*Journal of Information Hiding and Multimedia Signal Processing* ©2015 ISSN 2073-4212
**Ubiquitous International** Volume **6**, Number **1**, January **2015**

# Compressing Vector Quantization Index Table Using Side Match State Codebook

Chin-Chen Chang[1,2]

[1]Department of Information Engineering and Computer Science
Feng Chia University, Taichung, Taiwan, 40724, R.O.C.
ccc@cs.ccu.edu.tw

Chia-Chen Lin[3,*]

[3]Department of Computer Science and Information Engineering
Asia University, Taichung, Taiwan, 41354, R.O.C.
*Corresponding author
andrewlin@asia.edu.tw

Chih-Yang Lin[2]
[2]Department of Computer Science and Information Management
AProvidence University, Taichung, Taiwan 433, R. O. C.
[2]Corresponding author
mhlin3@pu.edu.tw@asia.edu.tw

Received November, 2013; revised April, 2014

Abstract. *In the memoryless vector quantization scheme, each image block is independently encoded as a corresponding index and then an index table will be generated. In this paper, we apply the side match concept and propose a new scheme, which can further compress the index table without introducing extra encoding distortion. Our scheme exploits the characteristic that the blocks of images are highly correlated to their neighboring indices in the compression process. To increase the compression, the side match concept is applied to dynamically generate a state codebook for each encoding block. Simulation results show that our schemes are superior to SOC and traditional memoryless VQ on the compression rate.*

**Keywords:** Vector quantization system; SOC; Side match

1. **Introduction.** Vector quantization (VQ) is an effective compression scheme of digital images for the purpose of transmission and storage [3, 12]. The major advantages of VQ are that the compression rate is very high and its hardware of encoder and decoder is very simple. In general, VQ consists of three phases: (1) codebook generation phase, (2) the encoding phase and (3) the decoding phase. At first, a codebook $K$, which is composed of the most representative codewords must be constructed. Then $K$ will be employed in both the encoding phase and the decoding phase. Generating a perfect codebook from a large amount of training set is a critical work in VQ. Many codebook generation algorithms have been proposed, and the most famous one is the LBG algorithm that was presented by Linde, Buzo and Gray in 1980 [9]. Basically, the LBG algorithm is an iterative algorithm that splits the training sets and updates the codebook iteratively. A perfect codebook is generated by a good initial guess and a fast convergence to obtain the most representative codewords. For example, assume that the size of the codebook is set to be 256, and each codeword consists of a 16-dimensions vector.

The image *(S)* first is partitioned into non-overlapping $4x4$ pixel blocks, each block represents one codeword, and then we randomly pick 256 codewords to form the initial codebook. The rest of the blocks are classified into a certain codeword when the Euclidean distance between it and the codeword is the shortest. Lastly, we can get 256 groups. Each group will be recalculated to generate a new centroid, and all new centroids will be treated as a new codebook. These operations are repeated until all centroids are convergent. The codebook generating process is illustrated in Figure 1. In the encoding

