International Journal of Computing and Digital Systems
2025, VOL. 17, NO.1, 1-11

http://dx.doi.org/10.12785/ijcds/1571019981

Using DCT and Quadtree for Image Compression

Chadi F. Riman[1] and Pierre E. Abi-Char[1]

[1]*College of Engineering and Technology, American University of the Middle East, Kuwait*

Received 16 April 2024, Revised 31 August 2024, Accepted 3 October 2024

Abstract: Image compression has been used for a long time to reduce image file size and became a necessity with the introduction of the world wide web since images are used extensively in web sites. Bitmap images are very popular today, but they use a lot of memory space, giving the need for compressing them. Various techniques are used for bitmap (BMP) image compression. Several are lossless, that is the quality of the image is not modified, and others are lossy, in which a part of the image is lost. Examples of lossless techniques include GIF and PNG, while JPG is an example of a lossy technique. JPG is composed of Discrete Cosine Transform (DCT), Quantization, Huffman Coding (HC) and Run-Length Encoding (RLE). Quadtrees are also used for lossless or lossy image compression. In this work, we proposed an algorithm based on DCT/Quantization and Quadtrees to be used in sequence together for lossy image compression. Our proposed method is compared with other techniques, namely JPEG and Quadtree with different parameters. In the results, our proposed algorithm performed well compared to other quadtree methods.

Keywords: Bitmap/JPEG image formats, DCT algorithm, Image compression algorithms, Quadtree decomposition, Quantization

1. INTRODUCTION

Computer images are an essential part of computer daily usage, especially for end consumers. They are identified with different color models such as RGB, CMY, YCbCr. RGB means that every pixel has 3 different colors, one for Red, one for Green, and one for Blue. CMY means that the image is composed of the colors cyan, magenta and yellow. YCbCr represents the image in terms of luma or brightness Y (which gives the grey image alone), Cb (blue-difference chroma B-Y), and Cr (red-difference chroma R-Y). The difference between RGB and YCbCr is that RGB represents colors as combinations of red, green and blue. On the other hand, YCbCr represents colors as combinations of a brightness and two chroma parts [1].

There are several formats to save a picture in the computer such as Raw format as taken from digital cameras or scanners, and Bitmap (BMP) format that was introduced by Microsoft for its Windows operating system in the 1980s. These formats usually consume a lot of memory to save an image, typically 3 bytes per pixel for BMP with 24 color bits, namely 8 bits (256 different values) per color: R (red), G (green), and B (blue) [2].

Data file compression was introduced in the 1970s to reduce file size files for storage purposes. Among the different techniques used, Huffman coding was one of them, followed by LZW algorithm which was widely used for most general-purpose compression systems [3].

With the widespread of World Wide Web in the early nineties, and with the initial slow network speeds, there was a need to reduce the size of an image so that it will be downloaded on a client's computer in reasonable time [4]. Several image compression techniques were introduced to solve this issue. They were categorized in 2 categories: lossless and lossy [5]. The lossless methods compress the image without losing any part of it, making it easy to decompress and retrieve all the bits. Examples of lossless image compression formats include Graphics Interchange Format (GIF) in 1987, and Portable Network Graphics (PNG) in 1997. The lossy methods compress the image while losing a part of it, making it impossible to retrieve the original image bits. Using lossy methods can be noticeable in the deterioration of the image's quality, but sometimes it is not noticeable by a human's eye. The most important type of lossy image compression format is Joint Photographic Experts Group (JPEG) released in 1992. The JPEG format gives an excellent compression while maintaining a good quality. JPEG uses a lossy form of compression based on the discrete cosine transform (DCT), quantization, run-length encoding, and then Huffman coding. Another way is to reduce the image resolution in any format (for example downsizing a 256x256 pixels BMP to 64x64) which also results in image size reduction, but with a big loss of quality

E-mail address: chadi.riman@aum.edu.kw, pierre.abichar@aum.edu.kw

[6], [7].

A quadtree is a type of tree data structure in which each internal node has exactly 4 sub-nodes. It is used to store information for a two-dimensional space [8]. Quadtrees are also used for image compression in both lossless and lossy algorithms, in which an image keeps being divided into 4 equal regions until each region values reach a certain threshold [9].

In this paper, we present an improvement to the original quadtree image compression by combining it with part of the process done in JPEG compression. The quadtree is created after performing the DCT and Quantization of the original image, with the latter two being part of JPEG compression. Computer experiments for different standard images are conducted to evaluate the performance of the proposed algorithms. The simulation results are also compared to other methods mostly using Quadtrees to show the effectiveness of the proposed system, especially when compared with the standard quadtree implementations.

The rest of this paper is organized as follows. In Section 2, we briefly survey the relevant literature review. Section 3 presents mathematical preliminaries. Section 4 describes the proposed DCT and Quantization with the Quadtree algorithm. In section 5, the performance of the proposed scheme is analyzed and compared to other methods. Finally, we conclude the paper in Section 6 with a perspective of the obtained results.

2. RELATED WORK

In the past years, several image compression approaches have been proposed in both lossless and lossy ways. Furthermore, several works covered using Quadtrees for compression of data and images. A few of the relevant recent work done is listed next.

Authors in [10] suggested a new hybrid image compression technique. Three transform-based techniques discrete Fourier transform (DFT), discrete wavelet transform (DWT), and discrete cosine transform (DCT) have been combined for image compression to combine the good characteristics of these methods. To test the level of compression, quantitative measures were used to test the compression level and the effectiveness of the suggested system.

In [11], the JPEG algorithm was improved by storing the location of end-of-block codes for empty blocks in a separate buffer and compressing the buffer with a lossless method (Huffman or arithmetic coding), used for all the image data. This way, with the same Peak Signal to Noise Ratio value, a higher compression ratio than the conventional JPEG encoder resulted. The improvement amount varied between images.

A work in [12] suggested a way to improve PEG compression algorithm for color image. It tried to improve compressed image quality by modifying luminance quantization table in frequency domain. The quantization table is modified in a way to only keep the lowest frequency DCT coefficients with significant amplitude values. The proposed method was verified with simulation results.

