

Hybrid Representative Coding Scheme for AMBTC Compressed Images

Hua Wu¹, Chia-Chen Lin^{2*}, Chin-Chen Chang³, Xu Wang⁴

¹ Beijing Information Science and Technology University, China

² Department of Computer Science and Information Engineering,
National of Chin-Yi University of Technology, Taiwan

³ Department of Information Engineering and Computer Science, Feng Chia University, Taiwan

⁴ School of Information Science and Engineering, University of Jinan, China
sunshinesmilewhh@126.com, ally.cclin@ncut.edu.tw, alan3c@gmail.com, ise_wangx@ujn.edu.cn

Abstract

Among the representative image compression techniques, absolute moment block truncation coding (AMBTC) offers acceptable image quality for the reconstructed image with a relatively low computational cost. Nevertheless, this approach has its constraints. In instances where an image contains numerous objects, the quality of reconstructed images or compression performance may be suffered. In order to solve this problem, we applied the idea of Huffman coding and proposed an adaptive image compression method with hybrid representative coding based on absolute moment block truncation coding (HRC-AMBTC). To achieve our object, the image blocks in an image are classified into three categories: flat, smooth, and complex. To maintain the image quality of the restored image while providing good compression performance, our designed representative coding is applied to the smooth block. Moreover, complex blocks are encoded by three quantized values, and their bitmaps are derived from our proposed difference clustering. In addition, to further improve HRC-AMBTC, we compress codes of the flat block, which bit per pixel is lower than our original HRC-AMBTC. Finally, the experiments confirm the effectiveness and reliability of the proposed compression methods after comparing them with other variants of the AMBTC method.

Keywords: Image compression, AMBTC, Huffman code, Hybrid representative coding, Difference-based clustering

1 Introduction

As the Internet and digital technologies advance, the abundance of images has surged dramatically. Image compression emerges as a crucial technique in information transmission, particularly in scenarios where bandwidth is constrained. It proves invaluable for prioritizing the semantics of images and facilitating swifter transmission. In general, compression techniques are roughly classified into lossless compression [1-3] and lossy compression [4].

In lossless compression algorithms, the original image is recovered from the compressed image without losing any quality within the image; therefore, it is applied in fields requiring high fidelity, such as medical images and remote sensing images. Familiar lossless compression algorithms include run-length coding [5-6], Huffman coding [7], learned compression methods [8-9], etc. In run length coding, the data is recorded as a single value and the number of the same consecutive values. It works well for data with long runs of identical symbols, such as binary diagrams. However, it is hard to compress most natural images, because of the many objects in natural images, and the run length coding-based algorithm will need to apply more codes to record discontinuous pixels. In this case, some compression methods pay attention to the occurrence frequency of a symbol in the data and utilize a shorter codeword to represent symbols that occur frequently, for example, Huffman coding. However, it is heavy work to generate statistics for each symbol in a massive dataset and to allocate distinguishable codes. For example, it takes huge computing resources to find the Huffman code of each pixel value of a set of images.

In situations where bandwidth is limited, receivers often prioritize the content of images over intricate details. Hence, lossy compression methods are more apt for real-time image compression applications or scenarios where only the context demands attention. These methods achieve higher compression rates by emphasizing the overall content rather than the finer texture details within an image. At present, popular lossy compression algorithms include vector quantization (VQ) [10-12], block truncation coding (BTC) [13-15], compressed sensing (CS) [16], etc. The VQ image compression is a block-based method. For a given image set, each image is divided into several non-overlapping blocks, and then a set of representative image blocks, also called codewords, are derived from these blocks to build a codebook by the c-means clustering approach. Once the codebook is trained, all blocks in an image can be replaced by the indices with the most similar codewords. Finally, the generated indices are treated as the compression results and transmitted along with the trained codebook to the receiver. Different from VQ, BTC was proposed by Delp and Mitchell in 1979 [13] and avoids the

*Corresponding Author: Chia-Chen Lin; Email: ally.cclin@ncut.edu.tw

iterative process of obtaining representative image blocks. One of its improved algorithms is absolute mean block truncation coding (AMBTC), which was proposed by Lema and Mitchell [16]. AMBTC prompts the compression performance by comparing with conventional BTC.

To enhance the compression performance of the conventional AMBTC method, several approaches, including bitmap omission [17], block classification [18–19], and adjustment of quantizers [20], have been explored to lower the bitrate while maintaining acceptable image quality of the reconstructed images. For example, in 2003, Hu [17] observed that when the difference between two quantizers is smaller than a predefined threshold, the bitmap will not affect the image quality of the reconstructed image. Therefore, Hu omitted the bitmap when a block is categorized as a flat one, and used the block's mean value to reconstruct the flat block. Chen et al. [19] applied quadtree partitioning and designed a variable-rate AMBTC compression method for color images. The basic idea of their idea [19] is to partition the image into blocks with various sizes according to the blocks' complexities. In 2015, Mathews and Nair [21] considered human visual characteristics and then designed an adaptive AMBTC method based on edge quantization. Their method categorized image blocks into edge and non-edge blocks and then calculated quantizers based on the edge information. Since they consider detailed textures, such as edge information, experimental results confirm their method provides better image quality than other AMBTC variants. In 2018, Hong [20] optimized the quantizers to reduce the impact of bitmap alteration during data embedding. In 2021, Chen et al. [22] proposed an AMBTC-based image compression scheme by using a ternary representation technique, which achieves high image quality because of the complex blocks described by ternary representation. In their method, the k-means clustering method is introduced to assist in encoding smooth blocks and complex blocks, respectively, and spends most of the time on finding suitable clustering centers.

To improve the image quality of reconstructed images post-decompression while maintaining a similar compression performance as Chen et al. [22], this paper introduces an adaptive image compression method featuring hybrid representative coding based on absolute moment block truncation coding (HRC-AMBTC). The aim is to preserve finer texture details within images. Recognizing that texture complexity varies across different areas, a universal approach cannot simultaneously optimize both image quality and compression performance. In our proposed scheme, image blocks are classified into three categories: flat, smooth, and complex. Smooth blocks undergo representative coding based on Huffman coding. Furthermore, complex blocks are encoded using three quantized values, with their bitmaps derived from the proposed difference clustering method. Additionally, an enhanced version of HRC-AMBTC is devised to further compress flat blocks, achieving a lower bit per pixel compared to our original HRC-AMBTC proposal.

The major contribution of our work can be summarized as follows.

(1) Classify an image into three block categories, i.e., flat block, smooth block, and complex block, based on the block's feature.

(2) A mean-based encoding method is applied to compress a flat block only with 9-bit code which is less than the 32 bits offered by the conventional AMBTC.

(3) Drawing inspiration from Huffman coding, the bitmaps of AMBTC compressed smooth blocks are collected in advance. Subsequently, a set of eight distinctive patterns can be derived. Ultimately, a Huffman coding table with a size of 35 bits is employed to encode the bitmaps of AMBTC compressed smooth blocks.

(4) A difference-based clustering method is proposed in this paper to generate three quantized values and its image quality of the reconstructed complex blocks can remain. Different from a conventional clustering algorithm, the corresponding bitmap for the complex block so that such as k-mean, the seeds of clustering in our difference-based clustering are determined by ordinal pixel differences instead of random selection.

The rest of this paper is organized as follows. Section 2 introduces related works, i.e., Huffman coding, AMBTC, and Chen et al.'s AMBTC-based compression [22], as basic knowledge. Section 3 introduces the proposed image compression method. Section 4 presents the experimental results. Finally, the conclusion is given in Section 5.

2 Related Works

In this paper, our goal is to preserve the benefits of AMBTC while improving upon the compression performance provided by Chen et al.'s scheme [22], all without compromising the image quality of the reconstructed images. To achieve our objectives, we designed representative coding based on the idea of Huffman coding. Then, we designed a novel AMBTC-based compression scheme by combining our proposed representative coding and a difference-based clustering method. To make our paper self-contained, we introduce Huffman coding and AMBTC in Subsections 2.1 and 2.2, respectively. As Chen et al.'s method [22] will be a primary comparison with our proposed scheme, we introduce Chen et al.'s scheme in Subsection 2.3.

2.1 Huffman Coding

Huffman coding [23] is a variable-length coding method that considers the frequency of symbols in data. It allocates shorter codes for the more frequently appearing symbols and more codes for fewer appearing symbols. The basic process of Huffman coding is as follows: 1) it generates statistics on the number of different symbols in an image and arranges the symbols in descending order by their probabilities. 2) Next, a Huffman tree is built by making these probabilities of symbols as the leaves of a binary tree. 3) Subsequently, it takes two symbols as nodes with the two lowest probabilities to generate a new node with a probability, that equals the sum of the

two lowest. Then the new probabilities are rearranged in descending order. 4) This last step is repeated until all the symbols appear in the Huffman tree. 5) Then the left and right branches of the Huffman tree are assigned 0 and 1, respectively. Finally, each symbol is obtained by its code by connecting a 0-1 sequence on the path from the root node to its related node.

2.2 The AMBTC Method

With the AMBTC method, each non-overlapping block of an image block is represented by two relative quantized values and a bitmap [20]. The original image I with a size of $W \times H$ is divided into M non-overlapping blocks of size $w \times h$, where w is the width and h is the height of a block, and the general size for a block is set as 4×4 .

$$M = \left(\frac{W}{w}\right) \times \left(\frac{H}{h}\right). \quad (1)$$

The mean μ_k of k th block $B_k = \{p_{k1}, p_{k2}, \dots, p_{km}\}$ is calculated by

$$\mu_k = \frac{1}{m} \sum_{i=1}^m p_{ki}, \quad (2)$$

where p_{ki} ($i=1,2,\dots,m$) are pixels in the k th block B_k , and $m = w \times h$. Subsequently, the pixels of B_k are divided into two sets. The Set B_k^H includes h_k elements whose pixel values are higher than or equal to μ_k , while the Set B_k^L includes h_l elements whose pixel values are lower than μ_k . Both two sets in a block are calculated by their means, also called two quantizers, as follows.

$$\mu_k^H = \frac{1}{h_k} \sum_{(p_{ki} \in B_k^H \cap p_{ki} \geq \mu_k)} p_{ki}, \quad (3)$$

$$\mu_k^L = \frac{1}{h_k} \sum_{(p_{ki} \in B_k^L \cap p_{ki} < \mu_k)} p_{ki}, \quad (4)$$

$$B_{k_map} = \begin{cases} 1, & \text{if } p_{ki} \in B_k^H, \\ 0, & \text{if } p_{ki} \in B_k^L, \end{cases} \quad (5)$$

where B_{k_map} represents the bitmap for the k th block.