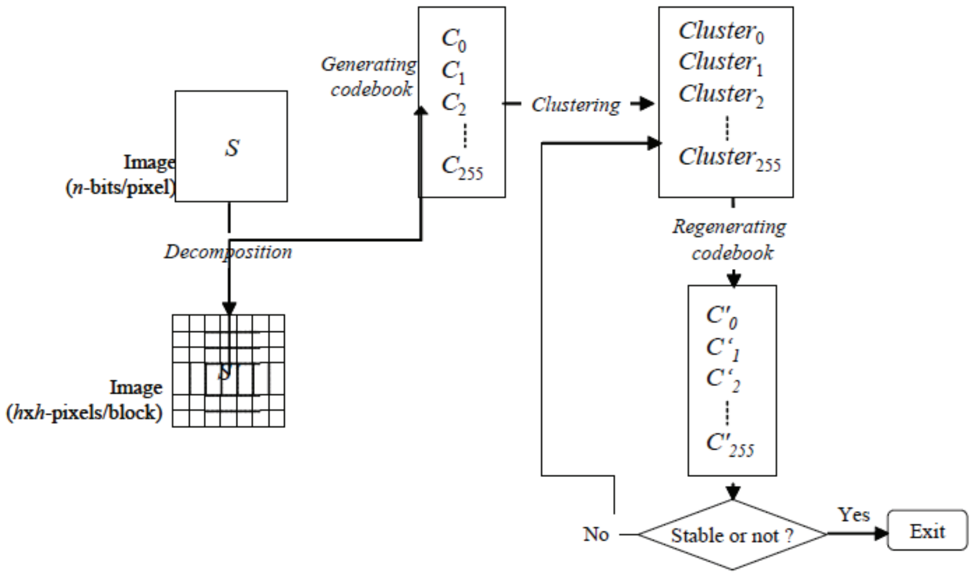


FIGURE 1. The flowchart of VQs codebook generation

phase, the input image $S$ is divided into many non-overlapping square blocks of size $h\times h$ pixels, and each of them stands for an $h^2$-dimension vector. The set of vectors is represented by $S = \{S_1, S_2, ..., S_m\}$, where m is the number of vectors in S. Suppose that the finite set $S = \{k_i \,|i = 0, 1, ..n\}$ is the codebook of size $n$ $k_i = \{k_{i1}, k_{i2}, ...k_{ih2}\}$ is the *ith* codeword. Therefore, VQ can be regarded as a mapping function $Q$ from a $p^2$-dimension Euclidean space $R^{h^2}$ to a finite subset $K \subset R^{h^2}$ $Q : R^{p^2} \to K$ The encoder designs a desired mapping function Q such that the Euclidean distance between the input vector $S_j$ and the mapped codeword $Q(S_j) = k_x$ is the shortest. The Euclidean distance is often used to measure the distortion and defined in Formula (1).

$$\sqrt{\sum_{a=1}^{h\times h}(S_{ja}-k_{xa})^2} = \min_{t=1}^{n}\sqrt{\sum_{a=1}^{h\times h}(S_{ja}-k_{ta})^2}, for 1 \le j \le m, and 1 \le x \le n \tag{1}$$

As a result, the corresponding distortion is minimal. Then, as shown in Figure 2, the index $x$ of codeword $k_x$ which is mapped from the input vector is transmitted to the decoder rather than the input vector $S_j$. Since the input vector is replaced by the index of $k_x$, the number of bits used for representing the index $x$ of the codeword $k_x$ is smaller than that of the input vector $S_j$. The original image is compressed. According to the encoding procedure described above. It is obvious that the *bpp* (bits per pixel) value of the compressed image generated by VQ is 0.5 since the number of bits used for representing the codeword is eight in the above example. In terms of image compression, the good

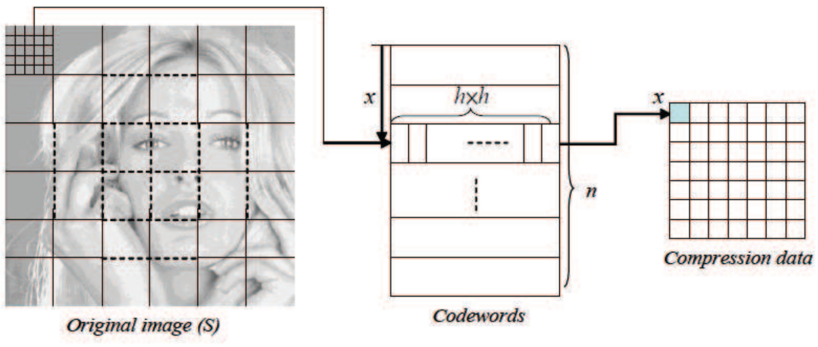


FIGURE 2. The flowchart of VQs codebook generation

compression ratio is provided by VQ.
Before the decoding phase, the decoder and encoder have to share the same codebook. Each index $x$ received by decoder is transmitted by the encoder, and an easy look-up table operation is performed to find the corresponding codeword $k_x$. As a result, the codeword $k_x$ is employed to reconstruct the input vector $S_j$. The original image is reconstructed by repeating the above processes for all the received indices. VQ not only provides good compression ratio but also its computation load is quite low such that it is superior to other compression techniques.
In general, the VQ scheme can be mainly classified into two categories: memoryless VQ and memory VQ. In a memory VQ, image blocks are encoded dependently such as finite-state VQ (FSVQ) [1, 5]. FSVQ employs the previously encoded blocks to make a selection only from the sub-codebook of a much smaller size. Therefore, FSVQ may lead some extra coding distortion. There are several other related techniques such as predictive VQ (PVQ) [2, 4] and address VQ (AVQ) [11]. PVQ can be regarded as an extension of vector quantization. AVQ is based on exploiting the inter-block correlation by encoding a group of blocks together using an address-codebook. However, most of the memory VQ schemes are very complex for hardware implementation and require more computational cost than memoryless VQ. On the contrary, memoryless VQ schemes exploit the high correlation and the spatial redundancy between neighboring pixels; meanwhile, they ignore the spatial redundancy between neighboring image blocks. In other words, each image block is encoded independently. After the encoding phase is completed, a set of indices will be sent to the decoder. Several efficient techniques have been proposed, such as predictive mean search algorithm (PMS) [10], index-compressed VQ (ICVQ) [13], and search-order coding (SOC) [6] and so on. Among these various techniques, it has been proven that SOC is efficient in improving the image compression rate [6]. The main concept of SOC is to search the same index from the neighboring indices of the current processing index and then use search-order codes as the encoded codes. Compression can be achieved when the length of binary representation of search-order codes is shorter than the number of bits required by the original index value. In 2009, Shie and Chen [14] exploited the benefits of combination of SMVQ and SOC to improve the compression rate. Their method does not include VQ indices, which may result in unacceptable image quality. In contrast, our proposed method will consider VQ, SMVQ, and SOC at the same time, which can produce better image quality.