Another work in [13] also suggested adding pre-processing steps that could be generalized to be implemented before any lossy technique and was performed on various images that varied in types, dimension, and bit-depth. JPEG was one of the used techniques along the proposed pre-processing steps. The results showed the advantage of combining pre-processing with JPEG in terms of enhanced peak signal-to-noise ratio (PSNR) and better compression ratio.

JPEG2000 was released in 2000 as an improvement over JPEG original algorithm. In [14], a comparison was made between the two. The main difference is that JPEG2000 used discrete wavelet transform (DWT) instead of discrete cosine transform (DCT). It also used Embedded Block Coding with Optimal Truncation (EBCOT) instead of Huffman coding. The results showed that JPEG had a slight quality edge at low compression ratios (below 20:1), while JPEG2000 was better at medium and high compression ratios.

In [15], a new compression concept based on convolutional neural networks (CNN) was suggested. To get high-quality image compression at low bit rates, two CNNs are combined into the compression algorithm. The two CNN cooperate and are trained using a unique optimization procedure. Results from the testing gave enhanced performance and quicker processing.

A work done in [16] proposed a method for lossy image compression based convolutional neural networks that is claimed to be better than JPEG. Three techniques were shown: hidden-state priming, spatially adaptive bit rates, and perceptually-weighted training loss. Combining these three techniques gave an improvement over standard image codecs such as JPEG.

The authors in [17] tried to improve the performance of JPEG2000 by combining it with features of the original JPEG method. They proposed a compression method in which both DWT and DCT were used to improve the compression ratio. Then removed the EBCOT used in JPEG2000 that has high computational complexity and replaced it with the simpler Huffman coding. This saved time and computations in addition to a better compression ratio.

The study done by the authors in [18] proposed hybrid model the using canonical Huffman coding (CHC) with discrete wavelet transform (DWT) and principal component analysis (PCA) for image compression. When compared to the existing approaches, the reconstructed images gave a better peak signal-to-noise ratio which reflects to a better image quality. It also gave lower bit rates which means a better compression factor.

The work done in [19] proposed a fast lossless Image Compression RVL-based system for real time compression to give high ratio. Each pixel in the picture uses a number of bits, called bit depth. The system was able to get a large compression ratio within a very strict time constraint since the system has to work in real time.

A work done in [20] suggested an improved lossless image compression algorithm. It combined linear prediction, integer wavelet transform (IWT) with output coefficients processing and Huffman coding. Due to the big correlation between the adjacent pixels of the source image, there is a large compression space. The suggested two-dimensional linear prediction model reduced the redundancy between the pixels, so that the error values are concentrated near zero and the data complexity is reduced. The overall system gave a higher compression rate with acceptable speed, mostly due to a reduced predicted image entropy.

In [21] and [22], a new image compression technique is proposed using quadtree decomposition with variable block size for coding images. Inactive blocks are coded by the block average value, while active blocks are coded by using a set of parameters according to a pattern inside the block. The goal is to achieve high compression ratios and preserve image quality.

Another method [23] used fuzzy sets based on type-2 fuzzy logic for the quantization control used in discrete cosine transform (DCT). An image coding system was also provided with the fuzzy optimization algorithm, in which the fuzzy rules of the gain factor used in image compression were built.

Fuzzy logic was also used with Huffman coding in [24] Coding is based on Huffman code with fuzzy logic-based weighting functions for the frequency of the existing symbols in data to generate efficient compression code. Fuzzy logic is then coded to provide data security under symmetric key encryption and decryption.

The work in [25] used DCT with quantization for image compression. The image was divided into blocks of 8x8 pixels, then the algorithm was applied to each block.

Another work in [26] compared the use of discrete cosine transform (DCT) to the use of discrete wavelet transform (DWT) for image compression. In both methods, the image is converted to grey scale, then divided into 8x8 pixel blocks. In the DCT way, DCT and quantization were applied. In the DWT way, DWT and reconstruction filter based on a threshold were applied.

A work in [27] suggested using DCT with Huffman encoding for image compression of medical Magnetic resonance images (MRI).

The authors of [28] worked on a smart fuzzy method using Quadtrees to perform image compressions. An image was divided into 16x16 blocks before compression.

The work explained by the authors in [29] combined the use of DWT and Quadtrees together for image compression. The last step of the compression involved Huffman coding as well. The result gave a good peak signal to noise ratio (PSNR).

3. MATHEMATICAL PRELIMINARIES

In this section, we describe how JPEG image compression works in detail. Also, we explain what is a quadtree and how it is used for image compression in both lossless and lossy ways. For the purpose of reducing complexities, we will assume working with grey images, thus having equal components for RGB images, or just the Lumen part of YCbCr images

A. *JPEG Image Compression*

The JPEG image compression format gives an excellent compression while maintaining a good quality. JPEG uses a lossy form of compression based on the discrete cosine transform (DCT), quantization, run-length encoding (RLE), and then Huffman coding [6].

1) *Discrete Cosine Transform (DCT)*

DCT shows a sequence of data points as a sum of cosine functions oscillating at different frequencies. It is a method to convert a signal into elementary frequency components. To apply DCT, the image is divided first into 8×8 blocks. For each block, DCT is applied after modifying each color range to be [-128, 127] instead of [0, 255] by subtracting 128 from each element. DCT for each cell (i, j) uses the following simplified equation for the standard 8x8 blocks that JPEG uses [7]:

$$D(i,j) = \frac{1}{4}C(i)C(j)\sum_{x=0}^{7}\sum_{y=0}^{7} p(x,y)cos\frac{(2x+1)i\pi}{16}cos\frac{(2y+1)j\pi}{16} \tag{1}$$

Where p(x, y) is the original pixel element (x, y) in matrix p, and C(x) is $\frac{1}{\sqrt{2}}$ if x=0, and 1 otherwise.

