future internet

MDPI

Article

An Innovative Recompression Scheme for VQ Index Tables

Yijie Lin [1], Jui-Chuan Liu [1], Ching-Chun Chang [2] and Chin-Chen Chang [1,*]

[1] Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan; p1263670@o365.fcu.edu.tw (Y.L.); p1200318@o365.fcu.edu.tw (J.-C.L.)
[2] Information and Communication Security Research Center, Feng Chia University, Taichung 40724, Taiwan; ccc@fcu.edu.tw
* Correspondence: ccc@o365.fcu.edu.tw

Abstract: As we move into the digital era, the pace of technological advancement is accelerating rapidly. Network traffic often becomes congested during the transmission of large data volumes. To mitigate this, data compression plays a crucial role in minimizing transmitted data. Vector quantization (VQ) stands out as a potent compression technique where each image block is encoded independently as an index linked to a codebook, effectively reducing the bit rate. In this paper, we introduce a novel scheme for recompressing VQ indices, enabling lossless restoration of the original indices during decoding without compromising visual quality. Our method not only considers pixel correlations within each image block but also leverages correlations between neighboring blocks, further optimizing the bit rate. The experimental results demonstrated the superior performance of our approach over existing methods.

Keywords: compression; vector quantization; search-order coding; side match; codebook; codeword; state codebook

Citation: Lin, Y.; Liu, J.-C.; Chang, C.-C.; Chang, C.-C. An Innovative Recompression Scheme for VQ Index Tables. *Future Internet* **2024**, *16*, 297. https://doi.org/10.3390/fi16080297

Academic Editor: Gianluigi Ferrari

Received: 10 July 2024
Revised: 13 August 2024
Accepted: 16 August 2024
Published: 19 August 2024

1. Introduction

With the rapid development of Internet technology in recent years, users can exchange almost any format of information. Images, belonging to the most popular communication media, have been widely used in various aspects. As hardware devices have advanced, image quality has improved significantly, leading to larger file sizes that require ample storage space and longer transmission times. To address these challenges, various compression techniques [1–4] have been proposed. With the rise of artificial intelligence, new technologies [5–8] based on learning or neural networks have also emerged. Among these, VQ-VAE-2 [5] is a classic neural network model that enhances generation by learning discrete latent representations and is improved through a hierarchical approach to capture more complex features. Compression technology has practical applicability in numerous scenarios, such as image library management, medical imaging, satellite imagery, and surveillance systems. These applications involve large volumes of images that require compression, and our recompression method can further reduce file sizes, thereby decreasing storage and communication costs.

Image compression techniques are generally divided into two categories: lossless image compression techniques and lossy image compression techniques. The difference between them lies in whether the original image can be fully restored. Although lossy image compression technology cannot completely restore the original image, the resulting image typically has only a slight distortion, which generally does not affect the visual understanding of the image. Vector quantization (VQ) [9,10] is an effective method for lossy compression of digital images for transmission and storage. The main advantage of VQ compression is its ability to achieve high compression ratios at low bit rates. In the VQ system, the LBG algorithm [11] is commonly used to divide the image pixels into fixed-size blocks for training the codebook. Compression is then achieved by converting the raw data into a sequence of codeword indices by matching each block with the closest codeword in the codebook.

VQ compression schemes can be divided into memory VQ and memoryless VQ. In memory VQ, image blocks are coded independently, utilizing the correlation of neighboring blocks. As a result, finite-state VQ (FSVQ) [12] was proposed, employing the statistical correlation between neighboring blocks to reduce the bit rate. On the other hand, side match VQ (SMVQ) [13] utilizes the correlation of neighboring block edges to build a state codebook, with the aim of reducing the bit rate. However, this approach may introduce additional coding distortion, potentially leading to poor image quality. In recent years, several algorithms [14–18] were proposed to enhance the performance of SMVQ. Furthermore, there are other relevant techniques, namely predictive VQ (PVQ) [19,20] and address VQ (AVQ) [21]. Generally speaking, most memory VQ schemes are more complex and require higher computational costs compared to memoryless VQ ones.

Differing from memory VQ, memoryless VQ primarily exploits the correlation between adjacent pixels within a block while disregarding the correlation between neighboring blocks. Several effective schemes were proposed in the past, including the predictive mean search (PMS) algorithm [22], index-compressed VQ (ICVQ) [23], search-order coding (SOC) [24], and the index searching algorithm with index associated list (ISA-IAL) [25]. In particular, the SOC algorithm focuses on searching for indices of the same value from neighboring blocks and utilizing a search-order code to replace the original VQ index. This approach enables effective compression by identifying a significant number of indices with the same value. Building upon this concept, Chang et al. [26] proposed a state codebook scheme in 2009. They sorted the VQ code table using the principal component analysis (PCA) [27] algorithm and enhanced the similarity search with the SOC algorithm, facilitating the detection of highly similar indices. The method introduced by Chang et al. [26] set a milestone and achieved excellent results with the SOC algorithm. However, our new scheme has significantly improved the compression performance.

Based on the SOC algorithm, this paper further employs the side match algorithm to recompress the VQ index and achieve lossless restoration of the original VQ index. In comparison to the scheme of Chang et al. [26], our method does not rely on the PCA algorithm and takes into account the similarity between neighboring blocks and the similarity of pixels within each block. Therefore, our contribution is a significant improvement on the compression rate of the VQ index table.

The structure of this paper is as follows: Section II discusses the related work, including VQ, SOC, SMVQ, and Chang's state codebook scheme. In Section III, we present our compression scheme. Section IV presents the experimental results. Finally, in Section V, we conclude this paper.

2. Related Work

There are a few commonly used methods related to our proposed scheme. Examples of these methods are provided in the following subsections to illustrate their general concepts.

2.1. Vector Quantization Algorithm

VQ is a common and popular technique used in image compression. VQ was first introduced in 1984 [9]. There were many studies and much research conducted using VQ afterwards. The proposed scheme uses VQ as its fundamental compression method. VQ compression involves a few basic steps:

Step 1: Codebook Training

A well-trained codebook is constructed by the most representative codewords from a large-enough training set and according to a specified codebook size. There are many codebook training algorithms, but the most famous one is the Linde, Buzo and Gray (LBG) algorithm presented in 1980 [11]. The LBG algorithm partitions the training set into subsets and iteratively refines the desired codebooks by utilizing an initial guess and achieving fast convergence, ultimately leading to the discovery of the most characteristic codewords.

Step 2: Encoding

As shown in Figure 1, a mapped index table is generated using Euclidean distance calculation. After obtaining a well-trained codebook, the input image I is divided into non-

overlapping square blocks, with each block consisting of $p \times p$ pixels and representing a p^2 vector. These blocks are then replaced by codewords to achieve the purpose of compression.

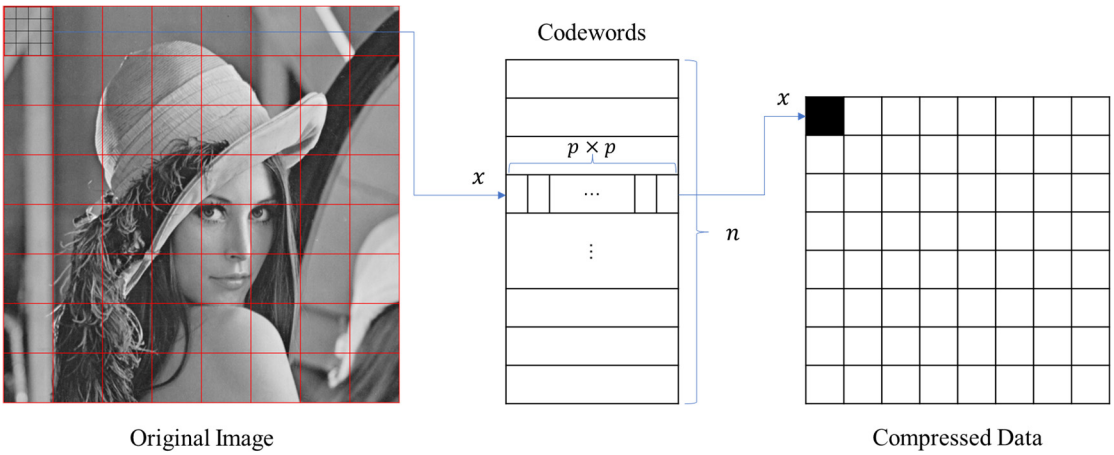

Figure 1. Encoding step of VQ.

As shown in Equation (1), i represents the vector dimension being processed. The set of vectors is denoted as follows: $V = \{ v_0, v_1, \cdots, v_{m-1} \}$, where m represents the total number of vectors in the input image I, and j represents the vector being processed. If CB is a codebook of size n, $CB = \{ cw_0, cw_1, \cdots, cw_{n-1} \}$, and k represents the codeword being processed; each codeword cw_k can be represented as $cw_k = \left(p_{j1}, p_{j2}, \cdots, p_{jp^2} \right)$. The Euclidean distance d_{VQ} between an input vector v_j and cw_k needs to be minimized in order to keep the distortion of the input image to a minimum.

$$d_{VQ} = min_{k=1}^{n} \sqrt{\sum_{i=1}^{p \times p} \left(v_{ji} - cw_{ki} \right)^2} \quad (1)$$

Through the VQ algorithm, the index of the codeword with the minimum Euclidean distance is recorded in the index table and transmitted to the decoder instead of the pixel values in a $p \times p$ square block. This approach significantly reduces the image size since the bit size of a codeword is much smaller than that of a 16-dimensional vector. As a result, a high compression rate is achieved.