Similarly, the other blocks of an image I are encoded in the same way. Finally, the image I is transformed to the compression codes C_I' which is represented as $C_I' = \{(\mu_k^H, \mu_k^L, B_{k_map})_K\}$, ($k=1,2,\dots,m$; $K=1,2,\dots,M$), where m is the number of pixels in a block and M is the number of non-overlapping image blocks in an image.

After getting the AMBTC-based compression codes, receivers can reconstruct pixels for each block with the same size as the encoding block, by the following formula.

$$\begin{cases} p'_{ki} = \mu_k^H, & \text{if } B_{k_map} = 1, \\ p'_{ki} = \mu_k^L, & \text{if } B_{k_map} = 0, \end{cases} \quad (6)$$

where p'_{ki} represents the i th pixel in the k th reconstructed image block. Experimental results confirmed that a reconstructed image can be derived and the average image quality will be larger than 30 dB.

2.3 Review of Chen et al.'s Scheme

In 2021, Chen et al. [22] proposed an improved AMBTC-based hybrid encoding scheme for compressed images according to the complexity of each divided non-overlapping block. There are three different kinds of blocks: flat, smooth, and complex. The flat blocks are encoded utilizing two quantized values obtained by the conventional AMBTC method. The decoding method for the flat blocks is very simple, all pixels are replaced with the recorded mean values. The clustering algorithm is applied to the smooth and complex blocks, respectively. The encoding methods for the smooth and the complex are introduced as follows.

In the encoding method for the smooth blocks, all the original AMBTC-based bitmaps of smooth blocks are clustered into k ($k=128,256,512$ or others) groups by the conventional clustering method, called the k -means clustering algorithm. In this method, k representative bitmaps are generated and then an index with the smallest Euclidean Distance to the current smooth block's bitmap is chosen to serve its final bitmap. Finally, two quantizers and indices of the selected bitmap derived from the k representative bitmaps serve as the compression code for the smooth block. In Chen et al.'s scheme, k has been tested as 128, 256, and 512. Experimental results confirm that their method successfully increases the PSNR of the decompressed image. However, the k -cluster centers and their cluster indices are also needed to send to the receivers, which helps in the decoding operation of the smooth block. The relationships of original bitmaps, final representative bitmaps obtained by the clustering algorithm, and its indices are described as follows.

$$a_s^* = \arg \min_a \left(\sum_{j=0}^{n \times n - 1} (B_{sj} - C_{aj})^2 \right)^{\frac{1}{2}}, \quad (7)$$

where a_s^* is the index of codeword having the nearest distance to B_{sj} . B_{sj} and C_{aj} represent the j th element of bitmap B_s and codebook C_a as final representative bitmaps, respectively.

However, the encoding method for complex blocks is different from that of smooth ones. This is because Chen et al.'s scheme collects all smooth blocks' bitmaps first, and then k representative bitmaps are concluded by using k -means. By contrast, only 16 pixels within a complex block are considered to divide pixels into three groups and then a trinary bitmap containing three groups is derived. In a word, a complex block is encoded with three quantized values acquired by the k -mean clustering algorithm, in

which k is only set as 3. In addition, a ternary digit ranging from 0 to 2 denoted in the complex blocks' bitmaps indicates the positions of three quantized values.

Owing to the different clustering objects of smooth and complex blocks, the decoding procedures designed for these two kinds of blocks are also different. For the smooth block, the clustering centers are obtained by the corresponding indices recorded in the compressed code. For the complex block, after the approximate bitmap is obtained, the received two quantizers are used to fill in according to the indications of this approximate bitmap. As for the complex block, the decoding procedure is quite straightforward, and the receiver only needs to use the trinary bitmap and three quantizers which are extracted from the compression code to build up the reconstructed block.

Compared with Subsection 2.2, Chen et al.'s method [22] improves the compression performance by considering the complexity of non-overlapping blocks, and such an arrangement makes the textures of complex blocks clearer than those offered by the conventional AMBTC [20] and successfully avoids serious image distortion. In detail, Chen et al.'s method [22] uses one quantized value without a bitmap to encode the flat block and an index selected from 128 representative bitmaps deriving from k -means clustering to encode the smooth block and finally uses a trinary bitmap along with three quantizers to encode the complex block. In [22], their experimental results show the conventional AMBTC uses fixed representation without distinguishing blocks according to their complexity. This makes the compression performance derived from 8 representative test images the same as 2 bpp, but the

PSNR is quite different because of the different image textures. Conversely, the average image quality offered by Chen et al.'s scheme is around 34.73 dB with two thresholds 4 and 16, which is better compared with that of AMBTC. Nevertheless, Chen et al.'s computational cost is relatively higher compared to conventional AMBTC. This is attributed to their encoding methods, which involve the application of two clustering techniques for smooth and complex blocks. These methods require several iterative inferences to obtain either bitmaps or quantized values.

While Chen et al.'s scheme enhances the compression performance of conventional AMBTC, we have noted that the k -means clustering approach with limited iterations is susceptible to the effects of initialization, resulting in instability in the image compression outcome. Therefore, we aim to maintain a similar approach to Chen et al.'s scheme but enhance the encoding methods for smooth and complex blocks without relying on the k -means clustering algorithm. This adjustment aims to either reduce computational costs or stabilize compression performance.

3 Proposed HRC-AMBTC Method

To achieve our objectives of maintaining stable compression performance and offering the same image quality of the reconstructed image as AMBTC, a hybrid representative coding-based AMBTC (HRC-AMBTC) is designed. The flowchart of our proposed HRC-AMBTC is depicted in Figure 1, and the detailed description of our proposed HRC-AMBTC is described in the following subsection.

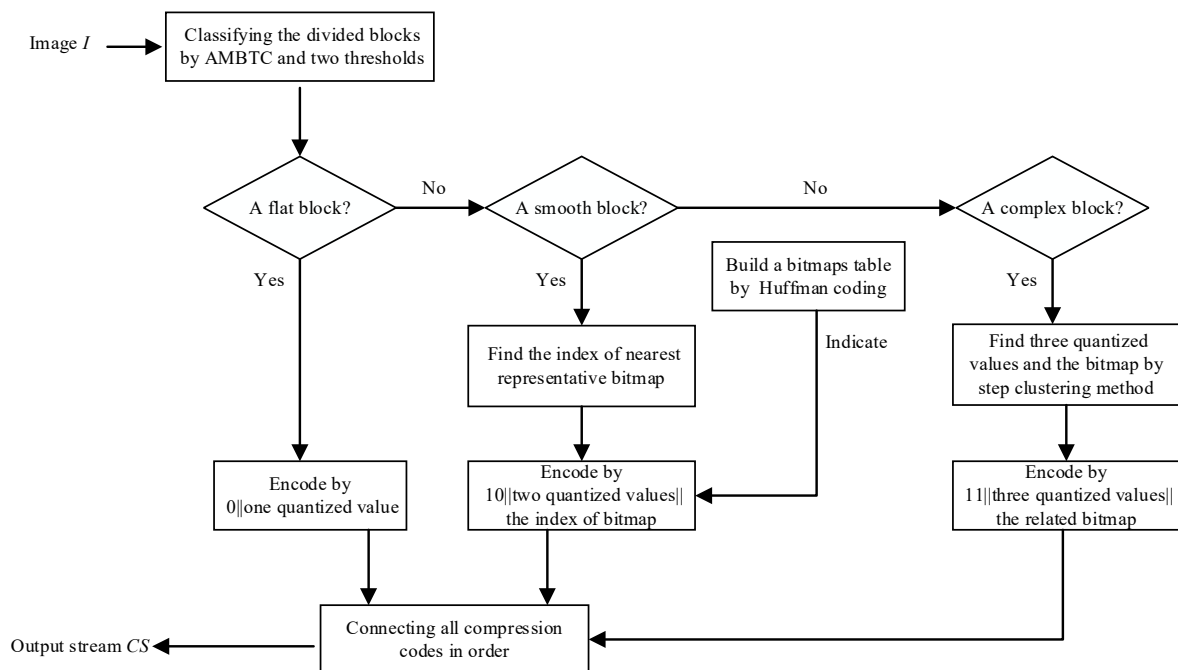


Figure 1. HRC-AMBTC flowchart

For image I , after performing AMBTC encoding, a set of compression codes are generated. Let $C_k = (\mu_k^H, \mu_k^L, B_{k_map})$ be the AMBTC compression codes of the image block B_k , where μ_k^H and μ_k^L are two quantized values, and $\mu_k^H \geq \mu_k^L$. B_{k_map} is the bitmap of block B_k . Two thresholds t_0 and t_1 are set to classify the image blocks into three block types before encoding: flat, smooth, and complex blocks. Then, the difference D_k between two quantized values, where $D_k = \mu_k^H - \mu_k^L$, is computed and compared with the above two thresholds. If $D_k \leq t_0$, image block B_k belongs to the flat block, in which all pixels are reconstructed by the block's mean without using a bitmap. If $t_0 < D_k < t_1$, image block B_k is categorized as a smooth block. The

smooth block is encoded by two quantized values because its texture is more complex than a flat block. Considering that the majority of image blocks in an original image are categorized as smooth blocks, we utilize a bitmap code derived from the Huffman coding table to substitute the bitmap. This substitution aims to further reduce the size of the bitmap associated with the smooth block. A detailed description of the encoding method for bitmaps, which belong to the smooth blocks is described in Subsection 3.1. If $D_k \geq t_1$, image block B_k is determined as a complex block, whose encoding method is similar to the conventional AMBTC, but three quantized values and the corresponding bitmap are used. The algorithm for our proposed HRC-AMBTC is described as follows:

Algorithm 1. HRC-AMBTC encoding an image

Input: an original image

Output: compression code TL

1. Divide an original image into n_t 4×4 image blocks.
 2. Read one image block and classify it into three block groups: flat, smooth and complex blocks, according to two thresholds t_0 and t_1 .
 3. If the current block is a flat block, it is encoded as L_k , including 1-bit flag 0_2 and one binary quantized value. The detail is demonstrated in Subsection 3.1.
 4. If the current block is a smooth block, it is encoded as its compression code L_k is marked as 10_2 . The codes for a 4×4 image block is obtained by connecting four parts, as $10_2 \parallel (\mu_k^L)_2 \parallel (D_k)_2 \parallel (M_k)_2$, where μ_k^L is 8 bits low quantizer of AMBTC, D_k is 4 bits of the difference between the original low quantizer and the original high quantizer, and M_k is the encoding result according to the encoding bits defined in the Huffman coding table. The detail is demonstrated in Subsection 3.2.
 5. If the current block is a complex block, its compression code L_K is marked as 11_2 , and the compression code includes three quantized values and a ternary bitmap, which is generated by our proposed difference-based clustering algorithm. The detail is shown in Subsection 3.3.
 6. If the current block is not the last block in the image, read the next block and perform the encoding operation by repeating Steps 3-5 in order.
 7. Obtain the final compression code TL by concatenating all blocks' compression codes $L_r = L_1 \parallel L_2 \parallel \dots \parallel L_k$ ($k=1,2,\dots, n_t$) and the Huffman table code L_H .
-