In this paper, we inherit the basic idea of SOC to further propose a new compression scheme that can provide better compression rate than SOC does without losing any image quality from the original VQ encoding. To make sure there are more similar codewords that can be referred for the current processing index, we not only use SOC but also apply the side match concept to dynamically generate a state codebook for each current processing index. Experiments provide supportive results to demonstrate our newly proposed scheme achieves significant reduction of bit rate without losing any image quality by the original VQ encoding.

This paper is organized as follows. Section 2 presents a brief review of some related works, including SOC algorithm [6] and side match VQ [7]. In Section 3, the proposed schemes shall be described in detail. Finally, in Section 4, simulation results shall be given, followed by some conclusions in Section 5.

## 2. **Related Works.**

The search-order coding (SOC) scheme will be described in Subsection 2.1. Since we apply the SMVQs Side-Match and state codebook to design our scheme, the SMVQ will be briefly introduced in Subsection 2.2

### 2.1. **Search-Order Coding Algorithm.**

The search-order coding (SOC) algorithm [6] is a technique that further achieves image compression on the index table of VQ and does not cause coding distortion again. Its basic concept is based on two image characteristics: an image is generally encoded block by block in a raster scan order (i.e. from left to right and from top to bottom) and the neighboring blocks are highly related to the same indices. For an index table, SOC works on the previously generated VQ indices and uses the corresponding search-order codes as the compression codes if any previous index is the same as the current processing index. The binary representation of the search-order codes are less than the original VQ indices, therefore, SOC's compression performance can been achieved.

According to the raster scan order, the current processing index is denoted as a search center (SC) and one of previous encoding indices is the starting search point (SSP). The black dot is the SC and starting search point (SSP) is defined in Figure 1.

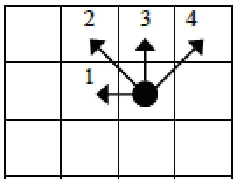


Figure 3. Four starting search points (SSP)

In Figure 3, there are four starting search points (SSP) defined by the SOC algorithm. To start a search, the SSP must be decided in advance. Then, four SPs in the 1st level will be searched for. If any same index is found, the search-order codes will be sent to the decoder. Otherwise, the SPs in the 2nd level will be searched for. If none of the SPs is equal with the current processing index, the original index value of SC will be sent. To distinguish the search-order codes from the original index value by the decoder, an extra bit, called an *indicator*, is also sent to the decoder in the front of them. Following the definition shown in Figure 3, we take Figure 4 for example to describe the idea of SOC. In Figure 4, there are two 7x7 index tables. It is obvious that these two index tables (shown in Figure 4(a) and 2(b)) demonstrate two different search paths for searching a same index value with the SC. The solid boundary squares are the search points (SP) that are previously encoded based on the raster scan order. In other words, non-search points cannot be considered in a search path because they are encoded after SC. It is noted that

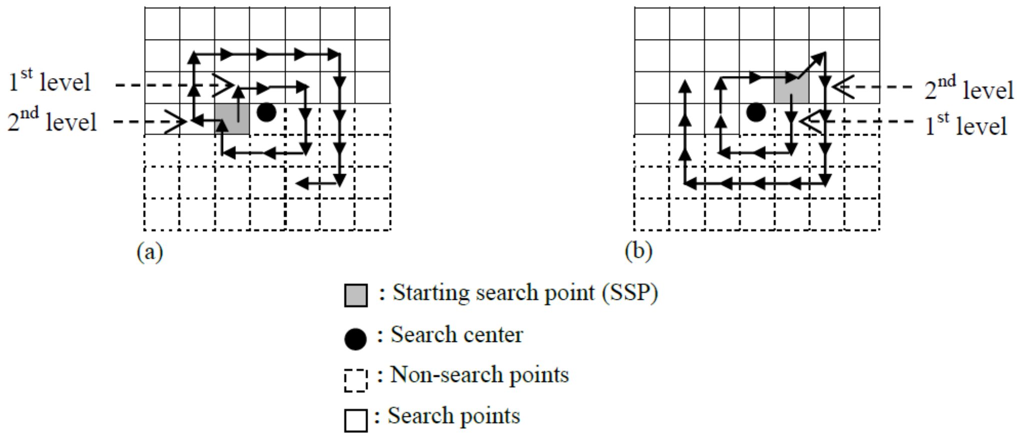


FIGURE 4. Search paths of SOC (a) SSP = 1 (b) SSP = 4

in each search, SOC always searches the SPs across the search path, 1st level, 2nd level, , and so on, until the first same index is found or none of the SPs is matched. For example, in Figure 5, the index of (3,3), which equals to 36, is the SC and SSP equals to 1. In this case, two repetition indices are located in (2,2) and (3,1), which are equal to 28 and 34, respectively. If the number of bits assigned to search-order codes, n, is limited to 2. Once these two repetition indices are included, the matched index of (2,1) that also equals to 36 will not be included in this search. However, the corresponding search-order codes, 11, can be included while the repetition indices are ignored. In the decoding process,