Step 3: Decoding

The well-trained codebook and the index table generated by the encoder are sent to the decoder. The codebook used for decoding is exactly the same as the one used for encoding. Therefore, a simple table look-up allows for the retrieval of the original image with minor distortion. The computational load is light, and the image quality can be controlled by using a larger codebook size. The larger the codebook size, the higher is the recovered image quality. However, as with many principles in life, there is a drawback: the larger the codebook, the lower the compression rate.

2.2. Search-Order Coding Algorithm

SOC [24] is a technique proposed by Hsieh and Tsai in 1996 for achieving further compression without causing image distortion. The SOC algorithm aims to further compress VQ compressed data. It utilizes a raster scan approach, moving from left to right and top to bottom. The current search point serves as the search center (SC), and a spiral search is conducted along the search path. The search-order code is expressed in binary, where n represents the number of bits, and the search range is 2^n. Typically, the SOC algorithm performs best when n is set to 2 or 3. If an index value equal to the center is found, the new SOC index can replace the VQ index, resulting in recompression. If a match is not

found, the original VQ index is still used for representation. To differentiate between the two cases, an additional 1-bit indicator is required. Due to the high degree of similarity between neighboring blocks, the SOC algorithm is capable of successfully searching for a large number of indices, effectively achieving the goal of recompression.

As shown in Figure 2, the SOC algorithm defines four starting search points (SSPs), which can be used as needed. If a fixed SSP is preset and remains unchanged, no additional auxiliary information is required. However, if the SSP needs to be determined dynamically, an additional bit is required as an indicator to record the SSP.

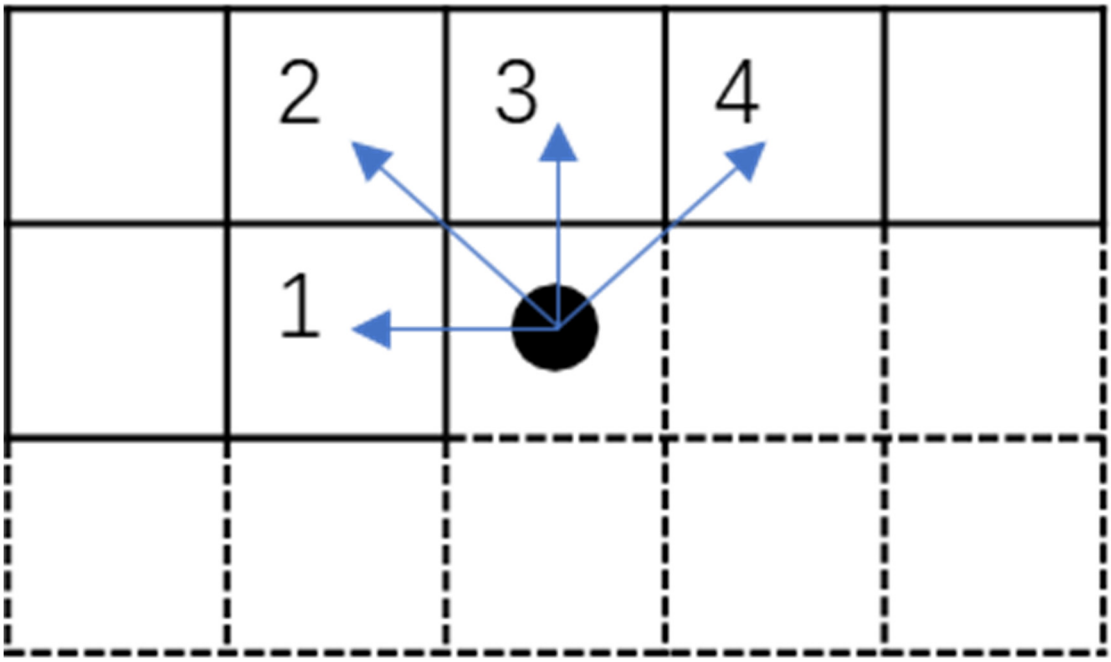

Figure 2. Four starting search points.

Figure 3 depicts two examples at different SSPs, specifically the cases where the SSP is 1 and 4. The search is performed starting from the block currently being processed, following the spiral search path originating from the respective SSP. Since the SOC algorithm operates based on raster search, we can only search blocks that have been processed. Blocks that have not been processed cannot participate in the search, as doing so would result in an irreversible final index with respect to the original VQ index. In the figure, we use a black circle to represent the SC and a gray block to represent the SSP. Blocks with solid borders represent points that can be searched, while blocks with dashed borders represent points that cannot be searched. We connect the points with blue arrows to represent the search path.

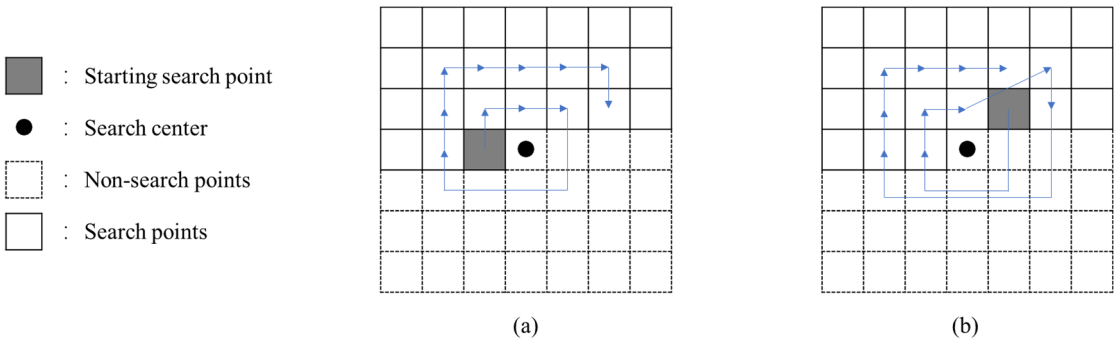

Figure 3. Search path of SOC algorithm at different SSPs. (**a**) SSP = 1. (**b**) SSP = 4.

Figure 4 provides an example with $n = 2$, where the blue square represents the starting search center with an index value of 44. The solid blocks indicate valid search points. Each search step is encoded in binary format. In the starting step, "00", the value is 48, which is not the target value of 44. The search continues, and the step count increases until either a 44 is found or the predetermined 2-bit tolerance search count is reached. If an index value appears earlier in the search path, it signifies a repetition that can be skipped, and the search count is not increased. In the example shown in Figure 4, a match of 44 is found in the last step, which corresponds to "11". Therefore, we can substitute the SOC index for the VQ index. Assuming a codebook size of 256, which represents the VQ index using 8 bits, we can use "11" to replace the binary representation "00101100" for the value 44.

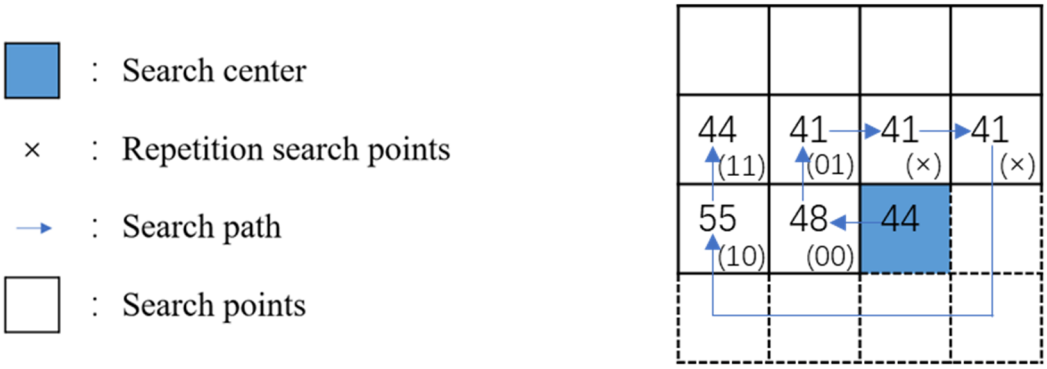

Figure 4. An example of an SOC algorithm.

The decoding step also follows the order of raster scanning. Upon receiving the compressed code, the decoder can differentiate between the SOC index and the original VQ index based on the index indicator. It then performs a search process similar to encoding, utilizing the same SSP and n as used in encoding. Finally, the original VQ index can be restored and the VQ codebook enables the reconstruction of the image.

2.3. Side Match Vector Quantization Algorithm

In 1992, Kim proposed SMVQ [13], which utilizes the high correlation between neighboring blocks of an image to search for indices based on edge similarity. Similar to the SOC algorithm, when processing a block based on raster scanning, it can only refer to previously processed blocks. Since the raster scan proceeds from left to right and top to bottom, the side match algorithm utilizes the upper and left neighboring blocks for matching.

The concept of SMVQ is illustrated in Figure 5. X is the block currently being processed, while the upper neighboring block U and left neighboring block L have already been processed. All pixel values in U and L are known and remain unchanged. Therefore, we utilize the information from U and L to find a suitable codeword from the codebook in order to restore X. Except for the pixel X_{11}, which is adjacent to both blocks and needs to be matched with both U_{m1} and L_{1n}, the other pixels that are only adjacent to one block just need to match their corresponding neighboring pixels. For example, $\{U_{m2}, U_{m3}, \ldots, U_{mn}\}$ matches $\{X_{12}, X_{13}, \ldots, X_{1n}\}$, and $\{L_{1n}, L_{1n}, \ldots, L_{1n}\}$ matches $\{X_{21}, X_{31}, \ldots X_{m1}\}$. Afterwards, their Euclidean distances are calculated and sorted in ascending order. The codeword closest to the current block is found from the codebook to restore X. Through SMVQ, the bit rate can be effectively reduced. The specific algorithm and examples are explained in Section III.