Algorithm 2. HRC-AMBTC reconstructing an image with the compression codes

Input: The stream of image compression code TL

Output: A reconstructed image

1. Obtain the auxiliary information. The first 36 bits of L_H recode Huffman coding rules, and the remainder codes L_r is used to build the reconstructed image.
 2. Build the Huffman coding table. Each list represents the compressed bitmap of a smooth block.
 3. Reconstruct the image by filling the block with the same size as the divided block in the compression process.
 4. If the first bit of code L_r is 0_2 , the current block is determined as a flat block. It is reconstructed according to the decoding operations described in Subsection 3.1.
 5. If the two bits of code L_r is 10_2 , the current block is determined as a smooth block. It is reconstructed according to the decoding operations described in Subsection 3.2.2.
 6. If the two bits of code L_r is 11_2 , the block is determined as a complex block. It is reconstructed according to the decoding operations described in Subsection 3.3.
 7. Reconstruct the image blocks B' one by one, though repeating Steps 4-6.
 8. Obtain a reconstructed image I' , when all the rebuilt blocks B' are located in order as the path in the compression process.
-

In our proposed HRC-AMBTC scheme, an original image I is divided into several non-overlapping blocks. Then, the blocks are compressed one by one via the compression algorithm described in Algorithm 1. Finally, all the compression codes of the blocks and the auxiliary information are generated, such as the representative code table. The reconstructing scheme is also called the decompressing scheme. It is simpler than the compression, and the decompression algorithm is described in Algorithm 2. The decompression algorithm processes the compression codes instructed by block-type flags embedded in the codes. To demonstrate the compression and decompression processes of different kinds of blocks, this paper provides four subsections to individually introduce the processes for flat, smooth and complex blocks respectively, in Subsections 3.1, 3.2, 3.3. Subsection 3.4 introduces an improved HRC-AMBTC named HRC-AMBTCv2.

3.1 Encoding and Decoding Process for the Flat Block

A flat block mainly includes low-frequency information. The differences among pixels within the flat block have a faint influence on semantic expression.

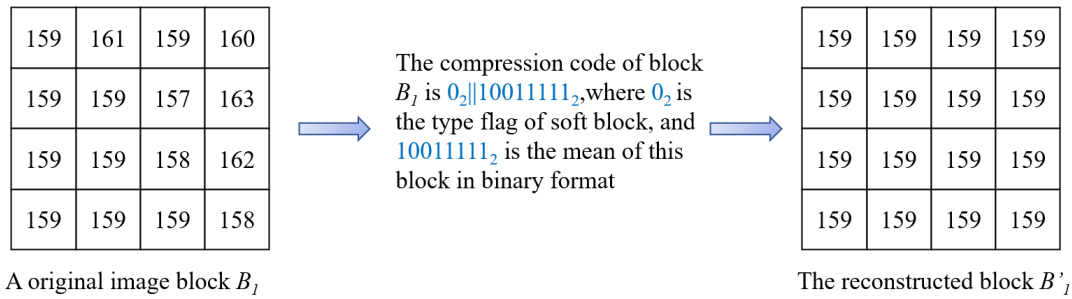


Figure 2. The compression and decompression processes for a flat block

The flat block is encoded with 9-bit compression code with our proposed HRC-AMBTC which is much smaller than the 32 bits obtained by conventional AMBTC. In addition, the block mean is directly used in both compression and decompression processes and avoids the deviation from the average of μ_k^H and μ_k^L . Therefore, the computation complexity of our proposed HRC-AMBTC is relatively lower than that of conventional AMBTC for a flat block.

3.2 Encoding and Decoding Process of Smooth Block

When $t_0 < D_k < t_1$, the current image block is determined as a smooth block. Typically, within a smooth block, the differences between neighboring pixels are greater compared to those in a flat block. Consequently, two quantized values are employed to depict pixels within a smooth block, and a bitmap containing '0' or '1' is essential to document the corresponding positions of two quantizers. To address this, we have introduced a novel approach in our HRC-AMBTC scheme, defining a 2×2 black-white block shown in Table 1 as the fundamental unit. This allows us to denote a representative bitmap for a smooth

block using fewer bits while maintaining acceptable image quality during smooth block reconstruction.

To achieve our objective, a 4×4 smooth image block's bitmap can be decomposed and represented by four 2×2 basic units. In summary, there are eight distinct patterns of 2×2 basic units, as illustrated in Table 1.

Table 1 reveals that the eight patterns of 2×2 basic units can be organized into four distinct groups. Within each group, we define one flag to indicate two types: single units and continuous units. The continuous type means the following unit is the same as the previous unit. The main color flag serves to differentiate between subgroups within a given group. For example, consider 'Group 3,' the main color flag '1' signifies that both pixels at the top of the 2×2 basic unit are black, and conversely for the other color configuration. Regarding the two types, 'Single unit' with its indicator '10' indicates that the pixels within the current 2×2 basic unit are uniformly black or white. Conversely, 'Continuous units' with its indicator '11' signifies the presence of two adjoining 2×2 basic units belonging to 'Group 3.' Through the amalgamation of these three types of indicators - main color flag, group, and types - we can

$$\mu_k^f = \text{mean}(B_{k_all}), \quad (8)$$

where k is the order number of the block and B_{k_all} represents all pixel values in block B_k .

During the decoding phase, the first bit 0_2 instructs that the current block is a flat type, while the following 8-bit are selected from the compression code and then converted into a decimal number. Finally, the compressed block is reconstructed by filling the decimal value in the 4×4 block. Figure 2 shows the encoding and decoding procedures for a flat block.

efficiently minimize the size of the bitmap associated with a smooth block.









To give a better explanation for our general encoding policy for a smooth block bitmap, an example of encoding and decoding a smooth block is shown in Figure 3. In the encoding process for a given smooth block and the type flag for smooth block is denoted as 10_2 , the smooth block is first divided into four basic units. Next, these four basic units are arranged in order by top-left top-right bottom-left bottom-right. Then, these four units are encoded into two parts according to the encoding policy listed in Table 1, which are the ‘main color flag’ part as 1011_2 and the encoding code of four basic units as $011_2||11_2$. Among the four basic units shown in Figure 3, the first two bits indicate the main color flags for the first two basic units and are denoted as 10_2 because one of them has the top black pattern and the other has the top white pattern. Then, follow the compression code as 011_2 because the two neighboring units belong to the same group ‘Group 0,’ while the last two basic units belong to ‘Group 3’ denoted as 11_2 and then with compression code as 11_2 because the two neighboring units also belong to the same group. Finally, the compression result of this sample smooth block is obtained as $10\ 1011\ 01111_2$ by concatenating the smooth block-type flag 10_2 , four main color flags 1011_2 , and the encoding code of four basic units 01111_2 . In addition, the decoding process of a smooth block’s code is the reverse process of the encoding program.

Building on the example demonstrated in Figure 3, it becomes clear that the general encoding rules, devised

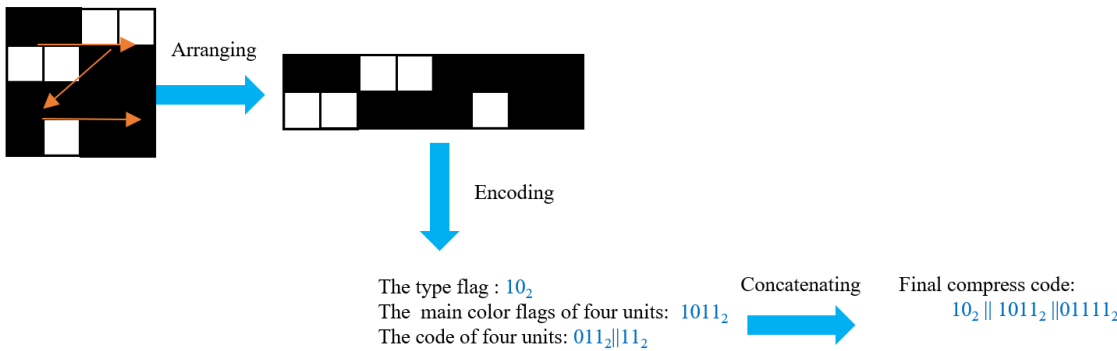
for the 8 patterns within a 2×2 basic unit, effectively reduce the size of the original bitmap for a 4×4 block. To further streamline the compression code’s size for smooth blocks, we integrate the Huffman coding technique, which combines the eight patterns illustrated in Table 1, as previously introduced. To apply Huffman coding, the first step involves determining the number of blocks associated with each pattern. The pattern with the highest frequency is assigned the shortest bit representation, as demonstrated in Table 2 using the ‘Lena’ test image as a case study. Ultimately, a Huffman coding table tailored to a specific image can be generated. Taking Table 2 as an example, within the ‘Lena’ test image, which comprises 21,324 smooth blocks consisting of 2×2 units, there are eight patterns along with their corresponding coding bits. It is worth noting that the pattern belonging to Group 3 ‘Continuous units’, for instance, is assigned a coding bit of ‘1’ with just 1 bit of representation. This is primarily because Group 3 ‘Continuous units’ encompass 9,294 blocks, making it the most prevalent pattern in comparison to the others.

Following the example illustrated in Figure 3(a), the ultimate compression code, utilizing the Huffman coding table provided in Table 2, can be further reduced in size, as evident in Figure 4. A comparison between Figure 4 and Figure 3(a) reveals that the compression code generated with the Huffman coding table is smaller than the one encoded by using the general encoding rules outlined in Table 1.

