IEICE TRANS. INF. & SYST., VOL.E100–D, NO.8 AUGUST 2017

LETTER

Rapid Generation of the State Codebook in Side Match Vector Quantization*

Hanhoon PARK[†a)] ***and*** **Jong-Il PARK**[††b)]**,** ***Members***

SUMMARY Side match vector quantization (SMVQ) has been originally developed for image compression and is also useful for steganography. SMVQ requires to create its own state codebook for each block in both encoding and decoding phases. Since the conventional method for the state codebook generation is extremely time-consuming, this letter proposes a fast generation method. The proposed method is tens times faster than the conventional one without loss of perceptual visual quality.

key words: *side match vector quantization, state codebook generation, codeword clustering, steganography*

1. Introduction

Vector quantization (VQ) is a lossy compression method for images. It uses a given codebook (called *main codebook*) to compress an image by substituting each block of the image with a similar codeword of the same size in the codebook, where the codebook can be trained by the LBG algorithm [6]. However, VQ encodes each block independently and thus tends to cause visible boundaries between neighboring blocks. Side match vector quantization (SMVQ) is an extension of the VQ [5]. It has been designed to enhance the visual quality of VQ by reducing the visible boundaries. It assumes that the correlation between neighboring blocks is statistically high and employs the previous coded blocks to help predict the current block so that the visible boundaries can be reduced. In addition, SMVQ creates a *state codebook* for each encoding block on the fly, where the state codebook consists of less codewords than the main codebook. As a result, the index length in SMVQ is reduced and the compression rate of SMVQ can be higher than that of VQ. To further improve the visual quality and the compression rate, some variants of SMVQ have also been proposed [2], [7].

Recently, it has been proved that VQ and SMVQ can also be used for data hiding (or steganography) [8]. Especially, SMVQ-based steganographic methods have provided a high hiding capacity without degrading the visual quality of cover images [1], [4]. In spite of these advantages and potentials of SMVQ, SMVQ (including the SMVQ-variants) has a drawback that the generation of the state codebook is extremely time-consuming. Thus, the main concern of this letter lies in speeding up the state codebook generation in SMVQ.

Manuscript received February 14, 2017.
Manuscript revised April 18, 2017.
Manuscript publicized May 16, 2017.

[†]The author is with the Department of Electronic Engineering, Pukyong National University, Busan 48513, Korea.
[††]The author is with the Department of Computer Software, Hanyang University, Seoul 04763, Korea.

*This work was supported by the research fund of Signal Intelligence Research Center supervised by Defense Acquisition Program Administration and Agency for Defense Development of Korea.

a) E-mail: hanhoon_park@pknu.ac.kr
b) E-mail: jipark@hanyang.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2017EDL8029

2. Generation of the State Codebook in SMVQ

In SMVQ, the state codebook is generated as follows. All the blocks in the top row and leftmost column of the image are encoded in advance using VQ with the main codebook. Then, SMVQ processes the image blocks in a scanline order. Let $\mathbf{x}$ denote the current block, and $\mathbf{m}_u$ and $\mathbf{m}_l$ be the codewords of the upper and left blocks, respectively. The border vector of $\mathbf{x}$ is defined as

$$\mathbf{v}_x = \Big\{(\mathbf{m}_u^{(b_h-1,0)} + \mathbf{m}_l^{(0,b_w-1)})/2, \mathbf{m}_u^{(b_h-1,1)}, \cdots, \mathbf{m}_u^{(b_h-1,b_w-1)}, \mathbf{m}_l^{(1,b_w-1)}, \cdots, \mathbf{m}_l^{(b_h-1,b_w-1)}\Big\}. \tag{1}$$

Here, b_w and b_h are the width and height of blocks. The side vector of each codeword ($\mathbf{m}_i$, $i = 0, 1, 2, \ldots, w_M$) in the main codebook $\mathbf{M}$ is defined as

$$\mathbf{v}_{m_i} = \Big\{\mathbf{m}_i^{(0,0)}, \mathbf{m}_i^{(0,1)}, \cdots, \mathbf{m}_i^{(0,b_w-1)}, \mathbf{m}_i^{(1,0)}, \mathbf{m}_i^{(2,0)}, \cdots, \mathbf{m}_i^{(b_h-1,0)}\Big\}. \tag{2}$$

Then, the side match distortion is computed between $\mathbf{v}_x$ and $\mathbf{v}_{m_i}$ as

$$d_i = \sqrt{\sum_{k=0}^{b_w+b_h-2} \Big[\mathbf{v}_x(k) - \mathbf{v}_{m_i}(k)\Big]^2}. \tag{3}$$

The state codebook of $\mathbf{x}$ (denoted by $\mathbf{S}_x$) is generated by choosing w_S codewords that have the smallest d_i values from the main codebook. Here, w_S indicates the state codebook size and is preset to the same value ($\leq w_M$) for all blocks.