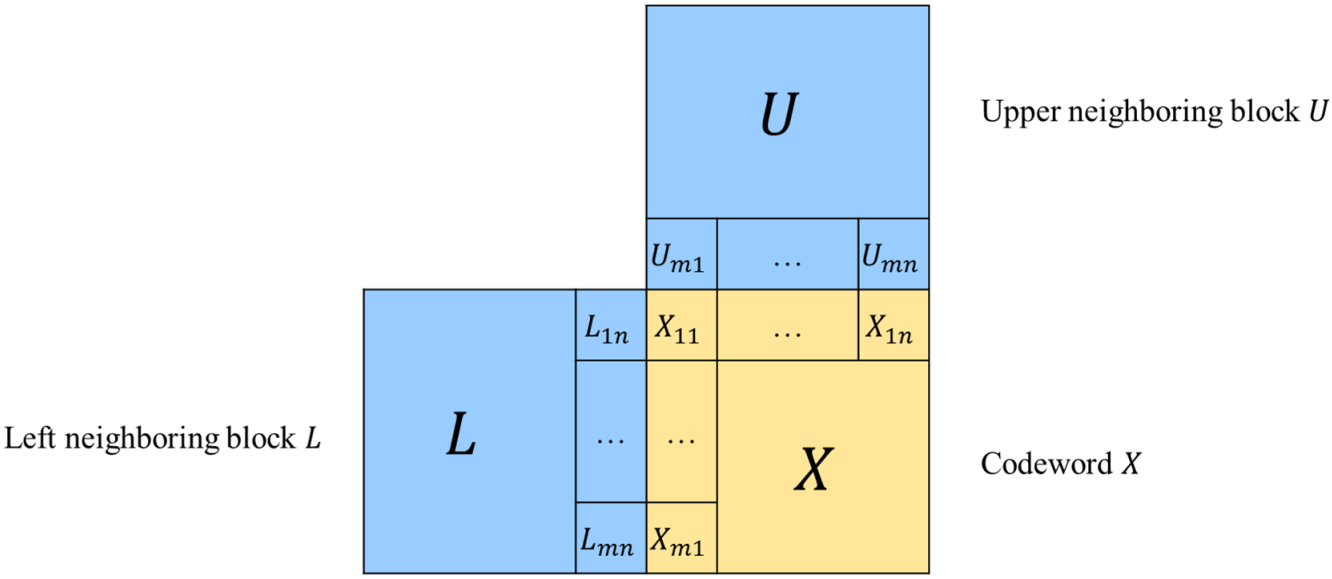

Figure 5. The concept of SMVQ.

2.4. State Codebook Scheme

When using the SOC scheme, a desired matched index value may not be found in the search path. Chang et al. proposed a variation, derived from the original SOC, which was introduced by adding on a state codebook [26]. These state codebooks are smaller-sized codebooks derived from the original codebook. The main idea is to expand a smaller-sized codebook from each index number in the search path. These local state codebooks contain more indices which are the closest to the original index. The size of the state codebook depends on the predefined similar index count 2^i in SOC and the closest index count 2^j, where i and j are limited bits needed for coding. The expansion amount will be limited to bits to be varied. During local state codebook building, if index values appear in the SOC search path or in previous local state codebooks, the index values are skipped, and the process continues until it finds the next available index value.

Figure 6 presents an example where a matched index value cannot be found in the SOC search path, but a match is found in one of the local state codebooks derived from the values in the SOC path. The search direction is predefined as 1, and the number of bits for SOC is set to 2. The index value of (3, 3) is 60, and there are no repeated values in its SOC search path. From the indices of (2, 3), (2, 2), (4, 2), and (1, 3), the index values are 55, 54, 52, and 46, respectively. Assuming the expansion i and j are limited to 2 bits, the size of the local state codebook becomes $2^2 \times 2^2 = 16$. Table 1 displays the closest index values for each search point in the SOC algorithm. If we take 55 as a base to find its closest values, the closest four values will be 53, 54, 56, and 57. Since 54 already appeared in the SOC search path, it is skipped, and 52 is found instead. However, 52 is also in the SOC search path, so it is skipped, and 51 is chosen instead. By expanding 54, 52, and 46 in the same manner, we can finally obtain a complete state codebook without repeated codewords. Once the local state codebook is built, a match is found for the index value 60 in the local state codebook derived from the index value 52 located at (4, 2). If the size of the VQ codebook is 256, each codeword is represented by 8 bits, and the index "1010" of the state codebook is used to replace the binary "00111100" corresponding to the original VQ index value 60.

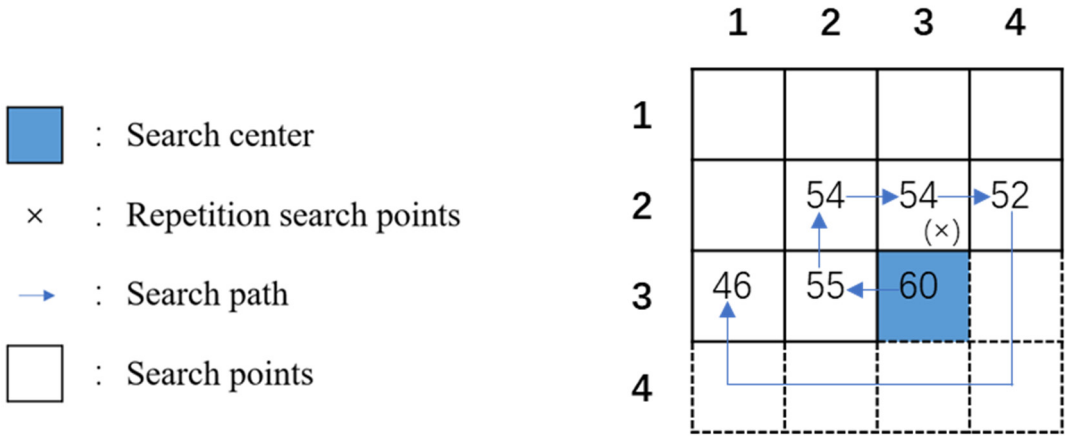

Figure 6. SOC and its local state codebook.

Table 1. The state codebook of Figure 6.

The similar neighboring (3, 2), which equals "55"	
The index of the state codebook	The indices of the closest codewords of "55" in the codebook
0000	51
----	52
0001	53
----	54
0010	56
0011	57
The similar neighboring (2, 2), which equals "54"	
The index of the state codebook	The indices of the closest codewords of "54" in the codebook
0100	49
0101	50
0110	58
0111	89
The similar neighboring (2, 4), which equals "52"	
The index of the state codebook	The indices of the closest codewords of "52" in the codebook
1000	47
1001	48
1010	60
1011	61
The similar neighboring (3, 1), which equals "46"	
The index of the state codebook	The indices of the closest codewords of "46" in the codebook
1100	44
1101	45
1110	62
1111	63

3. Proposed Scheme

In this section, we present a scheme that combines the SOC algorithm with the side match algorithm, allowing for a higher number of codewords to be searched and significantly improving the compression rate.

In our proposed scheme, we classify all the indices in the index table into three types. The first type consists of indices generated by the SOC algorithm for search-order coding. The second type comprises indices generated by the side match algorithm for match order coding. The third type includes indices representing the original VQ codewords, excluding the first two types. To enable the decoder to distinguish between these three types in the compressed code, an additional indicator should be added in front of the compressed code for each index. Since we only have three types, we can use one or two bits to represent them. Based on the experimental results, it was observed that the first type generally allows for a greater number of codewords to be searched. Consequently, for achieving more efficient compression, it is

common to utilize shorter designators for the more frequently occurring types. Specifically, we commonly assign "0" to the first type, "10" to the second type, and "11" to the third type.

Subsequently, we elucidate the three types of situations by providing concrete examples as follows:

In the given examples, we train a codebook with 256 codewords, and each codeword is a 16-dimensional vector. The SOC algorithm operates according to the rule that SSP equals 1, and the bit number n of the search-order code is equal to 2.

Case 1: The index is found by the SOC algorithm.

As shown in Figure 7, the index equal to the SC is located at (3, 1). Additionally, we observe that the indices at (2, 3) and (2, 4) are duplicates, both equal to 41, which is the same as the index at (2, 2). Hence, they can be ignored. By searching sequentially, we find that the search-order code corresponding to the SC is "10". Since the first type of indicator is "0", the final index value is "010".

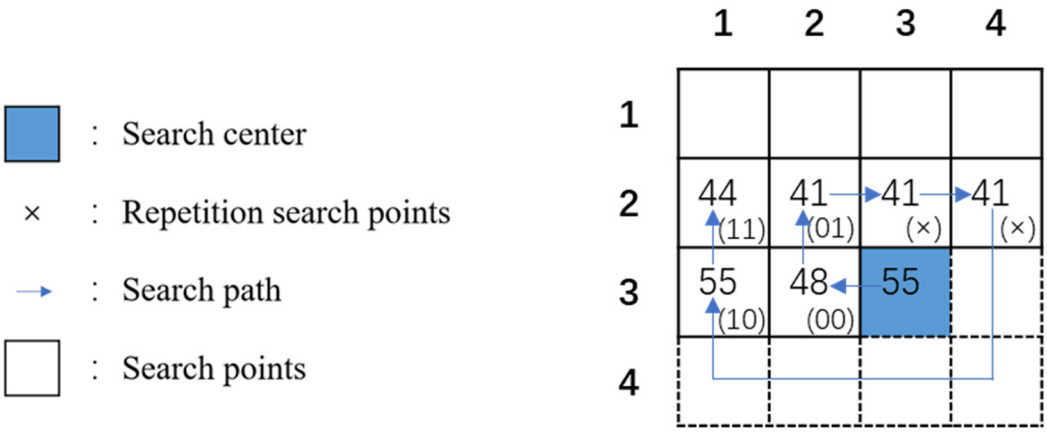

Figure 7. Example of Case 1.