From the above equation, we get a matrix form T which is easier for calculation. So that the DCT of 8x8 block is calculated by:

$$D = TMT' \tag{2}$$

where T is the DCT matrix, T' is the transpose of T, and M is the modified image block to the color range [-128,127]. $T(i, j)$ has the equation:

$$\begin{cases} \frac{1}{\sqrt{8}} & \text{if i=0} \\ \frac{1}{2}cos\frac{(2j+1)i\pi}{16} & \text{if } i > 0 \end{cases} \tag{3}$$

2) *Quantization*

After DCT, quantization is applied to compress the 8x8 DCT block D calculated in equation 2. Every element in

D is divided by a corresponding element in quantization matrix Q. There are different levels of compression of Q. The most used one, which is the best in terms of preserving image quality and high level of compression is Q_{50}. The resulting matrix C has the following equation:

$$C_{i,j} = round(D_{i,j}/Q_{i,j}) \tag{4}$$

In C, it is noticed that many cells on right and lower parts consist of zeros, which is due to the compression effect done by Q.

3) *Run-Length Encoding (RLE)*

After quantization, the matrix C is read in a zigzag order starting from the upper left corner, grouping repeated numbers together, especially the zeros. A sequence of the same number is stored as a single count and data value.

4) *Huffman Coding*

After run-length encoding, Huffman coding is applied to the block. All numbers in the block are replaced with codes having variable sizes. A more frequent number will have a smaller coding size than less frequent numbers.

B. *Quadtree Image Compression*

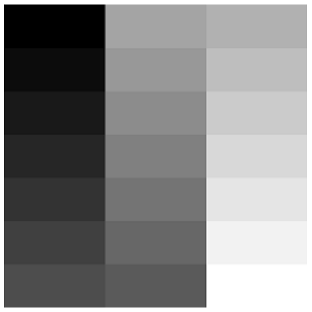
A quadtree is a type of tree data structure where each node is a leaf node or has exactly four children. This data structure is used to store information in a two-dimensional space. [8] This is very convenient for saving images because a planar image has two dimensions. A quadtree can be used for image compression in both lossless and lossy ways. A tree node is created to represent the full image. The difference of individual pixels and the average RGB color of an image is checked against an error threshold. If the difference is bigger than the threshold, the image is divided into 4 sub regions, and the node gets 4 children. The process is repeated recursively until it meets the threshold. The image values are saved in the leaf nodes. If the threshold is zero, then the compression is lossless, otherwise it is lossy. The end tree should be smaller than the original image [9].

To illustrate the above, 2 examples are done, lossless example with threshold of zero, and another lossy example with threshold of 10.

For the first example, a small portion 8x8 pixel block is taken from an image containing 21 blocks of grey color shades (Fig. 1a). The portion lies on the border between 2 different zones of grey. The threshold is set to be zero, which means a lossless compression case. This block can be divided into 22 regions using the zero threshold as shown in Fig. 2b.

Figure 3 shows the quadtree for the first example. The letter ‘x’ represents a node without a final value, but having 4 children nodes. In this example, the 22 regions give a compression ratio of 2.91 to 1 excluding the empty ‘x’ nodes, and 2.21 to 1 including the ‘x’ nodes.

For the second example, an 8x8 pixel block (shown in Fig. 4a) is taken from LENA grey image of size 256x256


(a) 21 shades of GREY image

(b) LENA grey image

Figure 1

90	90	90	90	90	90	90	90
90	90	90	90	90	90	90	90
90	90	90	90	90	90	90	90
96	96	96	96	96	96	96	96
103	103	103	103	103	103	103	103
103	103	103	103	103	103	103	103
103	103	103	103	103	103	103	103
103	103	103	103	103	103	103	103

(a) 8x8 sample pixel values taken from 21 shades of grey image

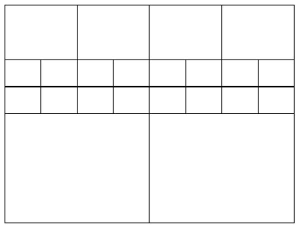

(b) 8x8 sample, with quadtree regions for zero threshold

Figure 2

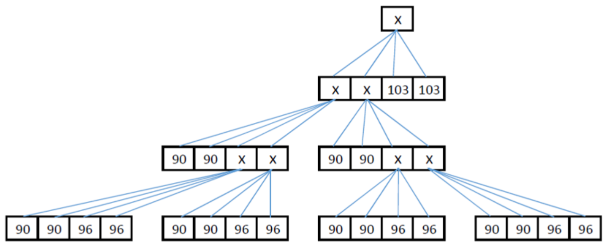


Figure 3. The quadtree for the 8x8 GREY sample, taken from figure 2a

(Fig. 1b), which is a standard used picture for image processing. We assume the threshold to be 10, so the compression is lossy. The rounded average of the cell values is 130. The deducted average from the cell values is shown in Fig. 4b. This block can be divided into 25 regions using the threshold 10, as shown in Fig. 4c.

Figure 5 shows the quadtree for the previous example. The letter ‘x’ represents a node without a final value, but having 4 children nodes. In this example, the 25 regions give a compression ratio of 2.56 to 1 excluding the empty ‘x’ nodes, and 1.94 to 1 including the ‘x’ nodes.