Table 1. General encoding rules for a 2×2 basic unit

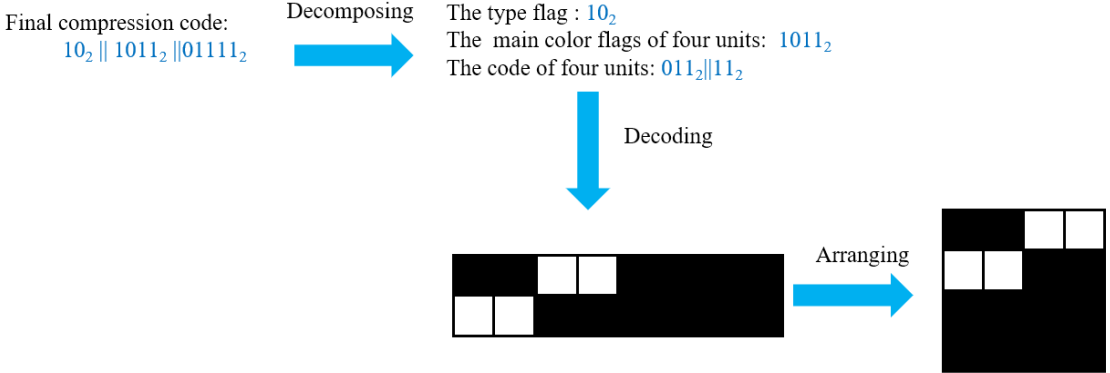
The 2×2 basic unit	Main color flag	Descriptions	Group	Types	
				Single unit	Continuous units
	1	Top black blocks	0	010_2	011_2
	0	Top white blocks			
	1	Left black blocks	1	110_2	111_2
	0	Left white blocks			
	1	Left diagonal black blocks	2	1000_2	1001_2
	0	Left diagonal white blocks			
	1	Most pixels or whole black blocks	3	10_2	11_2
	0	Most pixels or whole white blocks			

Encoding process:



(a) Encoding procedure of a smooth block with general encoding rules shown in Table 1

Decoding process:



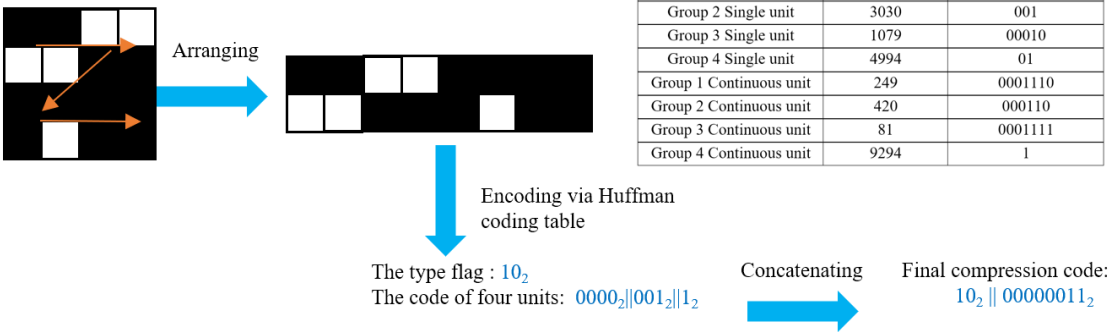
(b) Decoding procedure of a smooth block with general encoding rules shown in Table 1

Figure 3. Example of encoding and decoding smooth block with general encoding rules shown in Table 1

Table 2. Huffman coding table using the ‘Lena’ test image

Patterns of 2×2 basic units	Amounts of blocks	Encoding bits	Length of encoding bits
Group 0 Single unit	2177	0000	4 bits
Group 1 Single unit	3030	001	3 bits
Group 2 Single unit	1079	00010	5 bits
Group 3 Single unit	4994	01	2 bits
Group 0 Continuous unit	249	0001110	7 bits
Group 1 Continuous unit	420	000110	6 bits
Group 2 Continuous unit	81	0001111	7 bits
Group 3 Continuous unit	9294	1	1 bit

Encoding process:



Huffman coding table

Patterns of 2×2 basic units	Amounts of blocks	Encoding bits
Group 1 Single unit	2177	0000
Group 2 Single unit	3030	001
Group 3 Single unit	1079	00010
Group 4 Single unit	4994	01
Group 1 Continuous unit	249	0001110
Group 2 Continuous unit	420	000110
Group 3 Continuous unit	81	0001111
Group 4 Continuous unit	9294	1

Figure 4. Example of encoding a smooth block with the Huffman coding table shown in Table 2

Utilizing the encoding bits obtained from the Huffman coding table, as illustrated in Table 2, a 4×4 image smooth block can be encoded as $10_2 \parallel (\mu_k^L)_2 \parallel (D_k)_2 \parallel (M_k)_2$, where μ_k^L is 8 bits low quantizer of AMBTC, D_k is 4 bits of the difference between the original low and the high quantizers, and M_k is the encoding bits listed in Huffman coding table. The detailed description can be found in Subsection 3.2.1.

3.2.1 Encoding Process of Smooth Block

Here, we take a 4×4 image block as an example to illustrate the compression process of a smooth block as follows.

Step 1. Two quantized values and the bitmaps of the smooth blocks are calculated by conventional AMBTC as $C_k = (\mu_k^H, \mu_k^L, B_{k_map})$, where μ_k^H and μ_k^L are quantized values, and B_{k_map} is the bitmap.

Step 2. The difference D_k between μ_k^H and μ_k^L is recorded with bits. For example, if $t_1=16$, it takes 4 bits to present the difference D_k in a binary representation. Then, the lower quantized value is converted into an 8-bit binary code.

Step 3. Bitmap B_{k_map} is divided into four non-overlapping 2×2 units and then they are listed in a line via the scan order as top-left, to bottom-left, to top-right, and to bottom-right.

Step 4. Generate the Huffman coding table based on features of Bitmap B_{k_map} .

Step 5. Bitmap code M_k is derived by encoding four 2×2 units according to the Huffman coding table.

Step 6. Finally, the codes for a 4×4 image block is obtained by connecting four parts, as $10_2 \parallel (\mu_k^L)_2 \parallel (D_k)_2 \parallel (M_k)_2$, where μ_k^L is 8 bits low quantizer of AMBTC, D_k is 4 bits of the difference between the low quantizer and the high quantizer, and M_k is the encoding results according to the encoding bits defined in the Huffman coding table. The first two bits 10_2 is the type flag of a smooth block.

3.2.2 Decoding Process of Smooth Block

If a type flag of an image block begins with 10_2 , it is determined as a smooth block. This binary stream of a smooth block is used to reconstruct an image block in the following steps.

Step 1. The following 8-bit binary numbers are converted to a decimal quantized value μ_k^L . Then the 4-bit numbers after them are also converted to a decimal number as deference D_k ; therefore the other quantized value μ_k^H is obtained by $D_k + \mu_k^L$.

Step 2. Construct the bitmap B'_{k_map} according to the extracted Huffman coding table L_H and the received encoding bits.

Step 3. The 4×4 reconstructed image block is obtained by filling with μ_k^H and μ_k^L according to the bitmap B'_{k_map} .

The reconstructed bitmaps for all smooth blocks can be generated by employing the 35-bit Huffman coding table, which includes the corresponding encoding bits. However, it is important to emphasize that, as illustrated

by the eight patterns outlined in Table 2, the reconstructed bitmaps may not perfectly replicate the original ones. This discrepancy arises because patterns linked to Group 2 ‘Continuous units’ and Group 3 ‘Continuous units’ each encompass five possible variations, as indicated in Table 1. Nevertheless, our experimental results will demonstrate that our proposed scheme consistently outperforms prior methods in terms of compression performance and image quality of the reconstructed image.

3.3 Encoding and Decoding Process of Complex Block

Complex blocks typically contain a variety of pixel values, and the disparities among these values tend to be more substantial compared to the differences found in the other two block types. As a result, a complex block is characterized by three quantized values and a ternary bitmap. To obtain a representative ternary bitmap for a given complex block, we have developed a difference-based clustering method that involves five essential steps:

Difference-based clustering algorithm:

Step 1. The pixel values of one 4×4 image block B_k are listed in set $S_k = \{p_{k1}, p_{k2}, \dots, p_{kq}\}$ without repetitive numbers in ascending order, where $p_{k1} < p_{k2} < \dots < p_{kq}$.

Step 2. The differences $\{q_{k1}, q_{k2}, \dots, q_{kq-1}\}$ between two neighboring pixel values in set S_k are calculated and arranged in descending order.

Step 3. According to the biggest difference and the second biggest difference, the values of S_k are divided into three groups by cutting off the list $\{p_{k1}, p_{k2}, \dots, p_{kq}\}$ at the positions related to two differences. After calculating the means $(\mu_k^L, \mu_k^M, \mu_k^H)$ of these three groups, the members in the three different groups are replaced by their means, respectively. In addition, $\mu_k^L < \mu_k^M < \mu_k^H$. The 0, 1 and 2 used in the ternary bitmap B_{k_all} record the positions of $(\mu_k^L, \mu_k^M, \mu_k^H)$, which is used for reconstructing the image block similar to AMBTC.

Step 4. If the following differences q_{k3} are very close to q_{k2} , or the differences before q_{k3} are equal, more grouping forms of S_k are considered by cutting off the list at different positions. Then, repeat Step 3 against different grouping forms. The three final groups with the highest PSNR are found by comparing each possible reconstructed image and the original image. If the next difference q_{k4} has problems similar to q_{k3} 's, Step 4 repeats against q_{k4} .

Step 5. The image block is described as $L_k = (\mu_k^L, \mu_k^M, \mu_k^H, B_{k_all})$ through Step 1 to 4. In addition, if there are only two values of an image block, this block is compressed as a smooth block described in Section 3.2.

In our proposed HRC-AMBTC, complex blocks are compressed by variable-length coding, and their type flags are defined as 11_2 . Details of the compression process of a complex block are introduced as follows, based on the ternary code L_k .

Complex block's encoding algorithm:

Step 1. Arrange the values in ascending order without repeating numbers.

Step 2. Compute the differences of two neighbors and

rearrange them in descending order.

Step 3. Divide the sets B'_k into three parts, according to the two biggest values among these differences. Then calculate the three quantized values and the bitmap.

Step 4. Obtain the differences d_k^L and d_k^H by the neighboring quantized values in the ascending order.

Step 5. Encode three quantized values. The lower quantized value μ_k^L is converted into an 8-bit binary code. The differences d_k^L and d_k^H are calculated, where $d_k^L = \mu_k^M - \mu_k^L$ and $d_k^H = \mu_k^H - \mu_k^M$. The following several bits record both d_k^L and d_k^H in binary representation

according to the rules in Table 2. Therefore, the three quantized values $(\mu_k^L, \mu_k^M, \mu_k^H)$ are encoded as $c_k^q = (\mu_k^L)_2 \parallel (flag1)_2 \parallel (d_k^L)_2 \parallel (flag2)_2 \parallel (d_k^H)_2$, where $flag1$ and $flag2$ are length flags as shown in Table 3. As an illustration, consider a complex block with three quantized values: $(\mu_k^L, \mu_k^M, \mu_k^H) = (76, 136, 216)$. The differences between them are computed, yielding $d_k^L = 60$, and $d_k^H = 80$. Referring to Table 3, the code for these quantized values is represented as $c_k^q = 01001100_2 \parallel 0_2 \parallel 111100_2 \parallel 1_2 \parallel 1010000_2$.