The state codebook generation for each block is not computationally cheap and it should be repeated across all blocks. In fact, the state codebook generation is extremely time-consuming.

There has been a method for reducing the generation time using principal component analysis (PCA) [3]. In a

preprocessing step, it computes the first principal component direction (denoted by $\mathbf{p}_1$) of $\mathbf{v}_{m_i}$, projects $\mathbf{v}_{m_i}$ onto $\mathbf{p}_1$, and sorts the resulting projection values. For the current block $\mathbf{x}$, $\mathbf{v}_x$ is projected onto $\mathbf{p}_1$ and its nearest one is found among the projection values of $\mathbf{v}_{m_i}$ by applying a binary search. The codewords related to the nearest one and its neighbors forms the state codebook of $\mathbf{x}$. Therefore, the method requires a projection operation and a binary search operation to generate a state codebook and is much fast. However, the method terribly lowers the dimension of the border and side vectors, i.e. $(b_w + b_h - 2)$ to 1, which makes that the state codebook is roughly approximated. Thus, it still suffers from the problem with visible boundaries between neighboring blocks.

3. Proposed Method

In a preprocessing step, we create a codebook (called *super-codebook*) for $\mathbf{v}_{m_i}$ and perform clustering of $\mathbf{m}_i$ using the distances between $\mathbf{v}_{m_i}$ and the codewords in the super-codebook. If $\mathbf{v}_{m_i}$ is closest to the c-th codeword, $\mathbf{m}_i$ is assigned to the c-th cluster. Letting N_c be the number of clusters, N_c is determined by dividing the size of $\mathbf{M}$ (w_M) by the size of $\mathbf{S}$ (w_S). Here, $\mathbf{S}$ denotes one of the state codebooks obtained from each block. For the current block $\mathbf{x}$, a codeword that is nearest to $\mathbf{v}_x$ is found from the super-codebook. Then, the cluster that is related to the found codeword is composed of $\mathbf{m}_i$s whose side vectors are close to $\mathbf{v}_x$, and is designated as the state codebook of $\mathbf{x}$. Since only the nearest of $\mathbf{v}_x$ is found among the reduced number of codewords in the super-codebook, not in the main codebook, the codebook generation can be much faster. In the proposed method, the number of the side vectors in each cluster is varying and thus the encoding performance can be slightly different for different images. This will be shown in Sect. 4.

4. Experimental Results and Discussion

Three methods (conventional SMVQ, one using PCA, and proposed) were run on a PC (CPU: i7-3770 3.4GHz, RAM: 8GB, OS: Windows 10 Pro). b_w and b_h were equally set to 4. In Fig. 1, seven images (except *lena*) with a resolution of 512×512 were used for generating the main codebook, i.e. 114,688 ($128 \times 128 \times 7$) blocks were used for training the main codebook, and the *lena* image was compressed and decompressed. The compression rates, PSNRs of decompressed images, and processing times for each method were computed. The original image has 2,097,152 ($512 \times 512 \times 8$) bits and the compression rate was computed by comparing the number of bits before and after compression as follows.

$$\text{Compression rate } (\%) = (1 - \frac{N'_b}{N_b}) \times 100. \quad (4)$$

Fig. 1 Images used in experiments. From left-top, *airplane*, *baboon*, *boat*, *zelda*, *peppers*, *goldhill*, *fruits*, and *lena*. All the images are 512×512 and gray-scaled.