4. PROPOSED ALGORITHM

A. *Algorithm’s Decomposition*

The proposed scheme consists of three main parts, namely the Discrete Cosine Transform, the Quantization,

127	132	126	133	134	130	138	139
123	129	127	133	134	132	138	138
117	126	127	133	134	134	138	138
110	122	126	130	132	132	134	139
108	122	128	130	132	133	133	140
108	123	132	132	133	135	132	141
106	122	131	130	131	133	130	142
108	122	128	131	129	130	138	136

Figure 4a. 8x8 sample pixel values taken from LENA grey image

-3	2	-4	3	4	0	8	9
-7	-1	-3	3	4	2	8	8
-13	-4	-3	3	4	4	8	8
-20	-8	-4	0	2	2	4	9
-22	-8	-2	0	2	3	3	10
-22	-7	2	2	3	5	2	11
-24	-8	1	0	1	3	0	12
-22	-8	-2	1	-1	0	8	6

Figure 4b. 8x8 sample pixel values minus calculated average value of 130

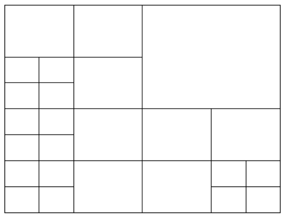


Figure 4c. 8x8 sample, with quadtree regions for threshold 10

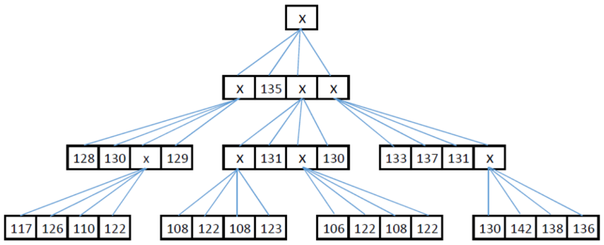


Figure 5. The quadtree for the 8x8 LENA sample, taken from figure 4a

and the Quadtree decomposition. These three parts are described as follows.

We Assume that the original image is 256x256 pixels. The image is initially decomposed to 8x8 pixel blocks, then the color range for each pixel is changed to be around zero, by deducting 128 from each number as explained in the previous section. After the initial centering around zero, the DCT is applied first similarly as applied to JPEG algorithm. Then DCT is applied to every block using equations (2) and (3) mentioned in the previous section. Next, the quantization step is applied also in the same way as in JPEG. Q_{50} is used as a compromise between having a good quality and a good compression ratio. The last step involves creating a quadtree after merging all the 8x8 blocks into the full image (which is assumed to be 256x256 pixels, with 8 bits gray color depth). The full image is checked for a single pixel value from the image. A quadtree node is created depicting this value. If a single value does not exist, then the image is divided into 4 equal blocks, with each block tested again for uniformity, assuming a threshold of zero. On the quadtree, this is done by adding 4 children to the parent node. This process keeps running recursively until no further decomposition can be done. Finally, the leaf nodes represent the actual parts of the image

B. Algorithm's Flowchart and Pseudo-Code

A simplified flowchart for the overall system is shown on the next figure 6. The proposed algorithm's pseudocode is shown after that.

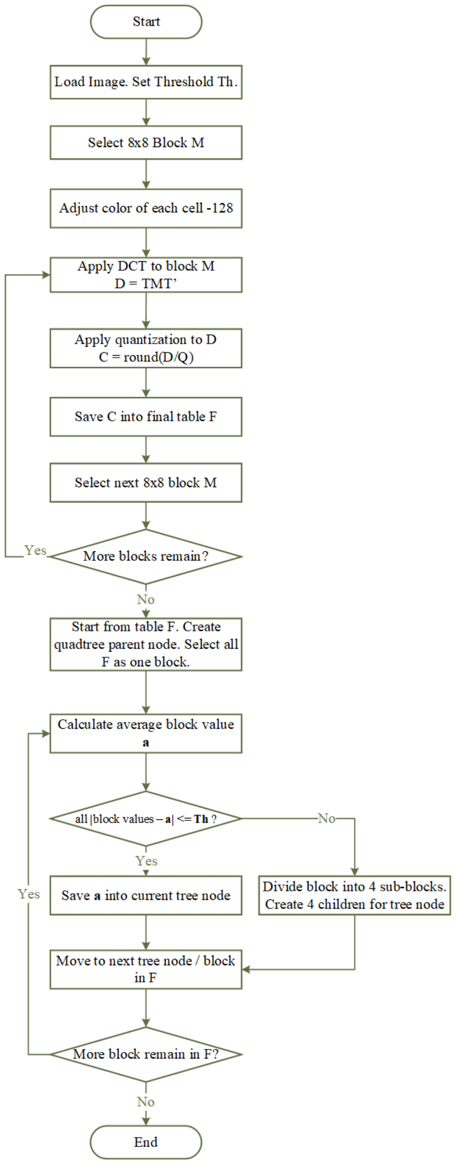

Start
Load Image. Set Threshold Th.
Select 8x8 Block M
Adjust color of each cell -128
Apply DCT to block M
D = TMT'
Apply quantization to D
C = round(D/Q)
Save C into final table F
Select next 8x8 block M
More blocks remain?
Yes
No
Start from table F. Create quadtree parent node. Select all F as one block.
Calculate average block value a
all |block values – a| <= Th ?
No
Yes
Save a into current tree node
Divide block into 4 sub-blocks. Create 4 children for tree node
Move to next tree node / block in F
More block remain in F?
Yes
No
End

Figure 6. DCT–Quadtree Image Compression Algorithm's Flowchart

Algorithm 1 DCT – Quadtree Image Compression

```
 1: Select the 256x256 grey image.
 2: Set threshold value **Th**.
 3: Divide the image into 8x8 blocks.
 4: while there are remaining 8x8 blocks do
 5:     Select the next 8x8 block M.
 6:     Modify the color of each pixel in M:
        $M(i, j) = M(i, j)–128$ (i/j going from 0 to 7)
 7:     Apply DCT to block M, getting matrix block D by computing:
        $D = TMT'$, where:
        $T(i, j) = \frac{1}{\sqrt{8}}, if i = 0, \frac{1}{2} cos\frac{(2j+1)i\pi}{16}, if i > 0$ and
        $T' = Transpose of T$
 8:     Apply quantization $Q_{50}$ to block D, getting matrix C by computing:
        $C_{i,j} = round(D_{i,j}/Q_{i,j})$
 9:     Save block C as part of 256x256 final table F.
10: end while
11: Start from table F as 1 block, create one parent tree node and set as current node.
12: while more node/blocks exist do
13:     Check the average block value a.
14:     If all |blockvalues–a| <= Th, save ‘a’ into the current tree node.
15:     If any absolute block value is greater than (Th - a), divide the block into 4 sub-blocks. Create 4 children for the current tree node.
16:     Move to the next tree node/block in F (movement is left-right-up-down).
17: end while
18: End of Algorithm.
```

C. Computational Complexity

In order to calculate the proposed algorithm's computational complexity, we need to calculate the complexity of each part, namely DCT transformation, Quantization, and Quadtree building. Then we choose the maximum value for the overall algorithm.