Case 2: The SOC algorithm does not find it, and the index is found through the side match algorithm.

As shown in Figure 8, because the blocks of the image are searched in raster scan order, for the processed block P, the indices of the upper neighboring block U and left neighboring block L have been restored. We need to use the data of U and L to find some of the closest codewords from the codebook to restore P. We define the pixels adjacent to U and L in P as $P_1 = \frac{U_{41}+L_{14}}{2}$, $P_2 = U_{42}$, $P_3 = U_{43}$, $P_4 = U_{44}$, $P_5 = L_{24}$, $P_6 = L_{34}$, and $P_7 = L_{44}$. We extract the codewords from the codebook one by one. For each 16-dimensional codeword, we convert it into a 4×4 calculated block C. For simplicity, we replace their coordinates with numbers. Specifically, we only need to compare the values of these few side positions with C, so we define $C_1 = C_{11}$, $C_2 = C_{12}$, $C_3 = C_{13}$, $C_4 = C_{14}$, $C_5 = C_{21}$, $C_6 = C_{31}$, and $C_7 = C_{41}$. Then, based on Equation (2), we can calculate the Euclidean distance d between the 7 side pixels of P and C. In other words, our scheme is reversible to the original VQ index, while SMVQ is lossy in terms of compression quality. We record the corresponding codeword index and the Euclidean distance. By calculating all the codewords and P one by one, we obtain a side match state codebook containing 256 codeword indices and their corresponding Euclidean distances. Finally, we sort the table based on the Euclidean distance, enabling us to find the closest codeword within the specified matching range. We use r to represent the bit number of the index corresponding to the matching range; the size of the side match state codebook is 2^r. If $r = 4$, the 16 codewords with the closest distance can be matched. Similarly, if $r = 5$, the 32 codewords with the closest distance can

be matched. Within the specified limit of r, if the same index value as P can be found from the side match state codebook, then P belongs to the second type.

$$d_{SM} = \sqrt{\sum_{i=1}^{7}(P_i - C_i)^2}. \quad (2)$$

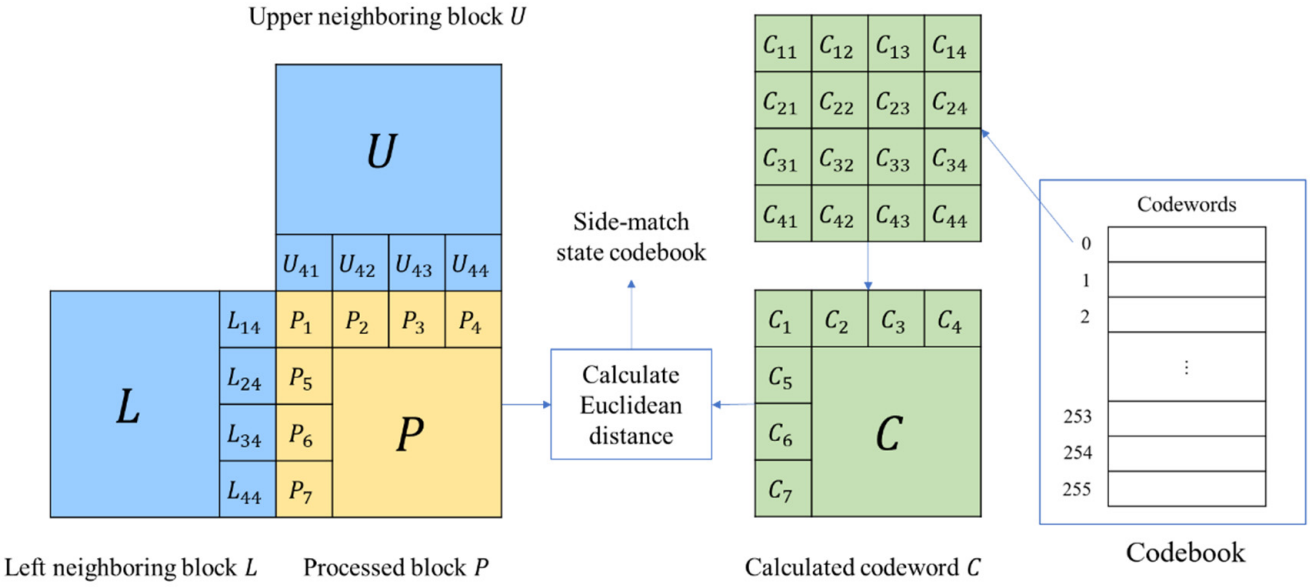

Figure 8. Construction of side match state codebook.

Figure 9 and Table 2 show a complete side match search process. As illustrated in Figure 9, after searching with the SOC algorithm, the same index as the SC is not found. Therefore, we need to use the side match algorithm for the search. Based on the data of the upper neighboring block (2, 3) and left neighboring block (3, 2) of the SC, we can calculate the side match state codebook, as shown in Table 2. In the table, we can find the same codeword index value "65" as the SC, and its corresponding side match index is "1100". Since the indicator of the second type is "10", the final index value is "101100".

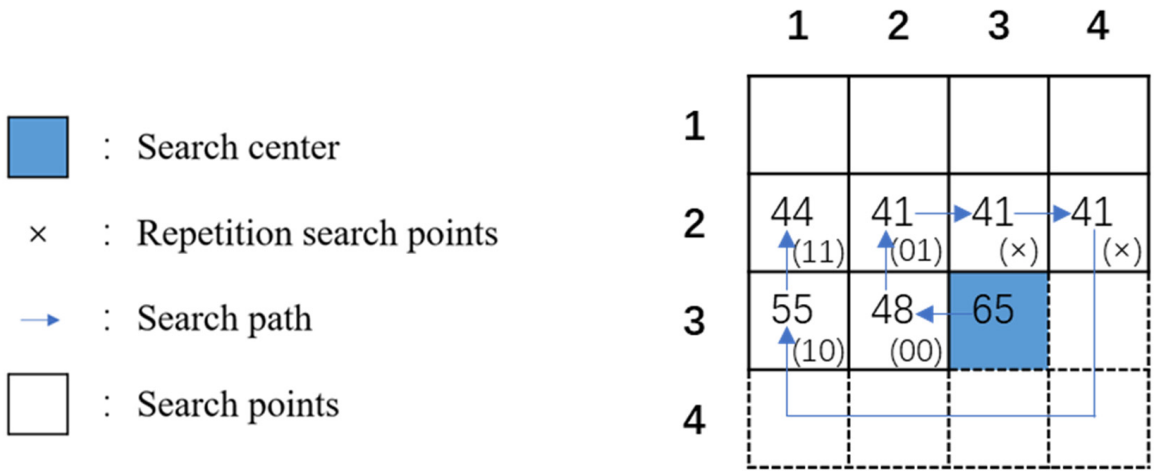

Figure 9. Example of Case 2.

Table 2. The side match state codebook of Figure 9 or Figure 10.

Side Match Index	Codeword Index	Euclidean Distance
0000	44	5.62
0001	86	7.12
0010	55	26.84
0011	77	72.23
0100	48	72.77
0101	41	108.08
0110	89	118.02
0111	73	191.51
1000	96	211.50
1001	83	219.61
1010	79	226.16
1011	75	259.82
1100	65	272.74
1101	108	290.89
1110	99	360.81
1111	85	364.79

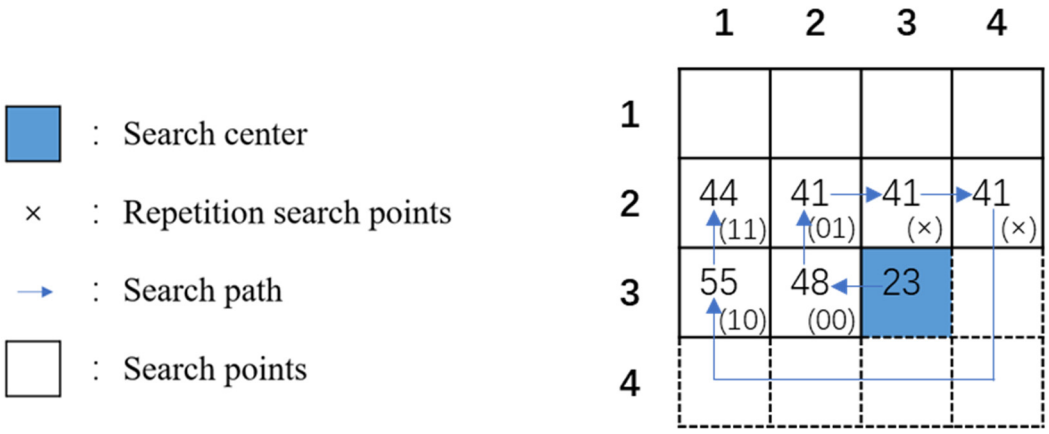

Figure 10. Example of Case 3.

Case 3: Neither the SOC algorithm nor the side match algorithm finds an index, and the original VQ index is used.

Since Figures 9 and 10 differ only in the SC, the other search points remain the same. Consequently, after applying the side match algorithm, their side match state codebooks are identical. Hence, Table 2 can still be utilized as the side match state codebook for Figure 10.

As shown in Figure 10, after executing the SOC algorithm, we did not find the same index value as the SC. After executing the side match algorithm, we also did not find the same codeword index in Table 2. Therefore, we can only use the original VQ index for representation. Since the codebook size is 256, the length of the VQ index is 8 bits. The index of the SC is "23", which is converted into a binary sequence: "00010111". Since the indicator of the third type is "11", the final index value is "1100010111".