Table 1 Compression rates, PSNRs of decompressed images, and processing times when using the conventional SMVQ

Size of **S**	Size of **M**: 256			Size of **M**: 512			Size of **M**: 1024		
	Rate (%)	PSNR (dB)	Time (ms)	Rate (%)	PSNR (dB)	Time (ms)	Rate (%)	PSNR (dB)	Time (ms)
8	98.41	24.84	667	98.40	24.30	1492	98.40	23.82	3347
16	98.41	27.19	670	98.40	26.89	1505	98.40	26.06	3273
32	97.64	28.81	679	97.63	28.65	1511	97.63	28.12	3300
64	97.64	29.95	704	97.63	29.84	1521	97.63	29.50	3312
128	96.88	30.34	741	96.86	30.69	1571	96.86	30.67	3358
256				96.86	31.06	1665	96.86	31.36	3440
512							96.09	31.69	3637

Table 2 Compression rates, PSNRs of decompressed images, and processing times when using the PCA-based method

Size of **S**	Size of **M**: 256			Size of **M**: 512			Size of **M**: 1024		
	Rate (%)	PSNR (dB)	Time (ms)	Rate (%)	PSNR (dB)	Time (ms)	Rate (%)	PSNR (dB)	Time (ms)
8	98.41	23.35	9	98.40	23.41	9	98.40	23.01	12
16	98.41	25.65	13	98.40	25.33	14	98.40	24.89	17
32	97.64	27.59	24	97.63	27.07	24	97.63	26.61	25
64	97.64	29.24	45	97.63	28.68	46	97.63	28.06	46
128	96.88	30.19	79	96.86	30.15	86	96.86	29.52	89
256				96.86	30.96	170	96.86	30.89	173
512							96.09	31.57	345

Table 3 Compression rates, PSNRs of decompressed images, and processing times when using the proposed method

Size of **S**	Size of **M**: 256			Size of **M**: 512			Size of **M**: 1024		
	Rate (%)	PSNR (dB)	Time (ms)	Rate (%)	PSNR (dB)	Time (ms)	Rate (%)	PSNR (dB)	Time (ms)
≈ 8	98.33	24.66	21	98.17	24.81	33	98.02	23.61	58
≈ 16	97.70	26.35	22	97.79	26.10	33	97.70	26.11	57
≈ 32	97.64	27.53	26	97.64	27.41	41	97.63	27.58	62
≈ 64	97.38	28.65	39	97.21	28.56	54	96.90	28.58	84
≈ 128	96.88	29.72	69	96.86	29.58	76	96.86	29.50	99
≈ 256				96.46	30.51	132	96.34	30.14	157
≈ 512							96.09	31.19	258

Table 4 Variation of the subcodebook size when using the proposed method

Size of **S**	Size of **M**: 256		Size of **M**: 512		Size of **M**: 1024	
	Min	Max	Min	Max	Min	Max
≈ 8*	2	23	1	21	1	29
≈ 16	7	34	4	33	3	42
≈ 32	22	53	15	56	7	63
≈ 64	46	89	49	89	15	120
≈ 128	109	147	101	158	67	185
≈ 256			245	267	193	325
≈ 512					480	544

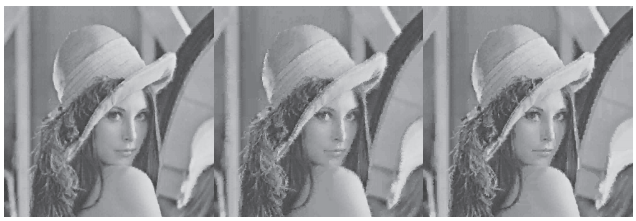
* In the proposed method, the size of **S** is not fixed. As mentioned in Sect. 3, we set the number of clusters in the $\mathbf{v}_{m_i}$'s clustering to a fixed value. Since each cluster usually have different numbers of codewords, the size of **S** can be varying for each block, with its mean value close to what we want (i.e. that of the conventional or PCA-based method). Therefore, "≈ 8" indicates that the size of **S** ranges from 2 and 23 and their mean across all blocks is 8.

Here, N_b and N'_b are the number of bits before and after compression, respectively. The PSNR was computed as

$$\text{PSNR (dB)} = 10 \times \log_{10} \frac{255^2}{\bar{e}}. \quad (5)$$

Here, $\bar{e}$ represents the mean square error between the pixel values of the original image and those of its SMVQ-decompressed image. The processing time indicates the total time taken for compression and decompression.