The computational complexity for 2 dimensional DCT transformation is $O(n^2.log(n))$, as explained in [30]. The computational complexity for Quantization is $O(n^2)$, but can be reduced to $O(n)$ as shown in the work in [31]. The computational complexity for building a Quadtree is $O(n.log(n))$ according to [32].

The overall computational complexity is:

$$O(n^2 log(n)) + O(n) + O(nlog(n))$$

which results to a final value of $O(n^2 log(n))$, the highest of the three parts.

D. Numerical Example

As an example, we will apply the same sample taken from LENA grey image in figure 4a. Figure 7 shows the updated numbers from figure 4a after deducting 128 from each cell.

$$M(i, j) = M(i, j) - 128 \tag{5}$$

-1	4	-2	5	6	2	10	11
-5	1	-1	5	6	4	10	10
-11	-2	-1	5	6	6	10	10
-18	-6	-2	2	4	4	6	11
-20	-6	0	2	4	5	5	12
-20	-5	4	4	5	7	4	13
-22	-6	3	2	3	5	2	14
-20	-6	0	3	1	2	10	8

Figure 7. 8x8 sample pixel values taken from LENA grey image (fig. 4a), with numbers centered around zero

After initially centering the matrix values around zero, the DCT is applied to the 8x8 block from figure 7 and the result is shown next in figure 8, with all numbers rounded off to one decimal place.

12.8	-51.5	-15.2	-20.8	-4	-8	-5.3	-2.7
11.7	10.3	8.9	10.1	5.6	2.9	-6.5	0.3
4.6	6.6	5.2	2.1	0.7	3.5	-4.3	0.7
2.6	1.8	0.5	-2	1.6	-3.9	3.1	-0.6
-3.8	-0.3	1.9	0.7	0	1.8	-3.2	0.7
-1.2	-0.7	0.1	-1.5	1.1	-2.5	1.8	-0.6
2.7	0.6	-0.6	1.2	-0.8	1.9	-1.5	0.1
0.2	-0.3	0.3	0	0.1	-0.6	0.2	-0.3

Figure 8. Results of applying DCT to the 8x8 sample pixel values in fig. 7

The DCT of 8x8 block is calculated by:

$$D = TMT' \tag{2}$$

where T is the DCT matrix, T' is the transpose of T, and M is the modified image block to the color range [-128,127]. T has the equation:

$$\begin{cases} \frac{1}{\sqrt{8}} & \text{if i=0} \\ \frac{1}{2} cos\frac{(2j+1)i\pi}{16} & \text{if } i > 0 \end{cases} \tag{3}$$

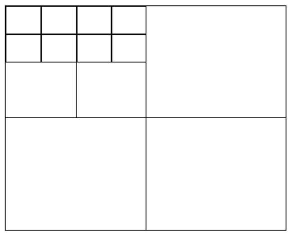
Next, tthe resulting block matrix from the first step (Fig. 8) is divided by the quantization matrix Q_{50}, using equation (4) from previous section. The results of applying quantization to the 8x8 block in figure 8 (after DCT) is shown next in figure 9. As mentioned in equation (4), all numbers are rounded to the nearest integer.

$$C_{i,j} = round(D_{i,j}/Q_{i,j}) \tag{4}$$

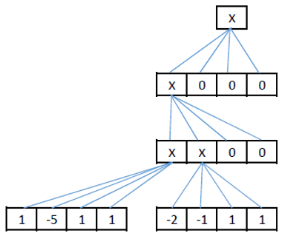
We apply the last step of creating a quadtree after merging all the 8x8 block values received after applying DCT, then quantization (Q_{50}). We also assume a threshold value of zero. The obtained quadtree is shown in the next figure 10.

1	-5	-2	-1	0	0	0	0
1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 9. Results of applying Quantization (Q_{50}) to the 8x8 block after DCT sample pixel values in fig. 8



(a)



(b)

Figure 10. Obtained Quadtree with zero threshold from the fig. 9 above, which is the result of DCT and Quantization (Q_{50}): (a) Quadtree regions. (b) Quadtree values

The obtained quadtree in figure 10 by applying our methodology (DCT-Quantization-Quadtree with zero threshold) is more compact than the quadtree obtained in figure 4c using only quadtree with threshold of 10. Our result contains only 13 nodes excluding parent nodes (17 including all nodes). The other examples contain 25 nodes excluding parent nodes (33 including all nodes). The overall compression using our method for this small sample is 4.92 to 1 excluding the empty 'x' nodes, and 3.76 to 1 including the 'x' nodes. This number is almost double the compression rate of the quadtree alone.

5. PERFORMANCE & ANALYSIS OF THE PROPOSED SCHEME

A. *Experimental Setup*

To verify the effectiveness of our proposed algorithm, extensive simulations are carried out for a range of standard images used for image processing and compression.

The system programming was done in C# computing language and run on Windows 10 Professional operating system. The hardware used for testing is a personal computer with Intel Core i5-4590S CPU running at 3GHz clock speed, having 8GB RAM and 500GB hard drive.

The suggested algorithm was run and compared with other algorithms that are mainly based on quadtree and JPEG. In total, the comparison was done on:

- Quadtree (Lossless, threshold zero)
- Quadtree (Lossy, with threshold 10)
- Quadtree with reduced image resolution (Lossy)
- JPEG: which consists of DCT / Quantization / Huffman / RLE. Quantization is set at Q_{50}.
- Our Algorithm: DCT / Quantization / Quadtree with both threshold zero and 10, but keeping only values for threshold zero. Quantization is set at Q_{50}.