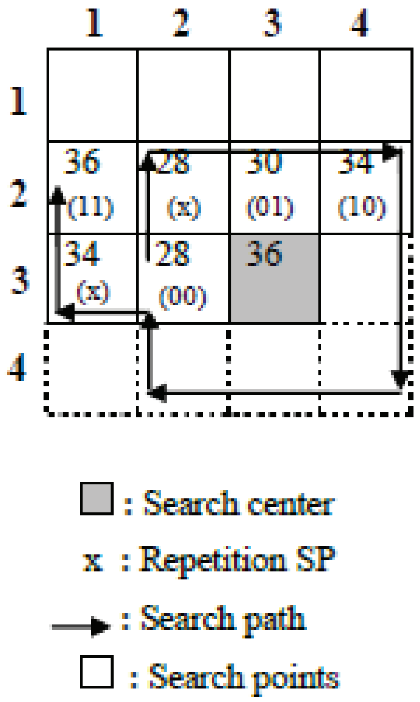


FIGURE 5. A 4  4 index table for showing the exclusion of repetition points

n is equal to that of the encoding process and the raster scan order is also used to be the decoding order. After receiving the compression codes, the decoder can distinguish the search-order codes from the original index value according to its indicator. Then a similar search procedure with the encoding process is conducted. Once the index table is recovered, the image can be reconstruct by using the traditional VQ codebook look-up procedure.

2.2. **The Concept of SMVQ.** To improve the compression bit rate, side match VQ was proposed by Kim in 1992 [7]. Kim assumes the pixels in the top row in the current block are correlated closely with those in the bottom row in the upper block, and the pixels in the first column in the current block are correlated closely with those in the right column in the left block, pixels in the fourth column in the current block are correlated closely with those in the left column in the right block, and that the pixels in the bottom row in the current block are correlated closely with those in the top row in the lower block. Based on this assumption, Kim used Side-Match approach to design SMVQ, and successfully reduces the blocking effect by using local edge information and provides better visual quality and compression ratio than VQ does. To perform SMVQ, a super codebook is required to encode the blocks in the first row and the first column, and a state codebook is required to encode the rest of the blocks. The state codebook is a subset of the super codebook. Consider the 4x4 image blocks shown in Figure 6, where U and L are image blocks reconstructed by the traditional VQ, and X is the current processing image block. The prediction process of SMVQ is as follows. These grey regions in Figure 6 are temporarily assigned by its upper and left neighboring blocks U and L, such that $X_{11} = (U_{41} + L_{14})/2, X_{12} = U_{42}, X_{13} = U_{43}$ , $X_{14} = U_{44}, X_{21} = L_{24}, X_{31} = L_{34}$, and $X_{41} = L_{44}$ , as shown in Figure 6. Then, the assigned the assigned values are used to search against the super codebook for generating a state codebook that contains N closest codewords for the current block X according to the side match distortion SMD.

$$SMD\,(X) = d_U\,(X, U) + d_L\,(X, L) = \sum_{j=1}^{l} (X_{1,j} - U_{4,j})^2 + \sum_{j=1}^{r} (X_{1,i} - L_{j,4})^2 \quad (2)$$

where $d_U$ and $d_L$ are the upper and left side-match distortions, respectively. The codeword of the state codebook with the minimum Euclidean distance from X is used to encode X. Since the state codebook is smaller than the super codebook, the bit rate of the predicted block is reduced from $\lceil \overline{\log 2\ ,\ M} \rceil$ to $\lceil \overline{\log 2\ ,\ N} \rceil$, where M is the size of the super codebook, and N is the size of the state codebook. Moreover, the state codebooks can be generated dynamically from the super codebook, and therefore need not be stored for the reconstruction of the original image.

The flowchart of how SMVQ works is illustrated in Figure 7.

3. **The Proposed Scheme.** In this section, we propose a lossless compression method for VQ indices. The compression scheme is based on SOC and Side-Match approaches by a raster scan order that is from left to right and top to bottom. The VQ image is composed of codewords indices of a universal codebook. Each codeword index represents a subimage block in the image. Since the size of the blocks used in VQ is very small relative to the original images, the neighboring blocks in the image usually have correlations each other. Therefore, the compression bit rate using VQ can be further improved by exploiting the interblock correlations.

In the original SOC method, the indices of the first row and first column are unchanged, but the other indices are required inspection by SOC. In contrast, the proposed method only remains the first index value (i.e., the upper left corner) intact, and the other indices are examined sequentially by two criteria: SOC and Side-Match. When an index is to

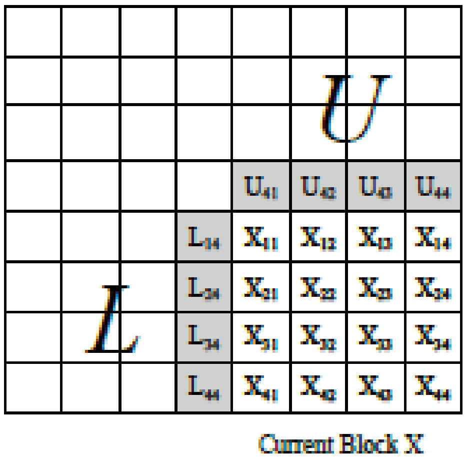


Figure 6. The upper (U) and left (L) blocks used to generate the state codebook and predict the current block X