When starting the encoding step, all indices are processed one by one. For the index being processed, it is necessary to check first whether it belongs to the first type; that is, the SOC algorithm must be executed to find the same index with the preset search rules. If it can be found, the index belongs to the first type, and uses "0" as its indicator. If not found, a further check is needed to see whether it belongs to the second type. The side match algorithm is executed; if the same index is found in the side match state codebook, then the index belongs to the second type, with "10" as its indicator. If the index belongs to neither the first type nor the second type, "11" is used as its indicator. Note that each index can only be classified into one type, and the proposed algorithm is described in detail as follows:

Input: The image is divided into blocks, and the processing follows a raster scan order.

Output: The result of the compression process is a compression code representing the input image.

Step 1: For each block being processed currently, the traditional VQ coding scheme is applied to produce its corresponding index. If the index falls under the category of the first type, the compression code is formed by appending a one-bit indicator ("0" is commonly used) followed by the search-order codes. The process advances to Step 4 afterwards. However, if the index belongs to a different type, the algorithm proceeds to Step 2 for further handling.

Step 2: Should the index be classified as the second type, it is encoded by adding a two-bit indicator ("10" is commonly used) followed by the index of the side match state codebook. Subsequently, Step 4 is followed. If the index does not fall into the second type category, Step 3 is continued

Step 3: This index belongs to the third type. A two-bit indicator is outputted ("11" is commonly used) followed by the original VQ index value and Step 4 is continued.

Step 4: Steps 1 to 4 are then executed iteratively until the input has processed all blocks within the image.

The decoder also follows the raster order to decode. Based on the indicator received for each index, the decoder can determine the type of each index. If the indicator corresponds to the first type, the index will be restored using the SOC algorithm. If the indicator corresponds to the second type, the index will be restored using the side match algorithm. If the indicator corresponds to the third type, the index represents the original VQ index value.

4. Experimental Results

To evaluate the performance of our proposed scheme, we compared it with the SOC algorithm and the state codebook algorithm. The experiments were conducted using MATLAB R2024a on a Windows 11 laptop equipped with a 3.20 GHz AMD Ryzen 7 CPU and 16 GB of RAM. As shown in Figure 11, we used six 512 × 512 size 256 gray standard images: Elaine, Lena, Pepper, Wine, Woodland, and Zelda. For our codebook, we trained these six images 100 times using the LBG algorithm [11], with each image block sized at 4 × 4 pixels.

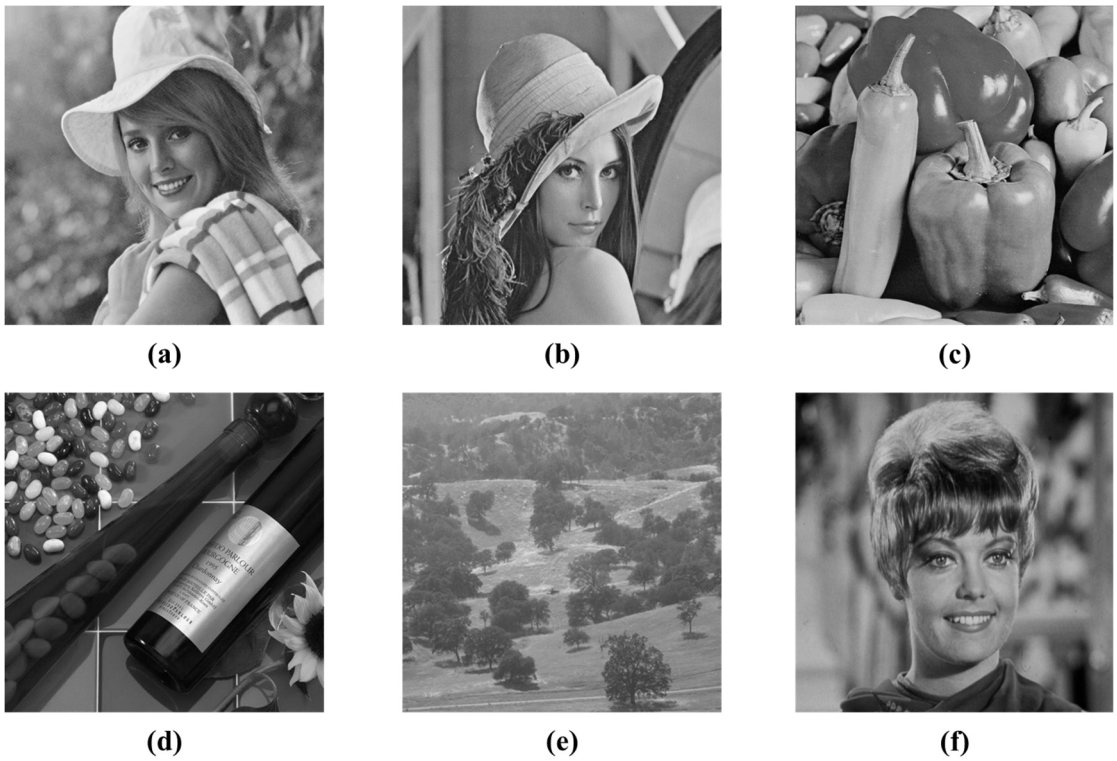

Figure 11. Standard images sized 512 × 512 pixels. (**a**) "Elaine". (**b**) "Lena". (**c**) "Pepper". (**d**) "Wine". (**e**) "Woodland". (**f**) "Zelda".

In order to compare the compression rates of different schemes, we use the bit rate (i.e., bits per pixel) as a metric. Equation (3) to Equation (5), respectively, show the bit rate of different schemes. We suppose we have an image of size $M \times N$, and the trained codebook size is N_c. Equation (3) shows the bit rate of the SOC algorithm, where n represents the bit number of the search-order code; num_{SOC} is the number of SOC indices used, and num_{VQ} is the number of VQ indices used. Equation (4) shows the bit rate of the state codebook algorithm, where num_{SC} is the number of state codebook indices, i represents the search range of the state codebook or the number of bits of the small state codebook, and j represents the number of bits in each small state codebook for the codeword quantity. Equation (5) shows the bit rate of the side match algorithm, where num_{SM} is the number of side match indices, and r represents the number of bits in the matching range, usually set to $r = 4$.

$$BR_{SOC} = \frac{num_{SOC} \times (1+n) + num_{VQ}(1+log_2 N_c)}{M \times N}, \tag{3}$$

$$BR_{SC} = \frac{num_{SOC} \times (1+n) + num_{SC} \times (2+i+j) + num_{VQ} \times (2+log_2 N_c)}{M \times N}, \tag{4}$$

$$BR_{SM} = \frac{num_{SOC} \times (1+n) + num_{SM} \times (2+r) + num_{VQ} \times (2+log_2 N_c)}{M \times N} \tag{5}$$

For the sake of fairness in the experiment, we make comparisons in the cases of $n = 2$, $n = 3$, and the best n, respectively. Here, n represents the number of bits for the SOC index, as the best n values vary for different methods. In addition to fixing n at 2 and 3 for comparison, the best n represents the value of n in the best-case scenario for the bit rate. Table 3 shows the comparison of bit rates and improvement rates using a codebook with 256 codewords. Please be aware that in the context of traditional VQ compression, when the codebook size is 256, the bit rate is 0.5. It can be seen from Table 3 that our method is significantly better than the SOC algorithm, with an average improvement rate as high as 13.41% in the case of $n = 2$. Additionally, the bit rate of our scheme is significantly lower than that of the state codebook algorithm. The main reason for this is that the side match algorithm relies on edge similarity. This approach can represent the details and differences between image blocks more accurately. This enables the algorithm to find more accurately matching indices.

Table 3. Comparison of the bit rates of other schemes with the proposed scheme using a VQ codebook with 256 codewords.

$n = 2$					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.4043	15.35%	0.3880	11.78%	0.3423
Lena	0.3662	10.28%	0.3538	7.13%	0.3286
Pepper	0.3726	12.27%	0.3602	9.25%	0.3269
Wine	0.3545	10.44%	0.3393	6.43%	0.3175
Woodland	0.4322	15.41%	0.4088	10.56%	0.3656
Zelda	0.4007	15.86%	0.3901	13.58%	0.3371
Average	0.3884	13.41%	0.3733	9.92%	0.3363
$n = 3$					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.3940	11.39%	0.3858	9.51%	0.3491
Lena	0.3697	6.52%	0.3644	5.16%	0.3456
Pepper	0.3760	8.43%	0.3704	7.04%	0.3443
Wine	0.3678	7.27%	0.3597	5.18%	0.3411
Woodland	0.4113	11.18%	0.3988	8.40%	0.3653
Zelda	0.3939	11.88%	0.3887	10.70%	0.3471
Average	0.3854	9.52%	0.3780	7.73%	0.3487

Table 3. *Cont.*

best n					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.3940	13.12%	0.3858	11.28%	0.3423
Lena	0.3662	10.28%	0.3538	7.13%	0.3286
Pepper	0.3726	12.27%	0.3602	9.25%	0.3269
Wine	0.3545	10.44%	0.3393	6.43%	0.3175
Woodland	0.4113	11.18%	0.3988	8.40%	0.3653
Zelda	0.3939	14.41%	0.3887	13.26%	0.3371
Average	0.3821	11.99%	0.3711	9.38%	0.3363

The same as for the metric when the codebook size is 256, Table 4 represents the experimental results for codebook size 512, and Table 5 represents the experimental results for codebook size 1024. It is worth noting that the conventional VQ compression bit rates are 0.5625 and 0.625 for codebook sizes 512 and 1024, respectively. From the experimental results, it is evident that the proposed method consistently outperforms the previous methods, demonstrating a significant improvement rate.

Table 4. Comparison of the bit rates of other schemes with the proposed scheme using a VQ codebook with 512 codewords.