B. *Selected Images*

Five standard images were selected to compare the different compression methods. To simplify the work, all images were taken with 8 bits shades of grey, so each pixel in the bitmap format took one byte. All pictures were 256 pixel length by 256 pixels wide. The total picture size was 64KB (kilo bytes) in one dimension, and the total bitmap size was 192KB because the bitmap gives 3 bytes for each pixel, 1 per color (red, green, blue), even if the picture is grey. The pictures are: Lena (Fig. 11a), Barbara (Fig. 11b), 21 shades of Gray (Fig. 11c), House (Fig. 11d), and Moon (Fig. 11e).

Lena image became a standard for image processing testing because of "its detail, flat regions, shading, and texture" according to an editor-in-chief of IEEE Transactions on Image Processing.

Barbara image is also being used in many image processing tests since its release in 1993 by Allen Gersho.

The shades of gray are a way to show the robustness of compression in presence of very few changes in the picture. The shades of gray, house, and moon pictures are all provided in free image repository for image processing.

C. *Performed Tests*

1) *Quadtree (Lossless, threshold zero)*

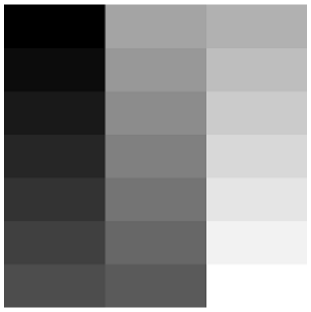
In this test, a normal quadtree is built based on the original bitmap. The threshold was set to zero, which means that any grouped nodes need to have exactly the same value. Therefore, this is a lossless compression. All images details are restored after decompression with the same accuracy.

2) *Quadtree (Lossy, with threshold 10)*

In this test, a quadtree is built based on the original bitmap. The threshold was set to 10, which means that any grouped nodes can have a maximum deviation of 10 with respect to the average value. Therefore, this is a lossy compression. A part of the image details is lost due to approximation.

(a) Lena

(b) Barbara


(c) Grey 21

(d) House

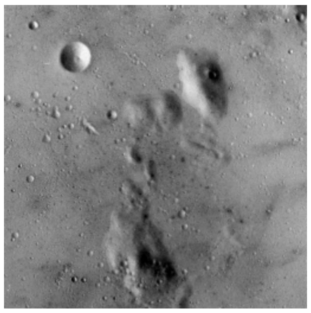

(e) Moon

Figure 11. Selected Images for compression comparison.

3) Quadtree with reduced image resolution (Lossy)

In this test, a quadtree is built based on a modified bitmap of the original image, whereby all pixel values are rounded down to the nearest 10. For example, the value 253 is saved as 250, and the value 27 is saved as 20. Therefore this is a lossy compression, with the loss happening in the rounding of color values.

4) JPEG

In this test, a JPEG image is built based on the original bitmap. As explained in the previous sections, JPEG compression consists of DCT, Quantization, Huffman, and RLE. It is a lossy compression method with the loss happening in the quantization part of the compression.

5) Our Algorithm: DCT / Quantization / Quadtree

In this algorithm, DCT is applied first on the original bitmap. Then quantization of 50 is applied on DCT. Next a quadtree is built based on the results. A threshold of zero is used first. Another threshold of 10 is also tested. Although the threshold of 10 gives a better compression, the image loses most of its quality because it was already inducing loss, so it is dropped, and only threshold zero was used in final testing. Therefore, this is a lossy compression, with the loss happening in the quantization part of the compression similar to JPEG.

D. Results of the Performed Tests

All the five compression algorithms were applied on all the five selected standard images. We considered every node in algorithms involving quadtrees as one byte value, even header nodes containing no data. So, assuming all images having the same size of 65536 bytes, a full quadtree with no compression will consume 87381 bytes. This number is received if we start by 1 header node, then keep multiplying by 4 and adding to the sum until we reach 65536 nodes. The calculation is shown next:

$$1 + 4 + 16 + 64 + 256 + 1024 + 4096 + 16384 + 65536$$
$$= 87381 nodes$$

The test results are as follows:

1) Lena Image

In the Lena image, for the quadtree with threshold zero, the resulting file was bigger than the original. This was due to not having many grouped groups of 4 nodes, and the additional header nodes added a considerable overhead. For the quadtree with threshold 10, a better grouping was performed, and the resulting tree had 31357 nodes with the file reduced to 48% of its original size. However, if the image resolution was reduced by a round of 10, then quadtree applied with zero threshold, the file was not compressed and kept its 100% size. The JPEG algorithm achieved the best compression with the file reduced to just 23% of its original size. Finally, our algorithm achieved a file reduction to 56% of its size.

2) Barbara Image

The Barbara image is considerably more complicated than all other images. For the quadtree with threshold zero, the resulting file was bigger than the original. This was due to the complicated image and not having many grouped groups of 4 nodes, and also the parent header nodes as additional overhead. For the quadtree with threshold 10, the resulting tree had 51713 nodes with the file reduced to 79% of its original size. But if the image resolution was reduced by a round of 10, then quadtree applied with zero threshold, the file became of bigger size than the original. The JPEG algorithm achieved the best compression with the file reduced to just 28% of its original size. Finally, our algorithm achieved a file reduction to 68% of its size with 44761 nodes.

3) *21 Shades of Grey Image*

The 21 shades of grey image is the simplest and most uniform picture in the selected set. Its simplicity made it possible for the quadtree with threshold zero to work efficiently, with a resulting file's size 14% of the original. The was improved with threshold 10 to become just 3% of the original size. Even for the image resolution reduction by a round of 10, then quadtree applied with zero threshold, the file was compressed to 11% of its size. The JPEG algorithm performed well, but not the best, with reduction to 6% of its original size. Finally, our algorithm achieved a file reduction to 34% of its size.