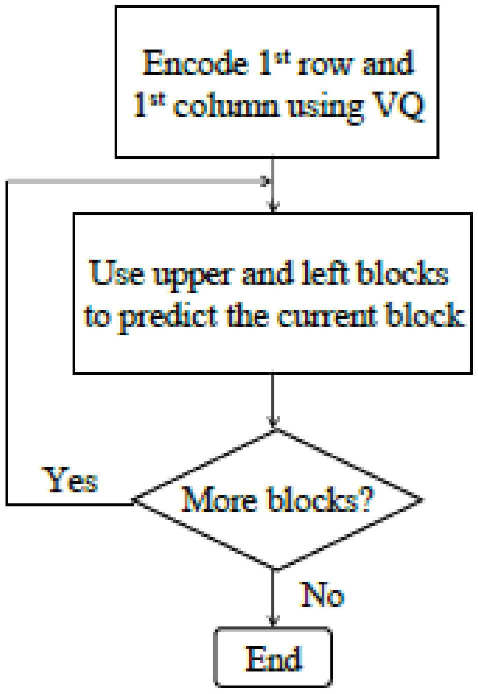
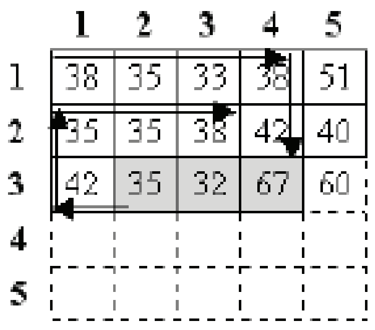

Encode 1st row and 1st column using VQ
Use upper and left blocks to predict the current block
More blocks?
Yes
No
End

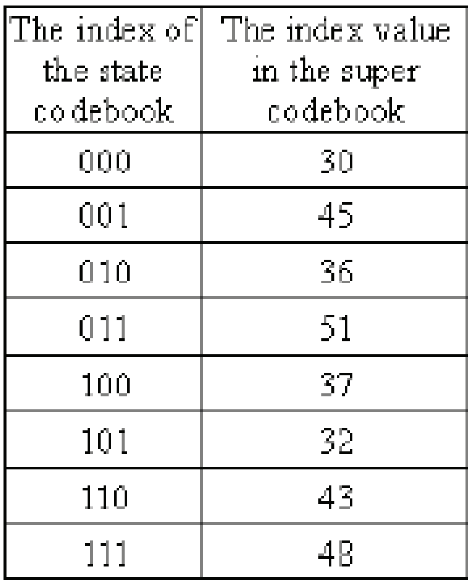
Figure 7. Flowchart of SMVQ

be encoded, it is first checked by SOC. If the neighbors of the current index (i.e., the search points) contain the index value, it is encoded as the search-order compression code (SOCC). Otherwise, the index is inspected further by Side-Match. The Side-Match uses the upper and left adjacent blocks of the current block to construct the state codebook, which has been mentioned in Subsection 2.2. The state codebook should not contain the same index values as the search points used in SOC. If the current index falls in the state codebook, it is represented by the index of the state codebook, which is denoted as a side-match compression code (SMCC). Once the current index cannot be found by using SOC and Side-Match, the index value is still encoded as traditional VQ compression code

(VQCC). It is noted that the indices of the first row and first column in the VQ index table can only refer to one-sided neighboring indices. In the proposed method, there are three types of compression codes in the VQ index table: SOCC, SMCC, and VQCC. In order to distinguish these types, the indicators 0, 10, and 11 are added to the preceding of the SOCC, SMCC, and VQCC, respectively. According to the experimental results in [6], the best choice assigning the number of bits for a SOCC code is two. Furthermore, the indicator for SOCC is 1-bit instead of 2-bits contributes higher compression bit rate in our experiments especially when the smooth image such as Lena is applied. Therefore, if the super codebook size is n and the state codebook size is p, the code lengths for SOCC, SMCC, and VQCC are 1+2, 2+$log_2$p , and 2 +$log_2$ n bits, respectively. We now give an example for the proposed method. Assume the number of nonrepetition search indices (i.e., the search points) used for SOC is 4, the super codebook size is 256, the state codebook size is 8, and the current index to be encoded is at (3,2) shown in Figure 8(a). The current index can be encoded by SOC since its four neighboring nonrepetition indices according to the clockwise search direction are 42, 35, 38, and 33, which the second value is the same as the current index value. Therefore, the index at position (3,2) is encoded as a SOCC code 001, where the first bit "0" is the indicator. On the other hand, the next index 32 at position (3,3) can not be encoded using SOC since its four neighboring nonrepetition indices (i.e., 35, 38, 42, and 33) exclude 32. Hence, the state codebook shown in Figure 8(b) is constructed by Side-Match using the adjacent blocks at positions (2,3) and (3,2). The state codebook at index 101 contains value 32, so the index is encoded as a SMCC code 10101, where the first two bits 10 are the indicator. As to the index 67 at position (3,4), the index can not be encoded by SOC and assume that its state codebook also excludes the index value, the index is encoded as a VQCC code 111000011, where the first two bits "11" denotes the indicator.



