# LazySOM: Image Compression Using an Enhanced Self-Organizing Map

Cheng-Fa Tsai and Yu-Jiun Lin

Department of Management Information Systems
National Pingtung University of Science and Technology, Pingtung, Taiwan, 91201
{cftsai,m9656013}@mail.npust.edu.tw

**Abstract.** A self-organizing map (SOM), i.e. a congenital clustering algorithm, has a high compression ratio and produces high-quality reconstructed images, making it very suitable for generating image compression codebooks. However, SOMs incur heavy computation particularly when using large numbers of training samples. Thus, to speed up training, this investigation presents an enhanced SOM (named LazySOM) involving a hybrid algorithm combining LBG, SOM and Fast SOM. The proposed algorithm has a low computation cost, enabling the use of SOM with large numbers of training patterns. Simulations are performed to measure two indicators, PSNR and time cost, of the proposed LazySOM.

**Keywords:** Image compression, vector quantization, SOM.

## 1 Introduction

With the increasing popularity of the Internet in recent years, limitations of network bandwidth and storage space have led to image compression issues, the most important being how to generate an appropriate codebook. Vector quantization (VQ) is a well-known image compression method, with a high compression ratio [1]. For instance, an uncompressed image has gray level 8 bits/pixel. Compression using VQ with a vector size of 4×4 and a codebook size of 1024 generates a compressed image with low coding rate 0.625 bits/pixel.

Image compression schemes can be classed as lossy or lossless. VQ is a lossy compression approach [1], [2]. Correlative researchers have developed several vector quantization methods, such as LBG [3] , SOM [4]-[7] and HSOM [8], [11], the most popular being SOM. Since SOM is a full search technique, it can find an impressive codebook for image reconstruction, but incurs a large computation effort, and has a time complexity of $O(n)$. By contrast, LBG is a partitioning method, enabling it to discern an acceptable codebook rapidly. However, random initialization schemes often yield solutions based on local optima. In terms of codebook design, LBG performs less well than SOM, thus producing lower compressed image quality. However, LBG is easy to implement and has fast convergence, making it a common component of hybrid algorithms. Hence, the purpose of this investigation is to accelerate compressing process, and enhance or keep the image quality using a hybrid scheme.

T. Wada, F. Huang, and S. Lin (Eds.): PSIVT 2009, LNCS 5414, pp. 118–129, 2009.

These algorithms are described in the next section. Section 3 illustrates the proposed method. Section 4 depicts the simulation results. Finally, conclusions are drawn in the last section.

## 2 Related Works

This section introduces the basic concepts of VQ, describing the well-known VQ techniques, LBG and SOM, and outlining their merits and drawbacks are outlined. Finally, image quality measures are presented. Fig. 1 illustrates the image compression process in the neural network system.

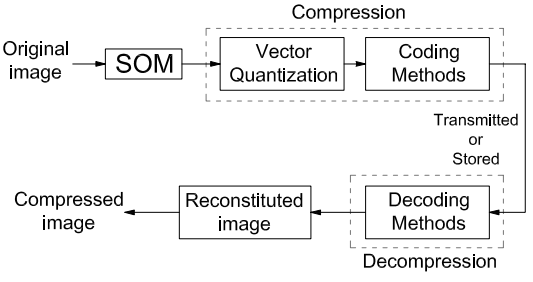


**Fig. 1.** The image compression process in the neural network system

### 2.1 Vector Quantization

Vector Quantization, which is adopted mainly to design codebooks, is a widely-employed lossy compression method that can decrease the compression rate and preserve good quality following compression. Fig. 2 depicts the coding process. An $N \times N$ gray image is first divided into $n \times n$ blocks, forming a block set $V$, which is converted into code-vectors, as $V = v_1, v_2, \ldots, v_{n \times n}$. A codebook $M$ is composed of $k$ code-words, $M = CB_1, CB_2, \ldots, CB_k$, where $CB_k$ indicates the $k$th code-vector in the codebook. A code-vector represents an index-value in the index-book. When transmitting images in the network, only the codebook and the index-book need to be transmitted, rather than the original image, thus lowering the storage space and time.

### 2.2 LBG

Linde Y., Buzo A. and Gray R.M. first developed the LBG algorithm, also called K-means, in 1980. LBG assigns code-vectors in the codebook by continuously comparing the distance between training dataset and centroid of cluster until the variant of average distortion is less than the pre-setting threshold. The simplicity of the LBG algorithm means that it can obtain the codebook efficiently. However, choosing an initial codebook randomly may cause the algorithm to fall into local optimum, possibly making the final result unstable.