4) *House Image*

In the house image, the quadtree with threshold zero performed badly with a resulting file bigger than the original, for the same reasons mentioned before in Lena and Barbara images. For the quadtree with threshold 10, a better grouping was performed, with the file reduced to 54% of its original size. For the third method of the image resolution reduced by a round of 10, then quadtree applied with zero threshold, the file was slightly compressed to 92% of its size. The JPEG algorithm achieved the best compression with the file reduced to just 20% of its original size. Finally, our algorithm achieved a file reduction to 44% of its size with 28705 nodes.

5) *Moon Image*

The last test was done on an image of the moon. The quadtree with threshold zero performed badly with a resulting file bigger than the original as in all other non-uniform images. For the quadtree with threshold 10, the file was reduced to 52% of its original size due to a better grouping. When the image resolution reduced by a round of 10, then quadtree applied with zero threshold, the file failed to be compressed and became larger in size. The JPEG algorithm achieved the best compression with the file reduced to 19% of its original size. Finally, our algorithm achieved a file reduction of 40% of its original size.

The results are shown and compared in the next two tables, the first with the size in bytes, and the other with compression percentage (new size divided by original size):

The tables I and II show that JPEG is clearly by far the most compact method in most of the scenarios, except in 21 shades of grey image. As for the others using quadtree, our method seems to be exceeding all the others in at least 3 out of 5 scenarios.

As for the images accuracy, the best is the quadtree with threshold zero because it is lossless, followed by JPEG and our algorithm with equal accuracy, then by the quadtree with threshold 10 and the reduced image with quadtree. To be able to check the accuracy, we need to calculate the peak signal-to-noise ratio (PSNR) of the image. PSNR is used along with the mean-square error (MSE) to compare image compression quality. The formula for PSNR is:

TABLE I. Comparison: Original and After Compression File Size in Bytes for the Tested Compression Techniques

Method	Picture Name				
	Lena	**Barbara**	**Grey**	**House**	**Moon**
Original Bitmap Size (3dimensions)	192KB	192KB	192KB	192KB	192KB
Original Bitmap Size (1 dimension)	65536	65536	65536	65536	65536
Quadtree (threshold zero)	86713	87193	9445	74505	87337
Quadtree (threshold 10)	31357	51713	1901	35293	34173
Reduced image with quadtree	65465	75557	3189	60513	78821
JPEG	15073	18350	4129	13369	12452
DCT, quantization, quadtree (Th=0)	36557	44761	22281	28705	26453

TABLE II. Comparison: Original and After Compression File % of Original File for the Tested Compression Techniques

Method	Picture Name				
	Lena	**Barbara**	**Grey**	**House**	**Moon**
Original Bitmap Size (1 dimension)	100%	100%	100%	100%	100%
Quadtree (threshold zero)	132%	133%	14%	114%	133%
Quadtree (threshold 10)	48%	79%	3%	54%	52%
Reduced image with quadtree	100%	115%	11%	92%	120%
JPEG	23%	28%	6%	20%	19%
DCT, quantization, quadtree (Th=0)	56%	68%	34%	44%	40%

$$PSNR = 20.log_{10}(MAX_I)–10.log_{10}(MSE) \tag{6}$$

Where MAXI is the maximum possible pixel value of the image, or 255 in our 8 bits gray examples. MSE is the sum of square of pixel value difference of corresponding original and compressed image (after decompression), divided by the number of pixels.

MSE is zero for lossless images because the two images are identical, and thus the difference between any corresponding pixels is zero. This gives infinite PSNR according to equation (6). As for the other lossy techniques, the values are calculated and shown in Table III. Our proposed algorithm has an identical PSNR as JPEG method because the loss is only in the quantization part. Furthermore, PSNR in all test images was superior for our proposed model than classical Quadtree with a threshold of 10, or reduced image with quadtree. Although the Quadtree with threshold zero has a much better PSNR (infinite) compared to others, but the tradeoff is that the size is larger than the original bitmap size in all different scenarios but one.

E. *Practical Implications*

Image compression techniques have several practical implications. The main application is storing and sharing images through computer communications: social ap-

TABLE III. Comparison: After Compression File PSNR in Decibels for the Tested Compression Techniques

Method	Picture Name				
	Lena	**Barbara**	**Grey**	**House**	**Moon**
Quadtree (threshold zero)	∞	∞	∞	∞	∞
Quadtree (threshold 10)	35.1	34.5	38.2	36.5	34.4
Reduced image with quadtree	35.1	34.5	38.2	36.5	34.4
JPEG	47	41.2	59	53.4	40.8
DCT, quantization, quadtree (Th=0)	47	41.2	59	53.4	40.8

plications, email or websites. Other applications include broadcast television, tele conferencing, medical images, and satellite images. Our proposed model is to be mostly used in computer communications, due to its static image properties. It is recommended to be used for photographs taken directly from cameras or indirectly after conversion from other image types.

6. CONCLUSION

In this paper, an algorithm based on DCT/Quantization and Quadtrees was suggested for image compression. This algorithm was tested and compared with other techniques mostly using quadtrees: quadtree with zero threshold, quadtree with threshold 10, reduced image resolution with quadtree. It was also compared with JPEG algorithm which is based on DCT. The simulation shows that although JPEG is superior to all the others in the comparison, our algorithm based on DCT with quadtree performed well with respect to the all other applied methods based on quadtrees. Our current study was only focused on merging DCT/Quantization and Quadtrees for image compression. Other methods can be merged and tested in the future for better compression efficiency.

References

[1] N. John, A. Viswanath, V. Sowmya, and K. Soman, "Analysis of various color space models on effective single image super resolution," in *Intelligent Systems Technologies and Applications: Volume 1*. Springer, 2016, pp. 529–540.

[2] A. S. D. George, "Types of bitmaps - windows forms .net framework — microsoft learn." 2023, accessed on June 2, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/advanced/types-of-bitmaps?view=netframeworkdesktop-4.8

[3] M. Sharma *et al.*, "Compression using huffman coding," *IJCSNS International Journal of Computer Science and Network Security*, vol. 10, no. 5, pp. 133–141, 2010.