Table 3. The compression rules of differences D_k^L and D_k^H

Condition I	Condition II	Length flag	Code length (bit)
$\mu_k^L \leq 128$	$d_k^L < 64$	0	6
	$d_k^L \geq 64$	1	$\lceil \log_2(256 - \mu_k^L) \rceil$
	$d_k^H < 64$	0	6
	$d_k^H \geq 64$	1	$\lceil \log_2(256 - \mu_k^M) \rceil$
$\mu_k^L > 128$	$d_k^L < 32$	0	5
	$d_k^L \geq 32$	1	$\lceil \log_2(256 - \mu_k^L) \rceil$
	$d_k^H < 32$	0	5
	$d_k^H \geq 32$	1	$\lceil \log_2(256 - \mu_k^M) \rceil$

Step 6. Encode the ternary bitmap of a complex block. Three position flags of bitmap $B_{k,all}$ are defined as 0_2 , 10_2 and 11_2 , respectively. Then, the bitmap is encoded in the stream c_k^l . It is noted that the ternary bitmap of a complex block is generated by our proposed difference-based clustering as described at the beginning of this section.

Step 7. Obtain the code of a complex block. Finally, the complex block is compressed by connecting type flag 11_2 , the code of three quantized values c_k^q and the bitmap's code c_k^l , as $11_2 \parallel c_k^q \parallel c_k^l$.

To give a better explanation for encoding complex blocks, here, we take a 4×4 complex block $B'_k = \{28, 219, 171, 167; 34, 223, 223, 222; 33, 217, 218, 218; 32, 154, 171, 185\}$ as an example to demonstrate the encoding process. There are seven steps, that combine our proposed difference-based clustering algorithm to find three quantized values and the compression process with these three found quantizers. The details are demonstrated as follows.

Step 1. Arrange the values in ascending order without repeating numbers as $S_k = \{28, 32, 33, 34, 154, 167, 171, 185, 217, 218, 219, 222, 223\}$.

Step 2. Compute the differences of two neighbors as $\{4, 1, 1, 120, 13, 4, 14, 32, 1, 1, 3, 1\}$, and rearrange the differences in the descending order as

$\{120, 32, 14, 13, 4, 4, 3, 1, 1, 1, 1, 1\}$.

Step 3. Divide the sets B'_k into three parts, according to the two biggest values as 223 and 222: $\{28, 32, 33, 34\}$, $\{154, 167, 171, 171, 185\}$, $\{217, 218, 218, 219, 222, 223, 223\}$. Then calculate the three quantized values and the bitmap, which are shown as follows: $B_{k,all} = \{0, 3, 2, 2, 0, 3, 3, 3, 0, 3, 3, 3, 0, 2, 2, 2\}$.

Step 4. Obtain the differences d_k^L and d_k^H by the neighboring quantized values, where $d_k^L = 138$ and $d_k^H = 50$.

Step 5. Encode three quantized values according to Table 3, and the code is shown as $c_k^q = 00100000_2 \parallel 1_2 \parallel 10001010_2 \parallel 0_2 \parallel 110010_2$.

Step 6. Encode the bitmap as $c_k^l = 011101001111110111110101010_2$.

Step 7. Obtain the final compression codes by connecting 11_2 , c_k^q and c_k^l as $11_2 \parallel 001000001100010100110010_2 \parallel 011101001111110111110101010_2$.

Lastly, a 4×4 complex block is condensed into a validated codeword consisting of three quantized values and one ternary bitmap. The decoding process for a complex block differs from that of a flat block and a smooth block due to the variable length of the compression code for a complex block. Consequently, the decoding procedure requires careful consideration to ensure that a sufficient number of bitmap values are decoded.

The detailed decompression procedure is introduced as follows.

Complex block's decoding algorithm:

Step 1. If a type flag begins with 11_2 , it is a complex block. The following 8-bit binary bits are converted into decimal value μ_k^L . After the differences d_k^L and d_k^H are obtained according to Table 3, the other two quantized values μ_k^M and μ_k^H are computed by $\mu_k^M = d_k^L + \mu_k^L$ and $\mu_k^H = d_k^H + \mu_k^M$.

Step 2. The reconstructed complex image block is filled in with μ_k^L , μ_k^M and μ_k^H by the bitmap's instruction.

3.4 Improved HRC-AMBTC

In our proposed HRC-AMBTC, flat blocks are encoded with a single quantized value, block mean, offering an enhanced compression capability for smooth. To further improve the compression performance, a refined approach is denoted as HRC-AMBTCv2, mainly designed for the flat blocks generated by our proposed HRC-AMBTC. Generally, HRC-AMBTCv2 remains consistent with HCR-AMBTC, except for the encoding and decoding processes of flat blocks. The novel algorithms introduced in HCR-AMBTCv2 for handling flat blocks are outlined below.

Step 1. Compute the difference between two consecutive flat blocks. This involves finding the difference between the mean of a flat block and the mean of the flat block immediately preceding it. If the first block is flat one, it is encoded directly according to Section 3.1.

Step 2. Design re-compression rules. We define four compression modes, which correspond to four different ranges, including $[-1,1]$, $[-3,3]$, $[-7,7]$, $[-15,15]$, and their range flags are denoted as 00_2 , 01_2 , 10_2 , 11_2 , respectively. For a flat block, the encoding comprises a 1-bit type flag, a 1-bit plus-minus sign, a 1-bit compression instruction, and a 1-bit, 2-bit, 3-bit, 4-bit, or 8-bit codeword used to indicate the difference between the means of the current flat block and its preceding flat block. Table 4 shows the encoding rules for flat blocks in HRC-AMBTCv2.

Step 3. Confirm the compression mode and encode the flat blocks.

According to Table 4, the coding efficiency based on each rule is calculated. The best one is chosen as the compression mode for all the flat blocks of the original image, and the range flag is recorded at the beginning of an image codeword. For example, the first two bits of an image code is 10_2 to indicate this block is flat one. The flat blocks of the original image are encoded to either a 2-bit or 8-bit code according to the range. If the mean of the current flat block is 168, and the mean of its preceding flat block is 165. Then the difference between the two of them will be 3 ($=168-165$). The decimal value 3 belongs to the range $[-3,3]$. Therefore, the following 2-bit 01_2 indicates the range is located at $[3,-3]$, and also takes a 2-bit codeword to record this difference as 11_2 . Then the final code of this flat block is encoded by $10_2||01_2||0_2||11_2$, where the first indicator denotes that the current block is flat, the second indicator denotes the difference range and the third indicator denotes that the difference is a positive value, and the four bits indicate the difference value. During decoding, the final code of the flat block can be used to restore the flat block. For example, the first 2-bit '10' indicates this is a flat block; the following 2-bit '01' indicates the range is $[3,-3]$, the following 1-bit '0' indicates only 2 bits are required to record the difference between the current flat block and its preceding flat block. Finally, the last 2-bit '11' denotes the difference value is 3. Therefore, the final compression code comprises 7 bits, represented as $10_2||01_2||0_2||11_2$. This is shorter than the 8-bit compression code generated by the encoding algorithm outlined in Section 3.1. It is noted that when the image code. If we rearrange the block flat and set block flat as '0' for the flat block, '10' for the smooth block, and '11' for the complex block. The 7-bit for flat block can be further shortened as 6-bit: $0_2||01_2||0_2||11_2$. In this case, for the continuous flat blocks, the second flat block can be presented with 6 bits instead of 8 bits compression code generated by the encoding algorithm outlined in Section 3.1. In the following discussion, the refined compression code $0_2||01_2||0_2||11_2$ will be used instead of $10_2||01_2||0_2||11_2$.

Table 4. Encoding rules of flat blocks based on the differences

The range flag	Range	Type flag	Plus/minus	Compression instruction	Code length
00	$[-1,1]$	0	0/1	0	1 bit
	$[-255,-2]U[2,255]$			1	8 bits
01	$[-3,3]$		0/1	0	2 bits
	$[-255,-4]U[4,255]$			1	8 bits
10	$[-7,7]$		0/1	0	3 bits
	$[-255,-8]U[8,255]$			1	8 bits
11	$[-15,15]$		0/1	0	4 bits
	$[-255,-16]U[16,255]$			1	8 bits

In HRC-AMBTCv2, the decoding process for flat blocks' compression codes also makes use of Table 4 and the mean of the flat block immediately preceding the current flat block. Following the example mentioned earlier, when the type flag indicates that the current block is flat, the first 1-bit is the indicator for flat block. The following 2-bit from the code of $0_2||01_2||0_2||11_2$ are interpreted as the range flag. The range flag 01_2 signifies the range $[-3,3]$. Assume, the mean value of the previous flat block is retrieved, such as 165, for instance. The subsequent '0' bit indicates that the mean of the current flat block exceeds 165 according to the definitions listed in Table 4, and the last 2-bit represents the difference. In this case, $11_2=3$, resulting in the calculation of the mean for the current flat block as 168 ($=165+3$). With HRC-AMBTCv2, the compression of flat blocks can be enhanced, albeit

at the expense of increased execution time required to determine the optimal range and compute the difference between a flat block and the preceding block.

4 Experimental Results

In this section, several experiment results of the proposed methods and comparative experiments are conducted to demonstrate the effectiveness and feasibility of the proposed methods. These experiments were done with Matlab on a laptop with a Core i7-10750H CPU and Windows 10. In addition, eight classic images and 1000 grey images with sizes of 512×512 were randomly selected from the BossBase-1.01 dataset, and are used in these experiments. The eight classic images are shown in Figure 5, including Peppers, Lena, Baboon, etc.

