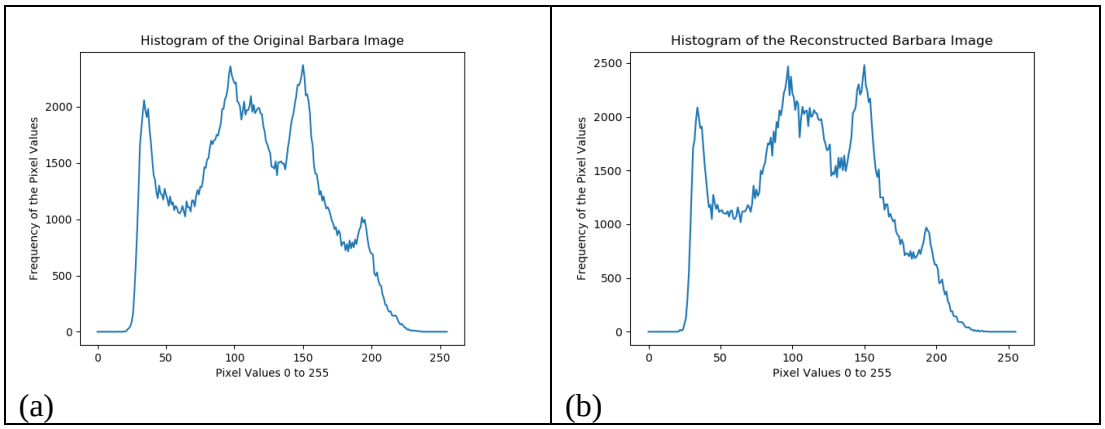

Figure 10: Histograms (a) Original image (b) Reconstructed image.

5.1. Performance and comparison of accurately reconstructed pixels

The compressed image size and the compression ratio are computed for performance evaluation. The PSNR of reconstructed images is also computed. These computed values are depicted in Table 1.

Table 1: Compression Ratio, PSNR for the proposed method

Image	*Compression Ratio*	*PSNR in dB*
Baboon	1.93	32.84
Barbara	2.03	34.98
Cameraman	2.20	40.01
House	2.35	43.59
Jetplane	2.13	38.9
Lena	2.06	37.78
Living room	2.01	36.22
Peppers	2.13	36.9
Bridge	2.05	34.75
Aerial	2.01	34.95
Truck	2.09	37.06
Tanker	2.06	36.23
Boat	2.03	35.57

The quality of the reconstructed images from the proposed method and JPEG method are compared. The PSNR value is computed for the reconstructed images from the proposed algorithm. JPEG images with the same PSNR are obtained by adjusting the quality. By keeping the PSNR of images constant in both methods, and accurately reconstructed number of pixels as the original image in both methods is computed. Table 2 shows the results.

5.2. Comparison by Correctness Ratio

The correctness ratio is computed for both methods using the accurately reconstructed number of pixels from Table 2. The increase in the accurately reconstructed pixels by the proposed method compared to the JPEG method and the correctness ratio obtained from both methods is given in Table 3. From the correctness ratio of the proposed method, it is clear that the quality of the reconstructed images by the proposed method is superior.

Table 2: Comparison of accurately reconstructed pixels at same PSNR for Proposed and JPEG method

Image	*PSNR in dB*	*Accurately reconstructed Number of pixels same as that of Original*	
		Proposed	***JPEG***
Baboon	32.84	136278	19163
Barbara	34.98	143635	30937
Cameraman	40.01	173379	57587
House	43.59	191603	111751
Jetplane	38.9	154891	49493
Lena	37.78	147913	39053
Living room	36.22	144151	30335
Peppers	36.9	143806	33187
Bridge	34.75	161661	27085
Aerial	34.95	145722	31261
Truck	37.06	151762	33712
Tanker	36.23	151869	29240
Boat	35.57	141238	27723

Table 3: Comparison of correctness ratio for Proposed and JPEG method

Image	*Increased number of accurately reconstructed pixels as per proposed method than JPEG*	*Correctness Ratio of proposed*	*Correctness Ratio of JPEG*
Baboon	117115	0.52	0.07
Barbara	112698	0.55	0.12
Cameraman	115792	0.66	0.22
House	79852	0.73	0.43
Jetplane	105398	0.59	0.19
Lena	108860	0.56	0.15
Living room	113816	0.55	0.12
Peppers	110619	0.55	0.13
Bridge	134576	0.62	0.10
Aerial	114461	0.56	0.12
Truck	118050	0.58	0.13
Tanker	122629	0.58	0.11
Boat	113515	0.54	0.11

5.3. Comparative Study of Compressed Image size

Comparative analysis is made for the compressed file sizes from proposed and JPEG methods. Comparison is made based on the correctness ratio. In the comparison, the accurately reconstructed number of pixels from the proposed method is considered constant. JPEG image quality is adjusted such that, JPEG image contains the same number of accurately reconstructed pixels as that of the proposed method. The image sizes from both methods are compared. In the comparison, the accurately reconstructed number of pixels of images in both methods is constant. Results are shown in Figure 11.