[4] C. Prudvi, D. Muchahary, and A. Raghuvanshi, "Analysis of image compression techniques for iot applications," in *2022 2nd International Conference on Intelligent Technologies (CONIT)*. IEEE, 2022, pp. 1–5.

[5] N. Karthikeyan, N. S. Kumar, and S. Mugunthan, "Comparative study of lossy and lossless image compression techniques," *International Journal of Engineering & Technology*, vol. 3, no. 34, pp. 950–953, 2018.

[6] A. Raid, W. Khedr, M. A. El-Dosuky, and W. Ahmed, "Jpeg image compression using discrete cosine transform-a survey," *arXiv preprint arXiv:1405.6147*, 2014.

[7] K. Cabeen and P. Gent Ridcully, "Image compression and the discrete cosine transform,," University Lecture - Math45, 1992.

[8] H. Samet, "The quadtree and related hierarchical data structures," *ACM Computing Surveys (CSUR)*, vol. 16, no. 2, pp. 187–260, 1984.

[9] T. Markas and J. Reif, "Quad tree structures for image compression applications," *Information Processing & Management*, vol. 28, no. 6, pp. 707–721, 1992.

[10] M. A. M. Y. Alsayyh, D. Mohamad, T. Saba, A. Rehman, and J. S. Al-Ghamdi, "A novel fused image compression technique using dft, dwt, and dct." *J. Inf. Hiding Multim. Signal Process.*, vol. 8, no. 2, pp. 261–271, 2017.

[11] Y. Iqbal and O.-J. Kwon, "Improved jpeg coding by filtering 8× 8 dct blocks," *Journal of Imaging*, vol. 7, no. 7, p. 117, 2021.

[12] L. Alam, P. K. Dhar, M. Hasan, M. G. S. Bhuyan, and G. M. Daiyan, "An improved jpeg image compression algorithm by modifying luminance quantization table," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 1, p. 200, 2017.

[13] A. L. Alshami and M. Otair, "Enhancing quality of lossy compressed images using minimum decreasing technique," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 3, 2018.

[14] F. Ebrahimi, M. Chamik, and S. Winkler, "Jpeg vs. jpeg 2000: an objective comparison of image encoding quality," in *Applications of Digital Image Processing XXVII*, vol. 5558. SPIE, 2004, pp. 300–308.

[15] F. Jiang, W. Tao, S. Liu, J. Ren, X. Guo, and D. Zhao, "An end-to-end compression framework based on convolutional neural networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 10, pp. 3007–3018, 2017.

[16] N. Johnston, D. Vincent, D. Minnen, M. Covell, S. Singh, T. Chinen, S. J. Hwang, J. Shor, and G. Toderici, "Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4385–4393.

[17] K. Dharavath and S. Bhukya, "A novel approach for improving image compression ratio," in *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*. IEEE, 2022, pp. 1–4.

[18] R. Ranjan and P. Kumar, "An improved image compression algorithm using 2d dwt and pca with canonical huffman encoding," *Entropy*, vol. 25, no. 10, p. 1382, 2023.

[19] R. Fischer, P. Dittmann, C. Schröder, and G. Zachmann, "Improved lossless depth image compression," 2020.

[20] X. Liu, P. An, Y. Chen, and X. Huang, "An improved lossless image compression algorithm based on huffman coding," *Multimedia Tools and Applications*, vol. 81, no. 4, pp. 4781–4795, 2022.

[21] F. Keissarian, "A new quadtree-based image compression technique using pattern matching algorithm," in *International Conference on Computational & Experimental Engineering and Sciences (ICCES)*, vol. 12, no. 4, 2009, pp. 137–143.

[22] ——, “A new quad-tree segmented image compression scheme using histogram analysis and pattern matching,” in *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, vol. 5. IEEE, 2010, pp. 694–698.

[23] Y. Chen and X. Luo, “Type-2 fuzzy logic based dct for intelligent image compression,” in *2014 IEEE International Conference on Computer and Information Technology*. IEEE, 2014, pp. 908–912.

[24] S. Nosratian, M. Moradkhani, and M. Tavakoli, “Hybrid data compression using fuzzy logic and huffman coding in secure iot,” *Iranian Journal of Fuzzy Systems*, vol. 18, no. 1, pp. 101–116, 2021.

[25] B. SrinivasLN, “Digital image compression using dct algorithm: An improvement,” 2019.

[26] P. Upadhyay and A. Singh, “Image compression using discrete cosine transform and discrete wavelet transform,” 2019.

[27] S. J. Rani, G. Glorindal, and I. A. Herman, “Medical image compression using dct with entropy encoding and huffman on mri brain images,” *Asian Journal of Applied Science and Technology (AJAST)*, vol. 6, no. 2, pp. 16–25, 2022.

[28] D. Ahmad, L. RITM, U. Pradesh, H. U. Rahman, M. S. S. Diwedi, and L. RITM, “Smart fuzzy scheme based image compression using quadtree approach,” 2021.

[29] S. Janaa and S. Mandal, “Dwt based image compression using quadtree decomposition and huffman encoding,” *Contemporary Issues in Computing, Topics in Intelligent Computing and Industry Design (ICID)*, vol. 2, no. 1, pp. 144–147, 2020.

[30] C.-Y. Pang, R.-G. Zhou, B.-Q. Hu, W. Hu, and A. El-Rafei, “Signal and image compression using quantum discrete cosine transform,” *Information Sciences*, vol. 473, pp. 121–141, 2019.

[31] P. T. Chiou, Y. Sun, and G. Young, “A complexity analysis of the jpeg image compression algorithm,” in *2017 9th Computer Science and Electronic Engineering (CEEC)*. IEEE, 2017, pp. 65–70.

[32] A. Prithul, “Implementing a quadtree,” 2021, accessed on June 23, 2024. [Online]. Available: https://medium.com/@aprithul/implementing-a-quadtree-baac7acf3ad0#:~:text=Creating%20the%20quadtree%20involves%20adding,N)%20for%20creating%20the%20quadtree