(a) The input indices to be encoded



| The index of the state codebook | The index value in the super codebook |
|---|---|
| 000 | 30 |
| 001 | 45 |
| 010 | 36 |
| 011 | 51 |
| 100 | 37 |
| 101 | 32 |
| 110 | 43 |
| 111 | 48 |

(b) The state codebook for position (3,3)

Figure 8. The flowchart of VQs codebook generation

The compression process is summarized as follows:

```
The Proposed Compression Algorithm
Input: The VQ indices that are processed by the raster scan order.
Output: The compression codes of the VQ index table.
Step 1: For an input index, if the same index value can be found using SOC, output the
        corresponding SOCC. Go to Step 4.
Step 2: Construct the state codebook for the input index. The values in the state codebook
        should be different to the search points used in the SOC. If the state codebook contains
        the index value, output the corresponding SMCC. Go to Step 4.
Step 3. If the index value cannot be found by SOC and Side-Match, output the corresponding
        VQCC.
Step 4: Repeat Steps 1 to 3 until all the input indices have been processed.
```

In the decoder, the image blocks can be reconstructed using the same raster scan order as in the compression process. When receiving the compression codes, the decoder uses indicators to distinguish SOCC, SMCC, and VQCC codes. If the decoder receives a VQCC code, the corresponding image block can be reconstructed by the table look-up operation. In contrast, if a SOCC or SMCC code is received, we first recover the original VQ index and then reconstruct the image block. The proposed compression method is lossless and easy to implement.

4. **Experimental Results.** In this section, we conduct several experiments to evaluate the proposed method. Six standard 512512 gray level images "Lena", "F16", "Pepper","Sailboat", "Tiffany", and "Baboon" shown in Figure 9, were used as the test images. The codebooks sized 128, 256, 512, and 1024 are used in the experiments were generated by the LBG algorithm [8]. Each codeword in a codebook has 16 dimensions; that is the size of each subimage block of the test images is 4x4 pixels. For convenient description, we call our method SOCSM (SOC plus Side-Match) in the experiments.

Tables 1-4 show the compression results using different sizes of codebooks. The last row of each table lists the PSNR value of each image. In the tables, VQ denotes the original gray-level images are encoded by a vector quantizer; Huffman indicates that the index table constructed by VQ is further compressed by Huffman coding; SOC sets 2-bits for the search points; SOCSM uses the state codebook with a size of 16. All the methods in these tables are lossless. From these tables we can observe that Huffman coding is no longer suitable for VQ indices compression when the codebook becomes large; furthermore, SOCSM has minimum bit rates for all images. The main reason for its success is that we use two-phase compression scheme instead of one-phase, so the correlations between local blocks can be exploited more thoroughly. Using SOC as the first criterion and then the Side-Match applied can speed up the compression process and lower the bit rates. The "Baboon" image is difficult to lower the bit rate since its context is more complex than the others. When the complex image is applied, the bit rate using SOC may be even worse than using the traditional VQ encoding method.

Tables 5-8 list the bit rates when different sizes of state codebooks are applied. In general, the smaller size of the state codebook is suitable for smaller size of the codebook. In contrast, larger state codebook is appropriate for large codebook. These tables show when the codebook size is 128 or 256, the size 8 of the state codebook is a better choice. However, when the codebook size is up to 512 or 1024, the size the state codebook should be set to 16 to demonstrate the better result.