$n = 2$					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.5064	14.87%	0.5011	13.97%	0.4311
Lena	0.4501	11.89%	0.4441	10.71%	0.3965
Pepper	0.4558	13.27%	0.4499	12.14%	0.3953
Wine	0.4127	11.49%	0.4055	9.91%	0.3653
Woodland	0.5322	13.16%	0.5270	12.29%	0.4622
Zelda	0.4869	14.42%	0.4902	14.99%	0.4167
Average	0.4740	13.26%	0.4696	12.44%	0.4112
$n = 3$					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.4878	12.27%	0.4881	12.33%	0.4279
Lena	0.4378	8.04%	0.4378	8.03%	0.4026
Pepper	0.4462	9.70%	0.4457	9.60%	0.4029
Wine	0.4196	8.46%	0.4177	8.05%	0.3841
Woodland	0.5112	10.96%	0.5093	10.63%	0.4552
Zelda	0.4743	11.82%	0.4809	13.03%	0.4183
Average	0.4628	10.30%	0.4633	10.38%	0.4152
best n					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.4878	12.27%	0.4881	12.33%	0.4279
Lena	0.4378	9.43%	0.4378	9.41%	0.3965
Pepper	0.4462	11.40%	0.4457	11.30%	0.3953
Wine	0.4127	11.49%	0.4055	9.91%	0.3653
Woodland	0.5112	10.96%	0.5093	10.63%	0.4552
Zelda	0.4743	12.15%	0.4809	13.36%	0.4167
Average	0.4617	11.30%	0.4612	11.22%	0.4095

Figures 12–14 show the bit rate comparison of the proposed scheme for different values of r with $n = 2$ and $n = 3$ when the VQ codebook size is 256, 512, and 1024, respectively. We can observe that when the VQ codebook size is 256 and $r = 4$ (this means 16 matches), the bit rate is the lowest. Similarly, when the VQ codebook size is 1024 and $r = 5$ (this means 32 matches), the bit rate is also the lowest. This is due to the increased number of codewords, resulting in more possible interference items being matched. Therefore, it becomes necessary to expand the scope of matching to increase the likelihood of finding a matching index. Exploiting this characteristic, we conducted an experiment using a

VQ codebook with 1024 codewords. Figures 15 and 16 represent the cases for $n = 2$ and $n = 3$, respectively. We compared the bit rate and improvement for $r = 5$ and $r = 4$. The experimental results demonstrate that employing a wider matching range can effectively enhance the compression effect for larger VQ codebook sizes.

Table 5. Comparison of the bit rates of other schemes with the proposed scheme using a VQ codebook with 1024 codewords.

$n = 2$					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.6024	9.12%	0.6131	10.70%	0.5475
Lena	0.5305	11.00%	0.5412	12.76%	0.4721
Pepper	0.5510	12.16%	0.5583	13.31%	0.4840
Wine	0.4804	11.92%	0.4834	12.47%	0.4231
Woodland	0.6190	8.09%	0.6390	10.97%	0.5689
Zelda	0.5725	11.35%	0.5924	14.32%	0.5076
Average	0.5593	10.51%	0.5712	12.38%	0.5005
$n = 3$					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.5838	7.74%	0.5974	9.83%	0.5386
Lena	0.5130	8.00%	0.5248	10.07%	0.4719
Pepper	0.5339	9.39%	0.5442	11.11%	0.4837
Wine	0.4826	9.26%	0.4887	10.40%	0.4379
Woodland	0.6001	7.03%	0.6199	10.00%	0.5579
Zelda	0.5550	9.71%	0.5747	12.81%	0.5011
Average	0.5447	8.48%	0.5583	10.70%	0.4985
best n					
Images	**SOC** [24]	**Improvement Rate**	**State Codebook** [26]	**Improvement Rate**	**Ours**
Elaine	0.5838	7.74%	0.5974	9.83%	0.5386
Lena	0.5130	8.00%	0.5248	10.07%	0.4719
Pepper	0.5339	9.39%	0.5442	11.11%	0.4837
Wine	0.4804	11.92%	0.4834	12.47%	0.4231
Woodland	0.6001	7.03%	0.6199	10.00%	0.5579
Zelda	0.5550	9.71%	0.5747	12.81%	0.5011
Average	0.5444	8.87%	0.5574	11.00%	0.4961

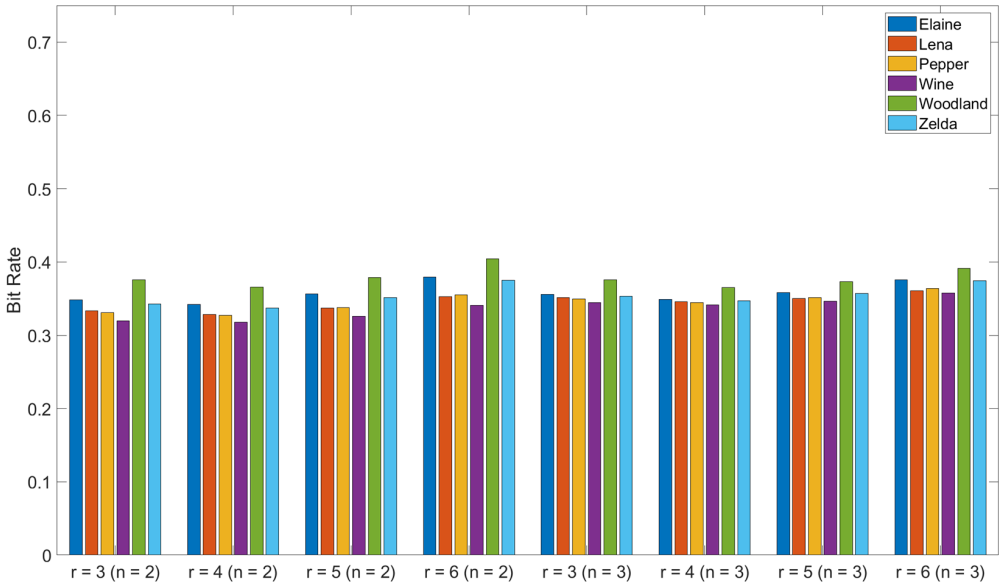

Figure 12. Comparison of the bit rates of the proposed scheme using a VQ codebook with 256 codewords for different values of r at $n = 2$ and $n = 3$.

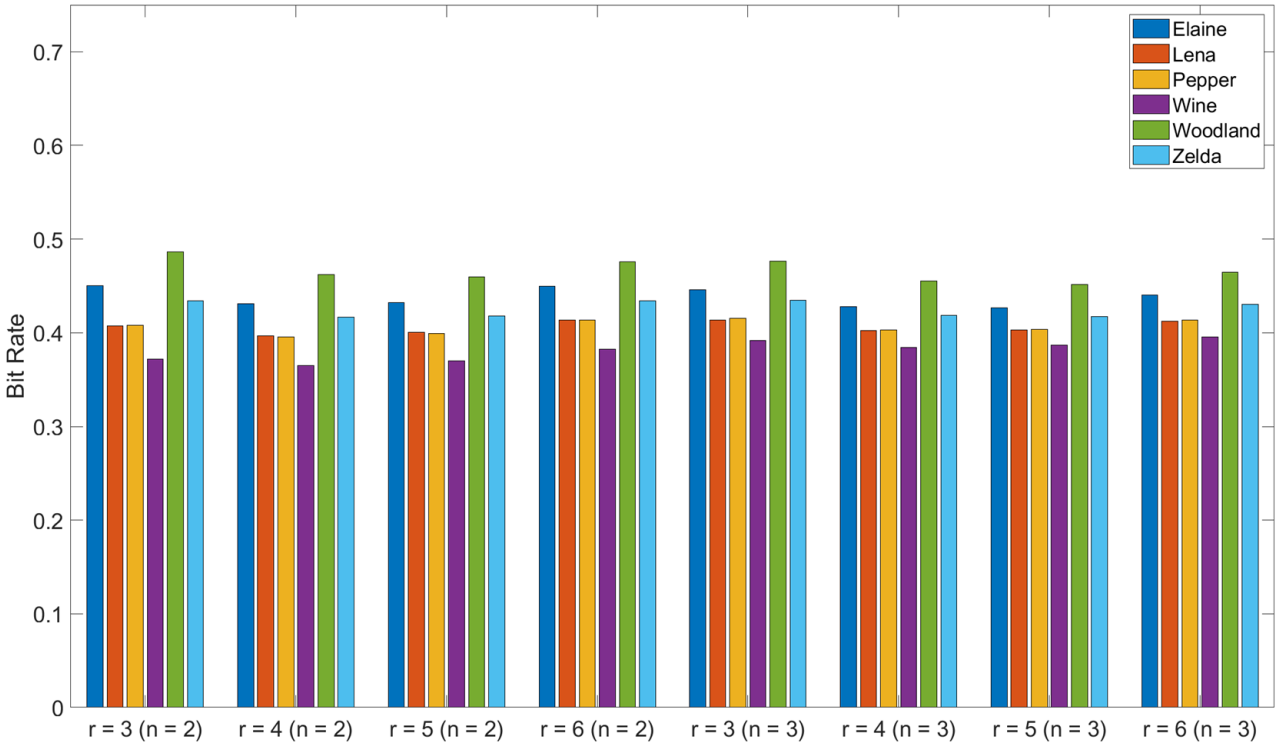

Figure 13. Comparison of the bit rates of the proposed scheme using a VQ codebook with 512 codewords for different values of r at $n = 2$ and $n = 3$.