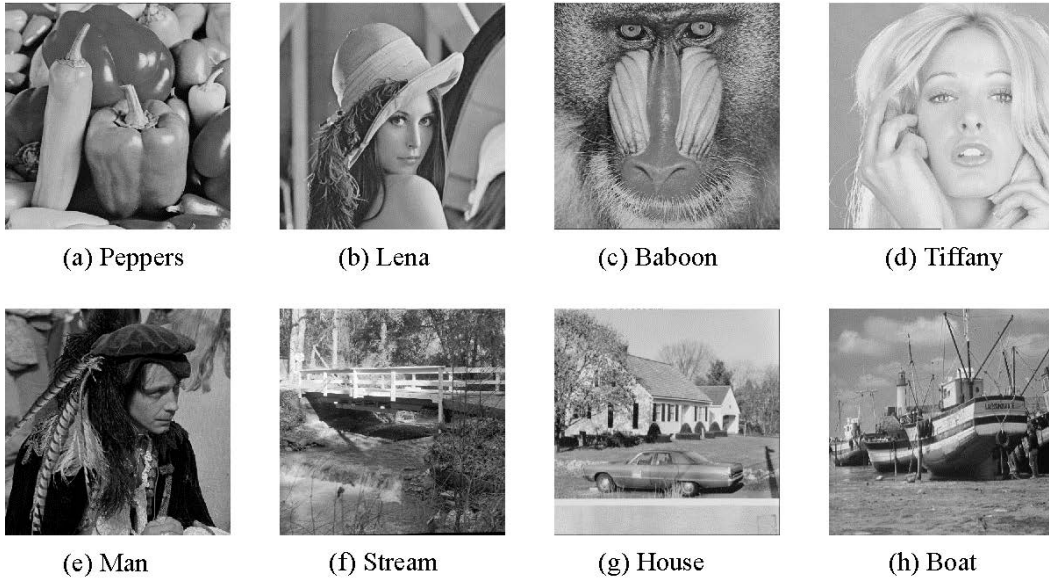


Figure 5. Eight classic images used served as the test images

In this paper, we evaluate the proposed methods by considering both bitrate defined in Eq. (9) and PSNR defined in Eq. (10) and Eq. (11). Bitrate represents the number of binary numbers required to record each pixel of an image and is named bit per pixel (bpp).

$$CE = \frac{N_c}{W \times H} (bpp), \quad (9)$$

where N_c is the bits of image compression codes.

PSNR represents the distortion between the original image and the reconstructed image defined as follows.

$$PSNR = 10 \log_{10} \left(\frac{255^2}{MSE} \right) (dB), \quad (10)$$

$$MSE = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W (p_{ij} - q_{ij})^2, \quad (11)$$

where p_{ij} and q_{ij} are the pixel values of the original image and the decompressed one, respectively; H and W are the height and the width of these images, respectively.

4.1 The Performance of the Proposed Methods

In general, the proposed methods are applicable for compressing image blocks of various sizes, including multiples of four pixels, such as 4×4 , 8×8 , 4×8 , and so on. In application, the image compression method based on 8×8 -sized blocks are similar to schemes based on 4×4 -sized blocks as mentioned in Sections 3 and 4. First, the original image is divided into many 8×8 non-overlap blocks, and each block is classified by AMBTC. Afterward, an 8×8 image block is divided into four 4×4 blocks. The 4×4 image blocks are compressed by the proposed HRC-AMBTC (or HRC-AMBTCv2). Then the codes of four blocks are connected as the code of an 8×8 -sized image block. Finally, the other 8×8 -sized blocks are encoded in the same way, and the final code of an image is acquired with several key elements. The processes

for a reconstructed image is similar to those described in Sections 3 and 4. For the sake of convenience, this paper focuses on discussions involving 4×4 -sized blocks in Section 4.

4.1.1 Coding Efficiency Comparisons

In this experiment, we compress eight test images using two methods: HRC-AMBTC and HRC-AMBTCv2. The first method utilizes the flat block compression scheme outlined in Section 3.1, while the second method focuses on compressing the differences between a flat block and its preceding flat block, as described in Section 3.4. The coding efficiency of the two proposed methods is illustrated in the table below. The block type is determined using two thresholds: $t_0=4$ and $t_1=16$.

In Table 5, the *PSNR* values for these eight test images are nearly identical when comparing the two proposed methods. However, the compression efficiency (*CE*) of HRC-AMBTCv2 is slightly lower than that of HRC-AMBTC. The reasons for these observations can be

explained as follows. Firstly, due to the utilization of the same thresholds, t_0 and t_1 , the quantities of three different types of image blocks remain unchanged. Secondly, in comparison to HRC-AMBTC, HRC-AMBTCv2 enhances the encoding of flat blocks by storing the differences between the flat blocks and their preceding counterparts. Thirdly, HRC-AMBTCv2 predominantly records these differences using variable-length coding between flat blocks and the flat blocks preceding them. When a greater number of differences between the means of flat blocks and their previous counterparts fall within the range of $[-3,3]$ than $[-7,7]$, fewer bits are required to represent these differences as shown in Table 4. Consequently, images compressed using HRC-AMBTCv2 exhibit a notable enhancement in *CE*. Nonetheless, when the majority of the mean differences exceed the range threshold, the impact on *CE* is relatively modest, as seen in the case of the test images ‘Baboon’ and ‘Stream’.

Table 5. Results of HRC-AMBTC and HRC-AMBTCv2

Methods	Items	Peppers	Lena	Baboon	Tiffany	Man	Stream	House	Boat
HRC-AMBTC	<i>PSNR</i>	35.7684	36.1412	31.1083	38.3870	34.0423	32.8848	35.1848	34.4224
	<i>CE</i>	1.7494	1.6312	2.7025	1.3579	2.1308	2.6785	1.9428	2.0687
HRC-AMBTCv2	<i>PSNR</i>	35.7666	36.1409	31.1083	38.3870	34.0375	32.8846	35.1744	34.4223
	<i>CE</i>	1.7238	1.5855	2.7019	1.2826	2.1144	2.6753	1.8793	2.0549

Table 6. Comparisons of HRC-AMBTC and HRC-AMBTCv2

Items	Peppers	Lena	Baboon	Tiffany	Man	Stream	House	Boat
Percent (%)	14.83	28.02	0.50	42.93	10.71	2.29	29.45	8.07
Change of <i>PSNR</i>	0.0018	0.0003	0	0	0.0048	0.0002	0.0104	0.0001
Change of <i>CE</i>	0.0256	0.0457	0.0006	0.0753	0.0164	0.0032	0.0635	0.0138

Table 6 shows the proportion of flat blocks in the original image, which is related to the final *PSNR* and *CE* of the two proposed methods. The different changes are calculated by $|PSNR_{HRC-AMBTCv2} - PSNR_{HRC-AMBTC}|$ and $|CE_{HRC-AMBTCv2} - CE_{HRC-AMBTC}|$. For example, the 0.0018 in the third line against ‘Peppers’ means that the *PSNR* of the reconstructed image based on HRC-AMBTC is 0.0018 more than the one based on HRC-AMBTCv2. In the last line, it is obvious that an image with a larger proportion of flat blocks has a lower *CE* value when using the HRC-AMBTCv2. For example, the proportion of flat blocks in ‘Tiffany’ is 42.93%, which is the highest value compared with the eight same-sized images. The *CE* for ‘Tiffany’ is decreased by HRC-AMBTCv2. In other words, our proposed HRC-AMBTCv2 effectively enhances compression performance when a higher proportion of flat blocks is presented in an image.

4.1.2 Performance Comparison with Various Threshold

In our proposed scheme, an image must be divided into

non-overlapping blocks in advance and then each block is determined as flat, smooth, or complex based on two pre-determined thresholds t_0 and t_1 as mentioned in Section 3. The quantity of flat blocks is related to t_0 and the quantity of complex ones is ensured by t_1 directly. In addition, the quantity of smooth blocks is associated with both t_0 and t_1 . In application, t_0 is usually no more than 4. In our scheme, a flat block is represented by the mean of its pixel values; however, the pixel values, with larger differences among them, are replaced by their mean which will reduce the *PSNR* of the block. Hence, this section mainly discusses how the parameter t_1 affects the *PSNR*s and *CE*s of compressed images. Table 7 shows the results obtained by two proposed compression methods, HRC-AMBTC and HRC-AMBTCv2, when $t_1=32$. The *PSNR*s and *CE*s against each image may be lower than the related value in Table 5. Table 8 provides the comparisons of HRC-AMBTCv2 with different t_1 . The proportion of smooth blocks increases with a larger t_1 , such as the smooth block proportion of ‘Lena’ increasing from 47.86% to 61.36%, when t_1 is

changed from 16 to 32. When t_0 is fixed, threshold t_1 is important, because it relates to the quantities of smooth and complex blocks. The compression of smooth blocks is achieved by referencing a dedicated Huffman coding table that corresponds to the shared images or by using a pre-established generic Huffman coding table and pre-shared between the sender and receiver.

Table 8 shows the different compressed results of eight

test images by HRC-AMBTCv2, with different values of threshold t_1 . Here, we can see that the larger threshold t_1 enhances the proportion of smooth blocks in an original image. When more blocks of an image are classified into smooth ones, the final image quality of the reconstructed image decreases. At the same time, the reduced quantity of complex blocks in an image adversely impacts the quality of the reconstructed image.

Table 7. Results of HRC-AMBTC and HRC-AMBTCv2 with threshold $t_1=32$

Method	Item	Peppers	Lena	Baboon	Tiffany	Man	Stream	House	Boat
HHC-AMBTC	<i>PSNR</i>	34.5390	34.6059	29.6888	36.3571	32.1540	30.3114	33.1535	32.9653
	<i>CE</i>	1.7895	1.6611	2.7201	1.3835	2.1594	2.6950	1.9600	2.1034
HHC-AMBTCv2	<i>PSNR</i>	34.5377	34.6057	29.6888	36.3235	32.1496	30.3082	33.1041	32.9652
	<i>CE</i>	1.5884	1.3931	2.3389	1.1428	1.7676	2.1081	1.5538	1.7800

Table 8. Comparisons of HRC-AMBTCv2 with different threshold t_1

t_1	Item	Peppers	Lena	Baboon	Tiffany	Man	Stream	House	Boat
16	Percent	64.12	47.86	28.03	40.91	45.79	26.31	27.58	55.57
	<i>PSNR</i>	35.7666	36.1409	31.1083	38.3870	34.0375	32.8846	35.1744	34.4223
	<i>CE</i>	1.7238	1.5855	2.7019	1.2826	2.1144	2.6753	1.8793	2.0549
	Percent	75.04	61.36	52.47	51.59	69.43	48.75	48.99	75.15
32	<i>PSNR</i>	34.5377	34.6057	29.6888	36.3235	32.1496	30.3082	33.1041	32.9652
	<i>CE</i>	1.5884	1.3931	2.3389	1.1428	1.7676	2.1081	1.5538	1.7800

This is because a complex block is characterized by three quantized values, one more than what is used for a smooth block. This allows it to describe more intricate details from the original complex blocks.

4.2 Comparisons and Analysis

In order to prove the compression capacity of our proposed methods, four image compression methods are compared in this experiment, including AMBTC [20], Chen et al.'s method [22], HRC-AMBTC and HRC-AMBTCv2. The input dataset includes 1000 grey images selected randomly from the BossBase-1.01 dataset, and the results are shown in figures and tables as follows.