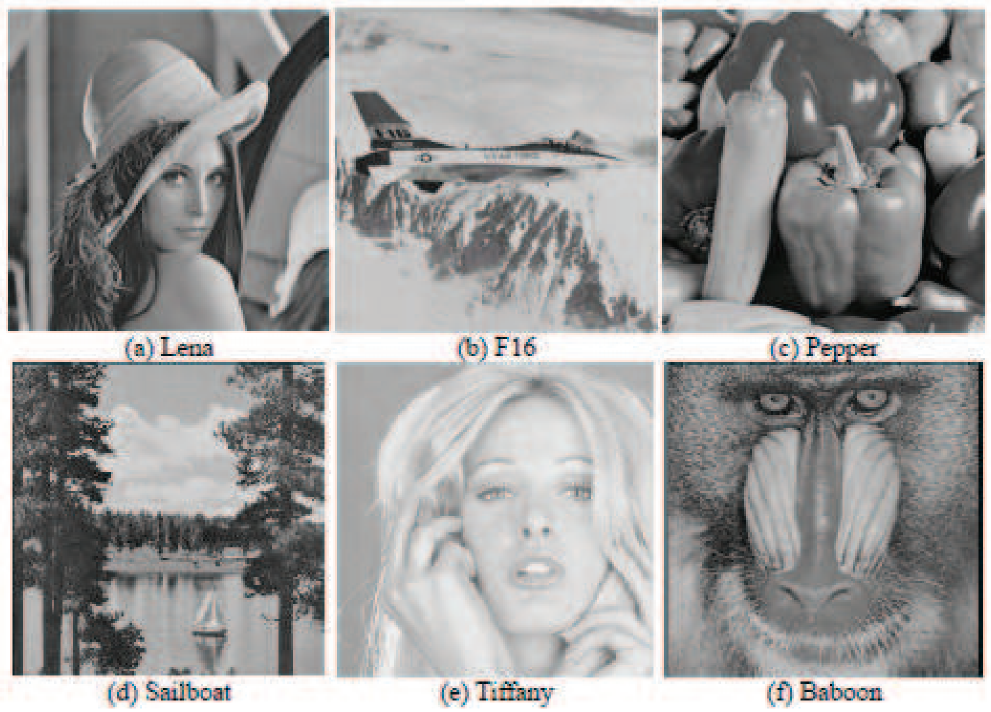


Figure 9. The flowchart of VQs codebook generation

Table 1. Bit rates (bit/pixel) of images using the codebook with size of 128

| Methods | Lena | F16 | Pepper | Sailboat | Tiffany | Baboon |
|---|---|---|---|---|---|---|
| VQ | 0.4375 | 0.4375 | 0.4375 | 0.4375 | 0.4375 | 0.4375 |
| Huffman | 0.306 | 0.273 | 0.287 | 0.285 | 0.247 | 0.391 |
| SOC | 0.277 | 0.27 | 0.277 | 0.291 | 0.212 | 0.399 |
| SOCSM | 0.256 | 0.253 | 0.253 | 0.272 | 0.207 | 0.372 |
| PSNR | 29.349 | 29.308 | 25.522 | 27.43 | 26.052 | 23.935 |

Table 2. Bit rates (bit/pixel) of images using the codebook with size of 256

| Methods | Lena | F16 | Pepper | Sailboat | Tiffany | Baboon |
|---|---|---|---|---|---|---|
| VQ | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| Huffman | 0.439 | 0.356 | 0.417 | 0.409 | 0.256 | 0.474 |
| SOC | 0.359 | 0.325 | 0.352 | 0.367 | 0.281 | 0.492 |
| SOCSM | 0.305 | 0.289 | 0.296 | 0.322 | 0.245 | 0.449 |
| PSNR | 31.366 | 30.582 | 29.734 | 28.622 | 29.248 | 24.379 |

Table 3. Bit rates (bit/pixel) of images using the codebook with size of 512

| Methods | Lena | F16 | Pepper | Sailboat | Tiffany | Baboon |
|---|---|---|---|---|---|---|
| VQ | 0.5625 | 0.5625 | 0.5625 | 0.5625 | 0.5625 | 0.5625 |
| Huffman | 0.548 | 0.491 | 0.529 | 0.524 | 0.371 | 0.569 |
| SOC | 0.454 | 0.433 | 0.442 | 0.46 | 0.347 | 0.575 |
| SOCSM | 0.36 | 0.355 | 0.349 | 0.385 | 0.29 | 0.529 |
| PSNR | 32.24 | 31.58 | 30.47 | 29.25 | 29.57 | 24.71 |

Table 4. Bit rates (bit/pixel) of images using the codebook with size of 1024

| Methods | Lena | F16 | Pepper | Sailboat | Tiffany | Baboon |
|---|---|---|---|---|---|---|
| VQ | 0.625 | 0.625 | 0.625 | 0.625 | 0.625 | 0.625 |
| Huffman | 0.683 | 0.684 | 0.642 | 0.682 | 0.603 | 0.698 |
| SOC | 0.55 | 0.492 | 0.529 | 0.545 | 0.406 | 0.65 |
| SOCSM | 0.434 | 0.391 | 0.408 | 0.456 | 0.323 | 0.611 |
| PSNR | 33.208 | 32.24 | 31.791 | 29.874 | 30.35 | 25.037 |

Table 5. Bit rates (bit/pixel) of images using the codebook with size of 256

| State codebook size | Lena | F16 | Pepper | Sailboat | Tiffany | Baboon |
|---|---|---|---|---|---|---|
| 4 | 0.255 | 0.254 | 0.251 | 0.274 | 0.207 | 0.387 |
| 8 | 0.252 | 0.251 | 0.248 | 0.27 | 0.206 | 0.376 |
| 16 | 0.256 | 0.253 | 0.253 | 0.272 | 0.207 | 0.372 |
| 32 | 0.265 | 0.26 | 0.264 | 0.279 | 0.21 | 0.381 |