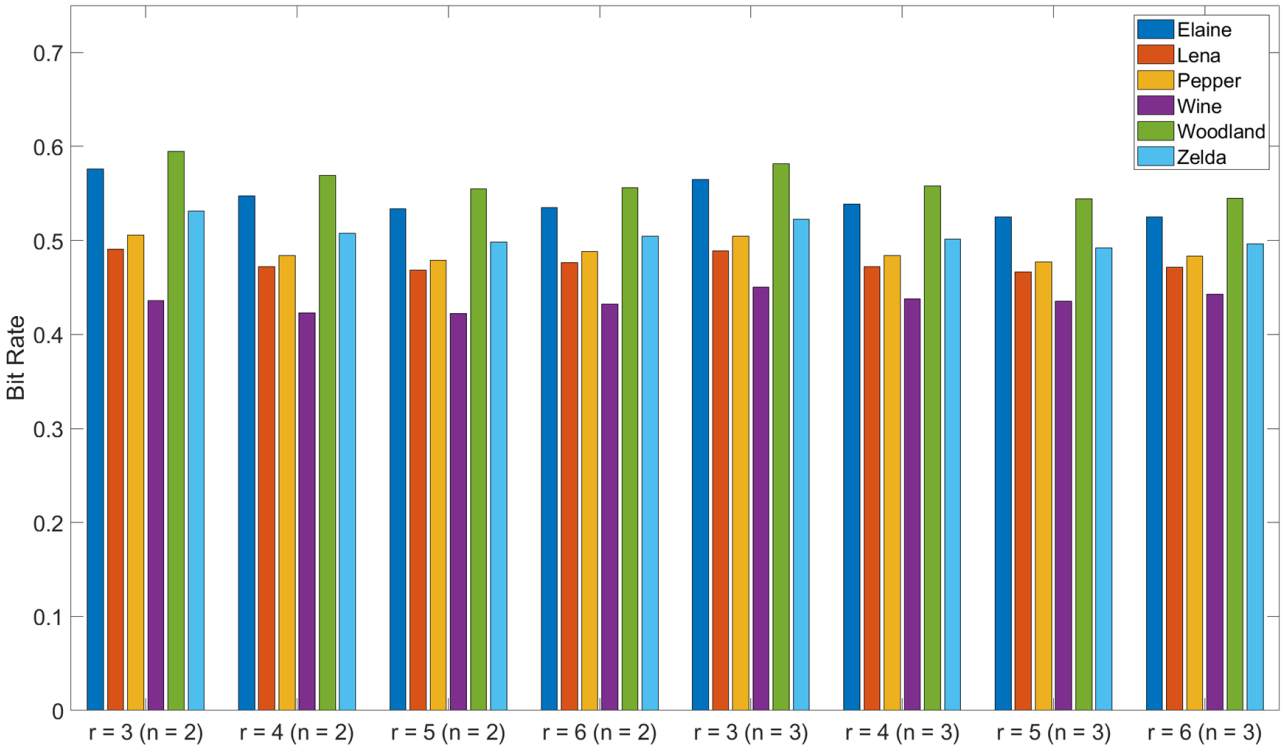

Figure 14. Comparison of the bit rates of the proposed scheme using a VQ codebook with 1024 codewords for different values of r at $n = 2$ and $n = 3$.

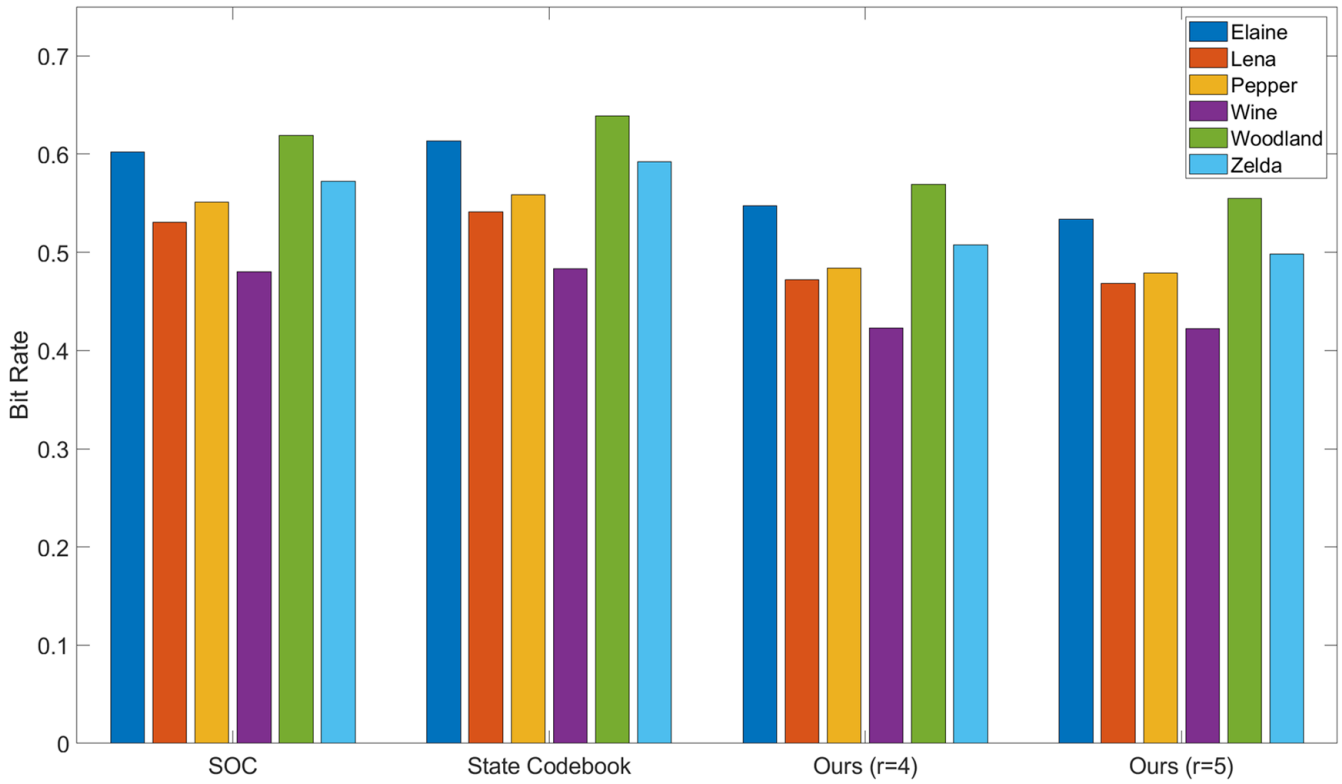

Figure 15. Comparison of the bit rates of other schemes and the proposed scheme ($r = 4$) with the proposed scheme ($r = 5$) using a VQ codebook with 1024 codewords at $n = 2$.

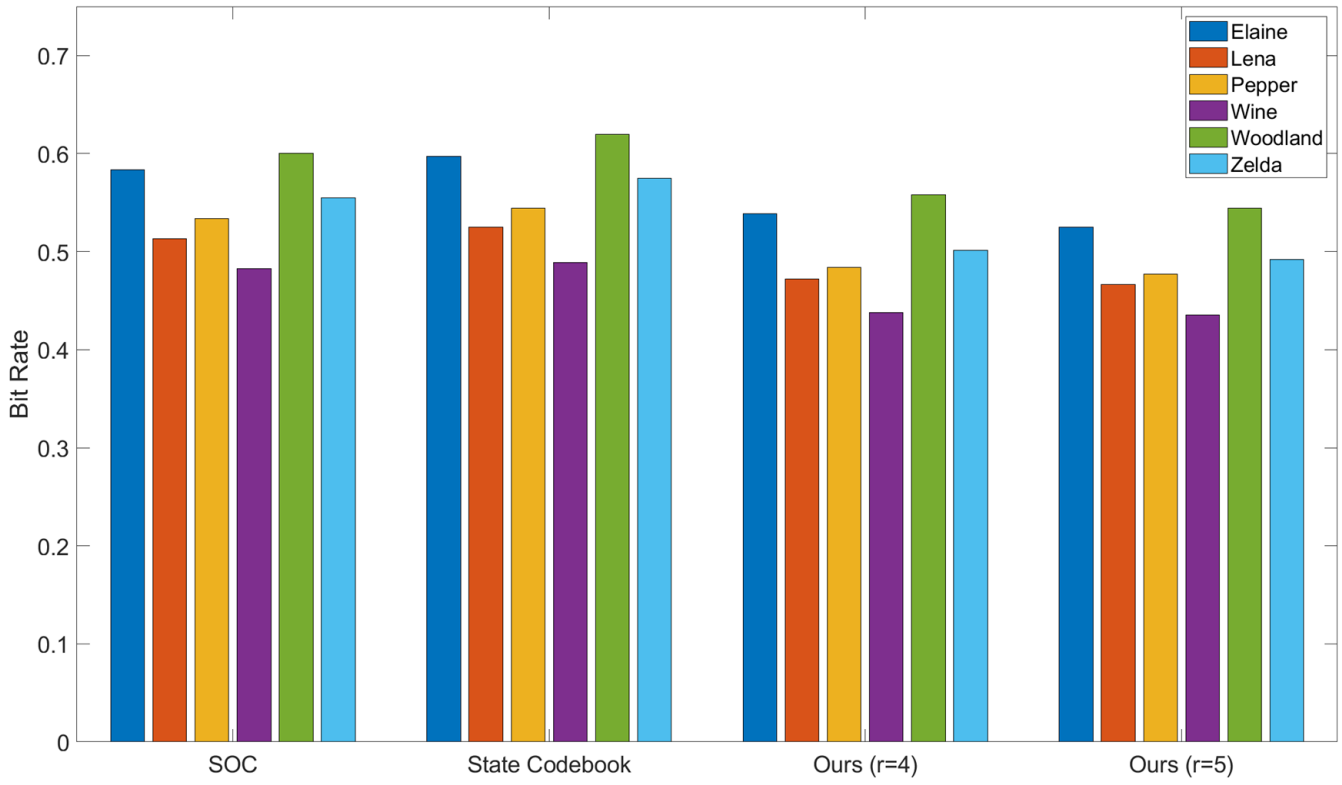

Figure 16. Comparison of the bit rates of other schemes and the proposed scheme ($r = 4$) with the proposed scheme ($r = 5$) using a VQ codebook with 1024 codewords at $n = 3$.