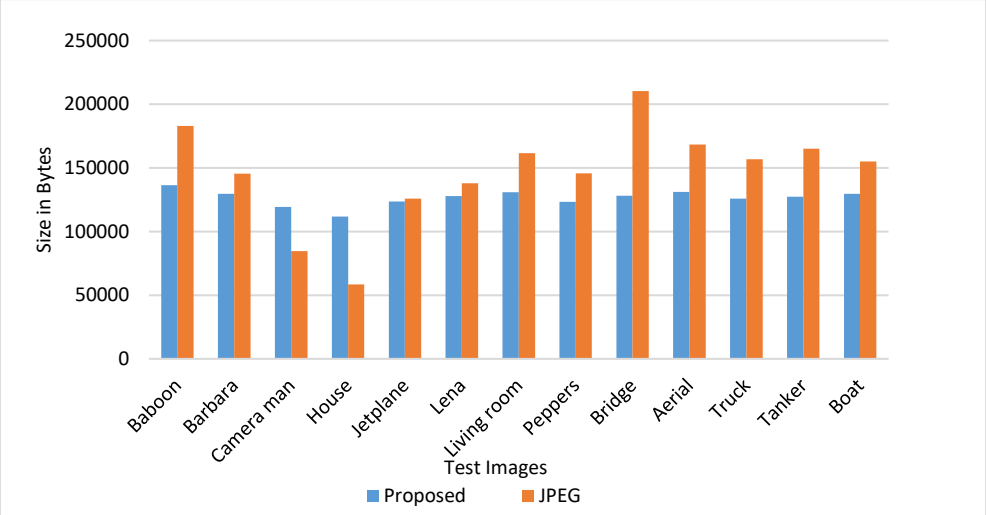

Figure 11: Image size comparison from both methods at the same accurately reconstructed number of pixels

5.4. Computational Cost

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 12.

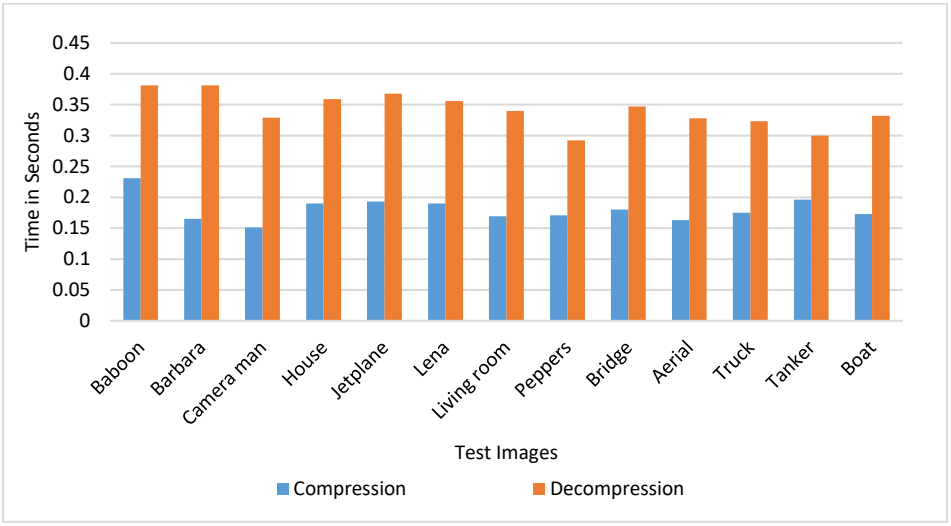

Figure 12: Comparison of execution time for compression and decompression of proposed method

6. Conclusion

The proposed image compression algorithm has low complexity. To achieve high compression of images, half of the pixels from an image are excluded from encoding. Mapping 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 excluded pixels in the encoding are reconstructed with the average method. The results show that the reconstructed image quality obtained from the proposed algorithm is good. The correctness ratio justifies that reconstructed image quality is good. The compressed image size from the proposed method is less than the JPEG image when both methods contain the same accuratel

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., "Application of WDR Technique with different Wavelet Codecs for Image Compression," *Iraqi Journal of Science*, vol. 59, no. 4B, pp. 2128–2134, 2018.

[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., "The Effect of Wavelet Coefficient Reduction on Image Compression Using DWT and Daubechies Wavelet Transform," *Science International*., vol. 30 no. 5, pp. 757-762, 2018 ISSN 1013-5316, 2018.

[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.