PSNR is an important index to measure reconstructed image quality. In this section, 1000 images are used to compare HRC-AMBTC with AMBTC, Chen et al.'s method [22], and HRC-AMBTCv2. Figure 6 clearly illustrates that HRC-AMBTC outperforms AMBTC significantly, with over 76.2% of the images showing an improvement in PSNR of at least 3 dB in the computed results. Furthermore, HRC-AMBTC exhibits an improvement of more than 1 dB in *PSNR* for 40.6% of the images compared to Chen et al.'s method [22]. In addition, the major difference between HRC-AMBTC and HRC-AMBTCv2 is the compression algorithm for flat blocks, potentially resulting in subtle distortions during the computation of representative values. Consequently, the *PSNR* difference between HRC-AMBTC and HRC-

AMBTCv2 across 1000 images is minimal, with only a 3.9% difference exceeding 0.1 dB.

Furthermore, the *PSNR* results of 1000 images are drawn in Figure 7. The *PSNR* of HRC-AMBTC, HRC-AMBTCv2, Chen et al.'s method [22] surpass AMBTC, because these three methods classify the divided blocks of an image as flat, smooth and complex.

Furthermore, the complex blocks are encoded using ternary quantized values and a bitmap, offering a clearer description of the image texture compared to AMBTC, which employs only two quantized values. With three quantized values, the encoding represents the lowest quantized value and two differences between three ascending quantized values. HRC-AMBTC and HRC-AMBTCv2 are different from Chen et al.'s method [22] at three parts. First, HRC-AMBTC (HRC-AMBTCv2) encodes a flat block with the mean, which avoids the deviation by calculating the mean of two quantized values. Then the representative bitmaps of HRC-AMBTC (HRC-AMBTCv2) and Chen et al.'s method [22] are acquired by a different scheme. The representative bitmaps by the first scheme are distributed regularly, but the bitmaps by the second are different in each computation, due to the clustering algorithm. Finally, HRC-AMBTC (HRC-AMBTCv2) applies a step clustering scheme to encode a complex block. It overcomes the problem where unsuitable initial values negatively impact the results by the general k -mean clustering method. In addition, the *PSNR* of HRC-

AMBTC is little better than HRC-AMBTCv2 for several images, due to the representative values of flat blocks recorded in the code. In other words, the quantized value μ^f of a flat block is encoded directly by HRC-AMBTC, but HRC-AMBTCv2 records the difference of μ^f and its front block.

The image compression efficiency is measured by the bit per pixel (*bpp*). Figure 8 shows the HRC-AMBTC and HRC-AMBTCv2 take less compression code to encode an image by comparing with AMBTC and Chen et al.'s method [22], while they also have good compression quality. Through AMBTC, all image blocks of an image are encoded with the same steps, so the *bpp* is 2 without

any changes. The other three methods compressed an image by considering the texture complexity of each block of the image. Hence, the *bpp* for images compressed using these three methods may not consistently equate to 2. Figure 8 shows *bpp* of 1000 images compressed by four different methods is distributed in 4 intervals, and the threshold value is 1, 1.5 and 2. It also shows that HRC-AMBTCv2 has clear advantages about compression over by Chen et al.'s method [22]. For example, when *bpp* < 1.5, the proportion of results based on HRC-AMBTCv2, HRC-AMBTC and Chen et al.'s method [22] are 91.7%, 55.8% and 48.4%, respectively. In addition, compared with HRC-AMBTC, HRC-AMBTCv2 has one more length step to compress all flat blocks with a suitable variable length coding.

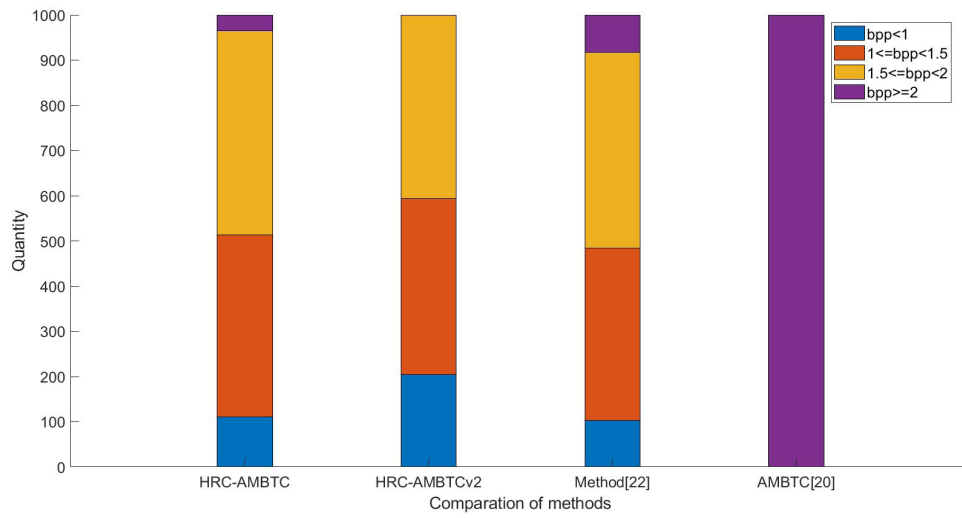


Figure 6. PSNR of 1000 images

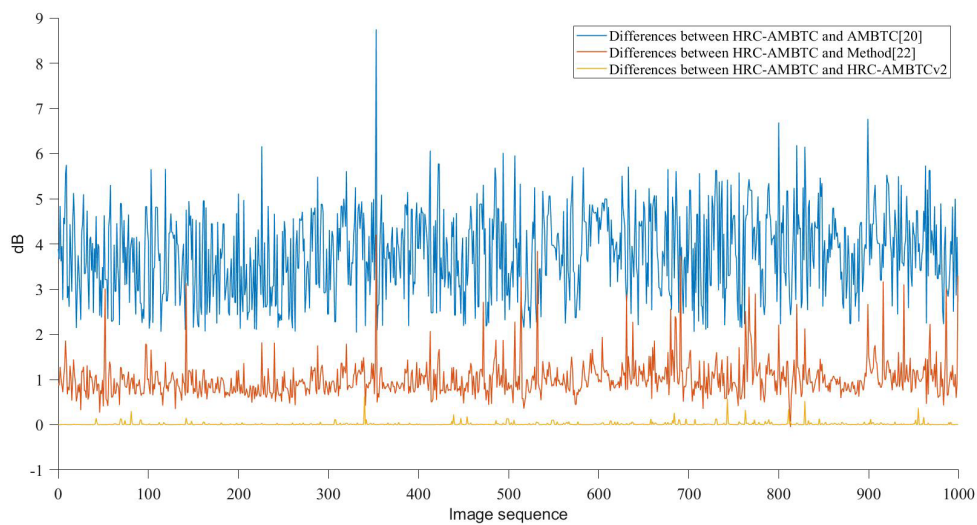


Figure 7. Differences in PSNR among 1000 images when using HRC-AMBTC as the baseline for comparison with the other three methods

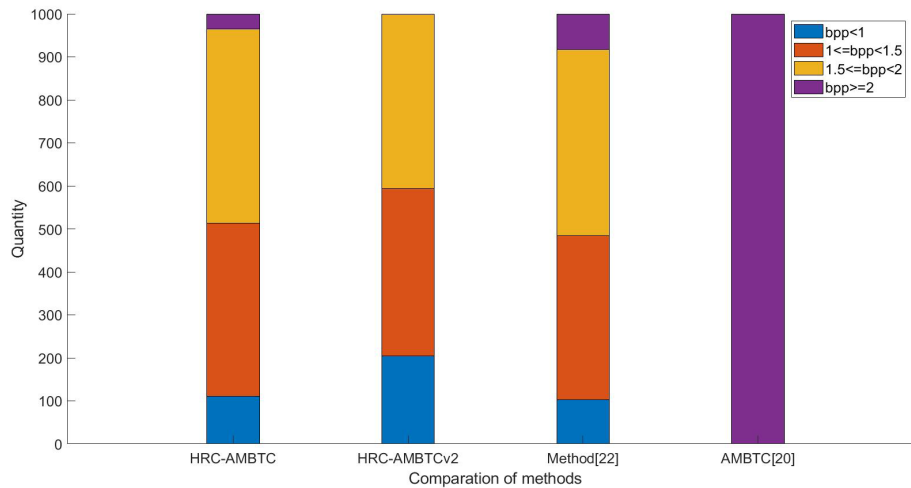


Figure 8. CEs of 1000 images with four methods

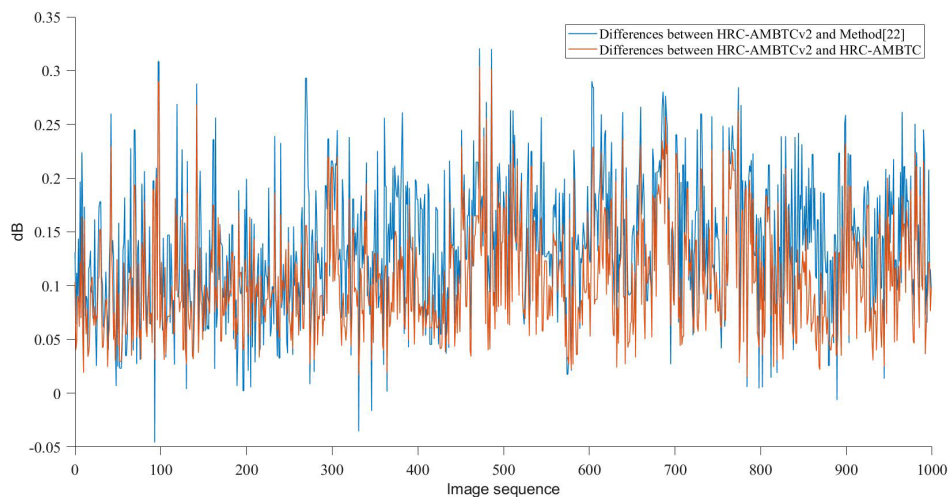


Figure 9. Differences in compression efficiency among 1000 images when using HRC-AMBTCv2 as the baseline for comparison with the other two methods

In this comparative experiment, HRC-AMBTC, HRC-AMBTCv2, and Chen et al.'s method [22] all evaluate the texture complexity of each image block using identical criteria. Figure 9 shows the differences in bpp between HRC-AMBTCv2 and Chen et al.'s method [22] with the blue line and the differences in bpp between HRC-AMBTCv2 and HRC-AMBTC with the red line. The results of bpp across 1000 images confirm that HRC-AMBTCv2 exhibits superior compression capabilities when compared to Chen et al.'s method [22]. The main reason is that the proposed HRC-AMBTC (HRC-AMBTCv2) only uses 35-bit code to build a Huffman coding table, instead of sending the whole code index table from a sender to a receiver. At times, HRC-AMBTCv2 achieves significantly higher compression efficiency than HRC-AMBTC, primarily because of its encoding algorithm's effectiveness when applied to a substantial number of flat blocks within an image.