Table 6. Bit rates (bit/pixel) of images using the codebook with size of 512

| State codebook size | Lena | F16 | Pepper | Sailboat | Tiffany | Baboon |
|---|---|---|---|---|---|---|
| 4 | 0.312 | 0.297 | 0.303 | 0.334 | 0.246 | 0.477 |
| 8 | 0.303 | 0.29 | 0.293 | 0.325 | 0.242 | 0.46 |
| 16 | 0.305 | 0.289 | 0.296 | 0.322 | 0.245 | 0.449 |
| 32 | 0.316 | 0.295 | 0.31 | 0.329 | 0.254 | 0.449 |

Table 7. Compressible rate using the codebook with size of 256

| Compressible rate | Lena | F16 | Pepper | Sailboat | Tiffany | Baboon |
|---|---|---|---|---|---|---|
| SOC | 0.543 | 0.632 | 0.562 | 0.522 | 0.752 | 0.188 |
| SOCSM | 0.873 | 0.867 | 0.894 | 0.817 | 0.954 | 0.561 |

We define the compressible rate is the number of indices that can be compressed by SOC or SOCSM divided by the total number of indices of a VQ image. From Tables 7 and 8 we can observe that with the help of the Side-Match, SOCSM can compress more VQ indices than SOC does.

5. **Conclusions.** In this paper, a new scheme has been proposed for VQ index compression. Our scheme is designed based on the observation that the high correlation of neighboring blocks in an image. To increase the compression rate, our scheme first uses the search-order coding algorithm to find the same one in the neighboring indices. After that, we apply the Side-Match concept of SMVQ to generate the state codebook for the current processing index and to find the same one from the state codebook. Hence, for generating the compression codes and recovering the original index, the index-based state codebook look-up procedure can be simply performed. Our experiments prove our scheme indeed yields a better performance than other lossless compression techniques, such as SOC, Huffman. Moreover, compared to the traditional memoryless VQ system and the SOC algorithm, both our proposed schemes certainly achieve the goal of reducing the bit rate.

Table 8. Compressible rate using the codebook with size of 512

| Compressible rate | Lena | F16 | Pepper | Sailboat | Tiffany | Baboon |
|---|---|---|---|---|---|---|
| SOC | 0.39 | 0.44 | 0.419 | 0.377 | 0.636 | 0.114 |
| SOCSM | 0.808 | 0.798 | 0.829 | 0.74 | 0.888 | 0.438 |

## References

[1] J. Foster, R. M. Gray, and M. O. Dunham, Finite-state vector quantization for waveform coding, *IEEE Trans. on Information Theory*, vol. 31, no. 1, pp. 348-359, 1985.

[2] R. M. Gary and Y. Linde, Vector quantization and predictive quantizers for Gauss-Markov sources, *IEEE Trans. on Communs.*, vol. COM-30, no. 1, pp. 381-389, 1982.

[3] R. M. Gray, Vector quantization,*IEEE ASSP Mag.*, vol. 1, no. 2, pp. 4-29, 1984.

[4] H. M. Hang and J. M. Woods, Predictive vector quantization of images, *IEEE Trans. on Communs.*, vol. COM-33, no. 11, pp. 1208-1219, 1985.

[5] C. H. Hsieh and J. S. Shue, Frame adaptive finite-state vector quantization for image sequence coding, *Image Commun.*, vol. 7, no. 1, pp. 13-26, 1995.

[6] C. H. Hsieh and J. C. Tsai, Lossless compression of VQ index with search-order coding, *IEEE Trans. on Image Processing*, vol. 5, no. 11, pp. 1579-1582, 1996.

[7] T. Kim, Side match and overlap match vector quantizers for images, *IEEE Trans. on Image Processing*, vol. 1, no. 2, pp. 170-185, 1992.

[8] R. C. T. Lee, Y. H. Chin, and S. C. Chang, Application of principal component analysis to multikey searching, *IEEE Trans. on Software Engineering*, vol. SE-2, no. 3, pp. 185-193, 1976.

[9] Y. Linde, A. Buzo, and R. M. Gray, An algorithm for vector quantizer design, *IEEE Trans. on Communs.*, vol. 28, no. 1, pp. 84-95, 1980.

[10] K. T. Lo and J. Feng, Predictive mean search algorithms for fast VQ encoding of images, *IEEE Trans. on Consumer Electronics*, vol. 41, no. 2, pp. 327-331, 1995.

[11] N. M. Nasrabadi and Y. Feng, Image compression using address-vector quantization," *IEEE Trans. on Communs.*, vol. 38, no. 12, pp. 2166-2173, 990.

[12] N. M. Nasrabadi and R. A. King, Image coding using vector quantization: A review," *IEEE Trans. on Communs.*, vol. 36, no. 8, pp. 957-971, 1988.

[13] J. Shanbehzadeh and P. O. Ogunbona, Index-compressed vector quantization based on index mapping, *Vision, Image and Signal Processing*, vol. 144, no. 1, pp. 31-38, 1997.

[14] S. C. Shie and L. T. Chen, Image compression based on side-match VQ and SOC, *Procceedings of Digital Image Computing: Techniques and Applications*, pp. 369-373, 2009.