In order to compare JPEG 2000 (Lossless Mode) [28] and PNG [29] compression techniques, we set the codebook size to 256 and selected images from the BOWS-2 dataset [30] for VQ codebook training. Figure 17 shows the comparison of the average bit rate results on the BOWS-2 dataset, where SOC, state codebook, and our scheme set $n = 2$. In the comparison, our scheme also sets $r = 4$. We can see that our proposed scheme still achieves the best compression effect with a bit rate of 0.3234.

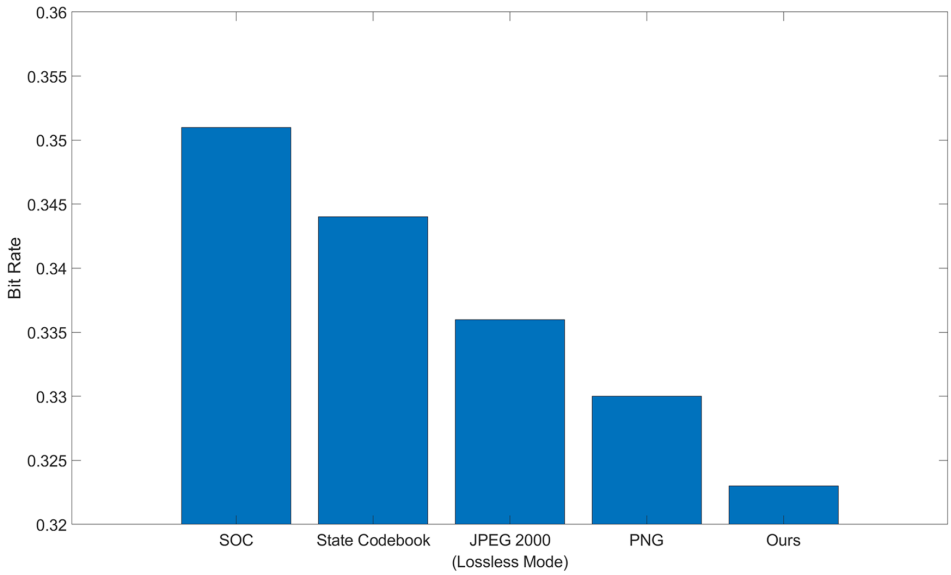

Figure 17. Comparison of the bit rates of other schemes with the proposed scheme using a VQ codebook with 256 codewords in the BOWS-2 dataset.

5. Conclusions

This paper presents a novel VQ index compression method which was designed based on the high correlation and edge similarity between neighboring blocks of the image. Our method initially employs the SOC algorithm to search indices with the same value among nearby indices. Subsequently, we utilize the side match algorithm to calculate the Euclidean distance between codewords and the edge pixels of the neighboring block being processed, enabling us to find the most similar codeword. This approach allows us to compress the traditional VQ index and achieve lossless restoration during decoding. The experimental results demonstrate that our method outperforms previous approaches, effectively compressing the VQ index and reducing the bit rate. In the future, we will study different carriers and investigate more efficient compression methods based on the characteristics of the trained VQ index tables.

Author Contributions: Conceptualization, Y.-J.L., J.-C.L., and C.-C.C. (Chin-Chen Chang); methodology, Y.-J.L., J.-C.L., and C.-C.C. (Chin-Chen Chang); software, Y.-J.L.; validation, Y.-J.L.; writing—original draft preparation, Y.-J.L. and J.-C.L.; writing—review and editing, Y.-J.L., J.-C.L., C.-C.C. (Chin-Chen Chang), and C.-C.C. (Ching-Chun Chang). All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Liu, L.; Xie, Z.; Yang, C. A novel iterative thresholding algorithm based on plug-and-play priors for compressive sampling. *Future Internet* **2017**, *9*, 24. [CrossRef]
2. Ma, H.; Liu, D.; Yan, N.; Li, H.; Wu, F. End-to-end optimized versatile image compression with wavelet-like transform. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *44*, 1247–1263. [CrossRef] [PubMed]

3. Mentzer, F.; Gool, L.V.; Tschannen, M. Learning better lossless compression using lossy compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 6638–6647. [CrossRef]
4. Chang, C.C.; Chang, J.F.; Kao, W.J.; Horng, J.H. Two-Layer Reversible Data Hiding for VQ-Compressed Images Based on De-Clustering and Indicator-Free Search-Order Coding. *Future Internet* **2021**, *13*, 215. [CrossRef]
5. Razavi, A.; Van den Oord, A.; Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. *Adv. Neural Inf. Process. Syst.* **2019**, *32*. [CrossRef]
6. Kwon, J.; Lee, S. Improving the robustness of model compression by on-manifold adversarial training. *Future Internet* **2021**, *13*, 300. [CrossRef]
7. Kang, N.; Qiu, S.; Zhang, S.; Li, Z.; Xia, S.T. Pilc: Practical image lossless compression with an end-to-end gpu oriented neural framework. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 14–19 June 2022; pp. 3739–3748.
8. Duan, Z.; Lu, M.; Ma, J.; Huang, Y.; Ma, Z.; Zhu, F. Qarv: Quantization-aware resnet vae for lossy image compression. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *46*, 436–450. [CrossRef] [PubMed]
9. Gray, R. Vector quantization. *IEEE Assp Mag.* **1984**, *1*, 4–29. [CrossRef]
10. Nasrabadi, N.M.; King, R.A. Image coding using vector quantization: A review. *IEEE Trans. Commun.* **1988**, *36*, 957–971. [CrossRef]
11. Linde, Y.; Buzo, A.; Gray, R. An algorithm for vector quantizer design. *IEEE Trans. Commun.* **1980**, *28*, 84–95. [CrossRef]
12. Dunham, M.; Gray, R. An algorithm for the design of labeled-transition finite-state vector quantizers. *IEEE Trans. Commun.* **1985**, *33*, 83–89. [CrossRef]
13. Kim, T. Side match and overlap match vector quantizers for images. *IEEE Trans. Image Process.* **1992**, *1*, 170–185. [CrossRef] [PubMed]
14. Chang, R.F.; Chen, W.T. Image coding using variable-rate side-match finite-state vector quantization. *IEEE Trans. Image Process.* **1993**, *2*, 104–108. [CrossRef] [PubMed]
15. Chang, R.F.; Chen, W.M. Adaptive edge-based side-match finite-state classified vector quantization with quadtree map. *IEEE Trans. Image Process.* **1996**, *5*, 378–383. [CrossRef] [PubMed]
16. Chen, T.S.; Chang, C.C. A new image coding algorithm using variable-rate side-match finite-state vector quantization. *IEEE Trans. Image Process.* **1997**, *6*, 1185–1187. [CrossRef] [PubMed]
17. Wei, H.C.; Tsai, P.C.; Wang, J.S. Three-sided side match finite-state vector quantization. *IEEE Trans. Circuits Syst. Video Technol.* **2000**, *10*, 51–58. [CrossRef]
18. Chang, C.C.; Liao, C.T. An image coding scheme using SMVQ and support vector machines. *Neurocomputing* **2006**, *69*, 2327–2335. [CrossRef]
19. Gray, R.; Linde, Y. Vector quantizers and predictive quantizers for Gauss-Markov sources. *IEEE Trans. Commun.* **1982**, *30*, 381–389. [CrossRef]
20. Hang, H.M.; Woods, J. Predictive vector quantization of images. *IEEE Trans. Commun.* **1985**, *33*, 1208–1219. [CrossRef]
21. Nasrabadi, N.M.; Feng, Y. Image compression using address-vector quantization. *IEEE Trans. Commun.* **1990**, *38*, 2166–2173. [CrossRef]
22. Lo, K.T.; Feng, J. Predictive mean search algorithms for fast VQ encoding of images. *IEEE Trans. Consum. Electron.* **1995**, *41*, 327–331. [CrossRef]
23. Shanbehzadeh, J.; Ogunbona, P.O. Index-compressed vector quantisation based on index mapping. *IEE Proc. Vis. Image Signal Process.* **1997**, *144*, 31–38. [CrossRef]
24. Hsieh, C.H.; Tsai, J.C. Lossless compression of VQ index with search-order coding. *IEEE Trans. Image Process.* **1996**, *5*, 1579–1582. [CrossRef] [PubMed]
25. Sun, H.M.; Ku, B.H.; Wu, C.C.; Lin, T.Y. Improvement of VQ index compression with relative index tables. In Proceedings of the 2008 International Conference on Computer Science and Information Technology, Singapore, 29 August–2 September 2008; pp. 492–496.
26. Chang, C.C.; Chen, G.M.; Lin, C.C. Lossless Compression Schemes of Vector Quantization Indices Using State Codebook. *J. Softw.* **2009**, *4*, 274–282. [CrossRef]
27. Lee, R.C.T.; Chin, Y.H.; Chang, S.C. Application of principal component analysis to multikey searching. *IEEE Trans. Softw. Eng.* **1976**, *SE-2*, 185–193. [CrossRef]
28. Skodras, A.; Christopoulos, C.; Ebrahimi, T. The JPEG 2000 still image compression standard. *IEEE Signal Process. Mag.* **2001**, *18*, 36–58. [CrossRef]
29. Roelofs, G. *PNG: The Definitive Guide*; O'Reilly & Associates, Inc.: Sebastopol, CA, USA, 1999.
30. Bas, P.; Furon, T. Image Database of BOWS-2. Volume 20. Available online: https://bows2.ec-lille.fr/ (accessed on 20 June 2017).