5 Conclusions

This paper introduces an adaptive image compression method founded on AMBTC. To enhance the handling of image complexity, non-overlapping blocks were categorized into three types, each employing a distinct encoding strategy:

(1) In our HRC-AMBTC, a flat block is represented by its mean value, and the compression code is the binary form of the mean value. To enhance the compression performance, in our HRC-AMBTCv2, all mean values of flat blocks are collected and further encoded according to the mean value of their preceding flat block. Experimental results confirm that our proposed HRC-AMBTCv2 outperforms HRC-AMBTC, AMBTC, and Chen et al.'s [22], particularly when the image contains a substantial number of flat blocks.

(2) In our HRC-AMBTC and HRC-AMBTCv2, a smooth block is characterized by two quantized values. To further improve the compression performance, in our methods, the compression code of smooth block includes the original low quantizer, the difference between the original low quantizer and high quantizer, and a bitmap code derived by the Huffman coding table.

(3) In both our HRC-AMBTC and HRC-AMBTCv2, a complex block is defined by three 2 values along with a relative bitmap. These three representative values are determined by using the proposed difference-based clustering, which relies on variations in image values. Additionally, the compression code includes the first representative value and two differences to improve compression efficiency.

In our future research, we will explore the possibility of increasing the unit size used to represent each pattern from 2×2 to 4×4 or even larger. This investigation aims to assess potential impacts and identify combinations that can enhance compression performance while preserving the image quality provided by both methods presented in this paper. Inspired by previous research in image encryption [23] and ownership protection [24], our next step involves not only enhancing image quality and compression performance but also investigating the feasibility of retaining critical image information or embedding ownership details during image compression.

Acknowledgements

Hua Wu was in charge of the conceptualization, data curation, software, visualization, and writing – the original draft for this work. Finally, she also wrote the draft version of this manuscript. Chia-Chen Lin was in charge of the investigation, methodology validation, writing – review & editing, revision and funding acquisition for this work. Chin-Chen Chang was in charge of the investigation, project administration and supervision of this work. Xu Wang was in charge of the partial programming, and revision of this work. The authors received MSOT 111-2410-H-167 -005-MY2.

References

- [1] D. Kaur, K. Kaur, Huffman based LZW lossless image compression using retinex algorithm, *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 2, No. 8, pp. 3145–3151, August, 2013.
- [2] Q. Zhong, Z. Chen, X. Zhang, G. Hu, Feature-based object location of IC pins by using fast run length encoding BLOB analysis, *IEEE Transactions on Components, Packaging and Manufacturing Technology*, Vol. 4, No. 11, pp. 1887–1898, November, 2014.
<https://doi.org/10.1109/TCPMT.2014.2350015>
- [3] P. G. Howard, J. S. Vitter, Parallel lossless image compression using Huffman and arithmetic coding, *Information Processing Letters*, Vol. 59, No. 2, pp. 65–73, July, 1996.
[https://doi.org/10.1016/0020-0190\(96\)00090-7](https://doi.org/10.1016/0020-0190(96)00090-7)
- [4] J. Lin, A new perspective on improving the lossless compression efficiency for initially acquired images, *IEEE Access*, Vol. 7, pp. 144895–144906, September, 2019.
<https://doi.org/10.1109/ACCESS.2019.2944658>
- [5] E. Aldemir, G. Tohumoglu, A. Selver, Binary medical image compression using the volumetric run-length approach, *The Imaging Science Journal*, Vol. 67, No. 3, pp. 123–135, 2019.
<https://doi.org/10.1080/13682199.2019.1565695>
- [6] M. L. Rhodes, J. F. Quinn, J. Silvester, Locally optimal Run-length compression applied to CT images, *IEEE Transactions on Medical Imaging*, Vol. 4, No. 2, pp. 84–90, June, 1985.
<https://doi.org/10.1109/TMI.1985.4307701>
- [7] M. Otair, L. Abualigah, M. K. Qawaqzeh, Improved near-lossless technique using the Huffman coding for enhancing the quality of image compression, *Multimedia Tools and Applications*, Vol. 81, No. 20, pp. 28509–28529, August, 2022.
<https://doi.org/10.1007/s11042-022-12846-8>
- [8] Z. Guo, Z. Zhang, R. Feng, Z. Chen, Causal contextual prediction for learned image compression, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 32, No. 4, pp. 2329–2341, April, 2022.
<https://doi.org/10.1109/TCSVT.2021.3089491>
- [9] Z. Cheng, H. Sun, M. Takeuchi, J. Katto, Learned image compression with discretized Gaussian mixture likelihoods and attention modules, *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Washington State, WA, USA, pp. 7936–7945, 2020.
<https://doi.org/10.1109/CVPR42600.2020.00796>
- [10] R. Gray, Vector quantization, *IEEE ASSP Magazine*, Vol. 1, No. 2, pp. 4–29, April, 1984.
<https://doi.org/10.1109/MASSP.1984.1162229>
- [11] T. Kim, Side match and overlap match vector quantizers for images, *IEEE Transactions on Image Processing*, Vol. 1, No. 2, pp. 170–185, April, 1992.
<https://doi.org/10.1109/83.136594>
- [12] H. Kasban, S. Hashima, Adaptive radiographic image compression technique using hierarchical vector quantization and Huffman encoding, *Journal of Ambient Intelligence and Humanized Computing*, Vol. 10, No. 7, pp. 2855–2867, July, 2019.
<https://doi.org/10.1007/s12652-018-1016-8>
- [13] E. Delp, O. Mitchell, Image compression using block truncation coding, *IEEE Transactions on Communications*, Vol. COM-27, No. 9, pp. 1335–1342, September, 1979.
<https://doi.org/10.1109/TCOM.1979.1094560>
- [14] S. A. Mohamed, M. M. Fahmy, Image compression using VQ-BTC, *IEEE Transactions on Communications*, Vol. 43, No. 7, pp. 2177–2182, July, 1995.
<https://doi.org/10.1109/26.392959>
- [15] X.-L. Liu, C.-C. Lin, K. Muhammad, F. Al-Turjman, S.-M. Yuan, Joint data hiding and compression scheme based on modified BTC and image inpainting, *IEEE Access*, Vol. 7, pp. 116027–116037, August, 2019.
<https://doi.org/10.1109/ACCESS.2019.2935907>
- [16] M. Lema, O. Mitchell, Absolute moment block truncation coding and its application to color images, *IEEE Transactions on Communications*, Vol. 32, No. 10, pp. 1148–1157, October, 1984.
<https://doi.org/10.1109/TCOM.1984.1095973>
- [17] Y.-C. Hu, Low-complexity and low-bit-rate image compression scheme based on absolute moment block

truncation coding, *Optical Engineering*, Vol. 42, No. 7, pp. 1964–1975, July, 2003.

<https://doi.org/10.1117/1.1576776>

- [18] Z. Xiang, Y.-C. Hu, H. Yao, C. Qin, Adaptive and dynamic multi-grouping scheme for absolute moment block truncation coding, *Multimedia Tools and Applications*, Vol. 78, No. 7, pp. 7895–7909, April, 2019.
<https://doi.org/10.1007/s11042-018-6030-5>
- [19] W.-L. Chen, Y.-C. Hu, K.-Y. Liu, C.-C. Lo, C.-H. Wen, Variable-rate quadtree-segmented block truncation coding for color image compression, *International Journal of Signal Processing, Image Processing and Pattern Recognition*, Vol. 7, No. 1, pp. 65–76, February, 2014.
<http://dx.doi.org/10.14257/ijsp.2014.7.1.07>
- [20] W. Hong, Efficient data hiding based on block truncation coding using pixel pair matching technique, *Symmetry*, Vol. 10, No. 2, Article No. 36, February, 2018.
<https://doi.org/10.3390/sym10020036>
- [21] J. Mathews, M. S. Nair, Adaptive block truncation coding technique using edge-based quantization approach, *Computers and Electrical Engineering*, Vol. 43, pp. 169–179, April, 2015.
<https://doi.org/10.1016/j.compeleceng.2015.01.001>
- [22] T.-S. Chen, J. Wu, K.-S. Chen, J. Yuan, W. Hong, Hybrid encoding scheme for AMBTC compressed images using ternary representation technique, *Applied Sciences*, Vol. 11, No. 2, Article No. 619, 2021.
<https://doi.org/10.3390/app11020619>
- [23] X. Li, B. Zhang, K. Wang, Z. Li, A multi-image encryption-then-compression scheme based on parallel compressed sensing, *Optik - International Journal for Light and Electron Optics*, Vol. 290, Article No. 171304, October, 2023.
<https://doi.org/10.1016/j.ijleo.2023.171304>
- [24] C.-C. Lin, P.-Y. Wang, Y.-H. Lin, H.-C. Huang, M. Saberikamposhti, Visible watermark removal with deep learning technology, *Proc. International Symposium on Computer, Consumer and Control (IS3C)*, Taichung City, Taiwan, 2023, pp. 186–189.
<https://doi.org/10.1109/IS3C57901.2023.00057>

Biographies



Hua Wu received a Ph.D. degree from University of Chinese Academy of Sciences in 2016. Currently, she is a lecturer in Beijing Information Science & Technology University. Her research interests include image processing, pattern recognition, information security, and artificial intelligence.

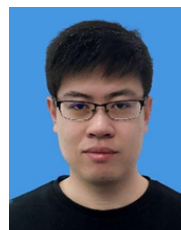


Chia-Chen Lin received the M.S. degree and the Ph.D degree in information management from Chiao Tung University, Hsinchu, Taiwan, in 1994 and 1998, respectively. She is currently a Professor in the Department of Computer and Information Management, Providence University, Sha-Lu, Taiwan.

Her research interests include image and signal processing, image data hiding.



Chin-Chen Chang received the Ph.D degree in computer engineering from National Chiao Tung University, Hsinchu, in 1982. From July 1998 to June 2000, he was Director of the Advisory Office, Ministry of Education, R.O.C. From 2002 to 2005, he was a Chair Professor at National Chung Cheng University. From February 2005, he has been a Chair Professor at Feng Chia University. In addition, he was severer as a consultant to several research institutes and government departments. His current research interests include database design, computer cryptography, image compression, and data structures.



Xu Wang received his M.E. degree from Central China Normal University in 2018 and received his Ph.D. degree from Feng Chia University in 2022. He is currently a Lecturer at the School of Information Science and Engineering, University of Jinan, Jinan, China. His research interests include image processing, multimedia security, information hiding, reversible data hiding, and deep learning.