Tables 1, 2, and 3 show the results. There was no noticeable difference in the compression rates. Since the size of **S** in the conventional method and the PCA-based method are fixed by a given value, the compression rates of both methods were always the same. However, in the proposed method, since the size of **S** was varying according to the cardinality of each cluster as shown in Table 4, the compression rates were different for different conditions or images (sometimes higher but usually slightly lower than those of the other methods). The PSNRs of the PCA-based method and the proposed method were lower than those of the conventional method but the difference was mostly lower than 1dB (less perceptible in the decompressed images, especially with PSNRs higher than 27dB as shown in Fig. 2). In contrast, both methods were much faster (tens or hundreds times) than the conventional method. There was a tendency that the proposed method had higher PSNRs but was slower than the PCA-based method when the difference between the sizes of **M** and **S** was large. However, when the difference between the sizes of **M** and **S** was small, the proposed method had lower PSNRs but was faster than the PCA-based method. This will be because the process of finding the nearest codeword from the super-codebook in the proposed method has a linear time complexity to the super-codebook size ($= w_M/w_S$), while the process of finding the nearest codeword by the vector projection and binary search



(a) When the sizes of **M** and **S** were 256 and 16

(b) When the sizes of **M** and **S** were 512 and 32

(c) When the sizes of **M** and **S** were 1024 and 64

Fig. 2 Decompressed images with different sizes of **M** and **S**. Left: conventional, middle: PCA-based, right: proposed.

Fig. 3 Decompressed images when using the PCA-based method and the proposed method. Left: PCA-based, right: proposed. Images in the bottom row are the enlargement of the delineated regions in the upper images. The sizes of **M** and **S** were 1024 and 16, respectively.

Fig. 4 Decompressed images when applying PCA to the proposed method. Left: the sizes of **M** and **S** were 256 and 16, right: 256 and 64, respectively.

in the PCA-based method has a constant time complexity. Therefore, one may think that both methods can be alternatively used, depending on the point he/she places priority to. However, it should be noted that the PSNR, a quantitative measure, cannot fully describe the difference in the perceptual visual quality of images. In practice, the perceptual visual quality of the proposed method was much better than that of the PCA-based method. This is because the PCA-based method cannot completely resolve the problem with the boundary artifacts (especially, prominent in low PSNRs and edge regions) as explained in the end of Sect. 2 (also see Fig. 3). Therefore, we can conclude that the proposed method is always more useful than the PCA-based method.

To further reduce the state codebook generation time of the proposed method, one can consider to apply PCA to the proposed method. In other words, by computing the first principal component direction of the codewords in the super-codebook and preparing their sorted projections onto the direction vector, a codeword that is nearest to $\mathbf{v}_x$ can be found by binary search. However, in our experiments, a further reduction was not observed. Furthermore, the visual quality of decoded images has become much worse as shown in Fig. 4.

5. Conclusion

In this letter, a fast method for state codebook generation in SMVQ was proposed. It was based on the generation of super-codebook and the clustering of the main codewords. The proposed method outperformed the conventional method and the PCA-based method in terms of the generation speed and the perceptual visual quality of decompressed images.

References

[1] C.-C. Chang, W.-L. Tai, and C.-C. Lin, "A reversible data hiding scheme based on side match vector quantization," IEEE Trans. Circuits Syst. Video Technol., vol.16, no.10, pp.1301–1308, 2006.
[2] R.-F. Chang and W.-M. Chen, "Adaptive edge-based side-match finite-state classified vector quantization with quadtree map," IEEE Trans. Image Process., vol.5, no.2, pp.378–383, 1996.
[3] T.-S. Chen and C.-C. Chang, "A new image coding algorithm using variable-rate side-match finite-state vector quantization," IEEE Trans. Image Process., vol.6, no.8, pp.1185–1187, 1997.
[4] L.S.-T. Chen and J.-C. Lin, "Steganography scheme based on side match vector quantization," Optical Engineering, vol.49, no.3, 037008, 2010.
[5] T. Kim, "Side match and overlap match vector quantizers for images," IEEE Trans. Image Process., vol.1, no.4, pp.170–185, 1992.
[6] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design," IEEE Trans. Commun., vol.28, no.1, pp.84–95, 1980.
[7] X. Ma, Z. Pan, S. Hu, and L. Wang, "Enhanced side match vector quantisation based on constructing complementary state codebook," IET Image Processing, vol.9, no.4, pp.290–299, 2015.
[8] W.-J. Wang, C.-T. Huang, and S.-J. Wang, "VQ applications in steganographic data hiding upon multimedia images," IEEE Syst. J., vol.5, no.4, pp.528–537, 2011.