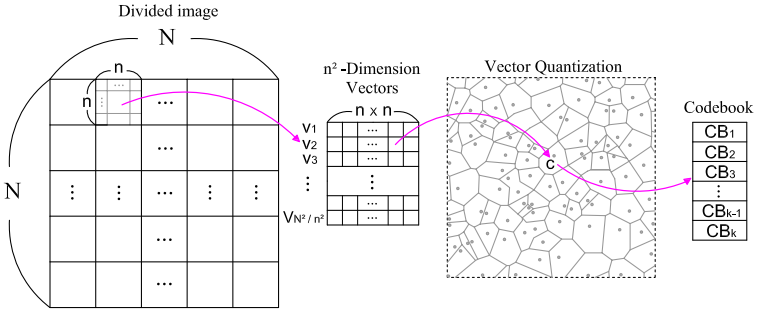


**Fig. 2.** The coding process of VQ

### 2.3 Self-Organizing Map

Kohonen proposed a Self-Organizing Map (SOM) for unsupervised neural networks in 1980. Although SOM is a competitive learning network, it is not based on "winner takes all". SOM utilizes the concept of "neighborhood", in which neurons neighboring the winning neuron are also activated. Fig. 3 displays the operation of "neighborhood" in SOM. In general, a Gaussian Function is performed to implement this concept. SOM usually produces better results than LBG. However, SOM takes a full search to compete with the training data set, so has a fairly high time complexity.

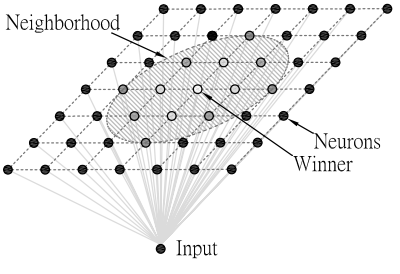


**Fig. 3.** The operation of "neighborhood" in SOM

Many elevated SOMs, such as the standard and advanced Fast SOM systems, try to decrease the large computation effort of SOM [8], [9]. These schemes focus mainly on the initialization of the neural network. Restated, they are mapped in the initial phase, and decrease the length of training time. However, Fast SOM initializes the neural network by K-means. The time complexity of the initialization of Fast SOM is $O(nkT)$, where $k$ represents the number of clusters, and T denotes the epoch of convergence. This approach raises the computation effort, particularly with many training patterns (Notably, Fast SOM was only designed for pattern recognition problem [9]). Therefore, an improved Fast SOM algorithm with an efficient initialization scheme, called Advanced Fast SOM herein,

was presented in 2002. The time complexity of initialization of Advanced Fast SOM is $O(n)$. Although Advanced Fast SOM has more efficient initialization than Fast SOM. It adopt two-dimensional space, making it difficult to apply to high-dimensional data. However, Advanced Fast SOM still works when designing enhanced SOM for improved neural networks initialization.

Another Self-Organizing Map, Hierarchical Self-Organizing Map (HSOM), was presented by Barbalho in 2001. The major advantage of HSOM is that it can reduce computation cost of SOM from $O(n)$ to $O(\log\ n)$. However, it still has a deficiency, i.e. the number of neurons (split sub-maps) of HSOM is fixed. This way makes it easy to trap into local optima. Moreover, the data distribution of HSOM can not be mapped effectively.

### 2.4 The Measurement of Image Quality

Some objective measures are available to verify the quality of a compressed image. For instance mean square error (MSE) and Peak Signal-to-Noise Ratio (PSNR) are typically used, and are formulated as follows.

$$PSNR = 10 \times \log_{10}(\frac{255^2}{MSE}) \tag{1}$$

$$MSE = \frac{1}{M \times N} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\widehat{I}(x,y) - I(x,y)]^2 \tag{2}$$

## 3 The Proposed Algorithm

This work proposes an enhanced SOM that employs virtual patterns to accelerate the training process. The virtual patterns are estimated by LBG. The main feature of the proposed algorithm is that it does not utilize all true patterns to train neurons, but instead only employs by a few virtual patterns. The proposed approach reduces computation effort successfully, and thus is called LazySOM. Although it is named "Lazy", it was still found to be significantly faster than other tested algorithms, in particular the original SOM.

LazySOM has three main steps, called "Initial Phase", "Remapping Phase" and "Fine-tuned Phase". In phase 1, the 2-dimensional neural network was first estimated randomly, and further, virtual patterns were generated by LBG with $\varepsilon$=0.01, where $\varepsilon$ denotes the variant of average distortion. This phase only requires a roughly mapped neural network as the initial solution. Hence, the size of $\varepsilon$ in LBG is not a major issue. Furthermore, conventional SOM and virtual patterns were adopted to train neurons for 100 training epochs. Training was completed briskly due to the small number of virtual patterns. Fig. 4 depicts the training process of 2-dimensional data for SOM, Advanced Fast SOM and LazySOM. Fig. 4(i) shows the initialization of LazySOM, in which the data patterns distribution is mainly mapped into the "Initial Phase". In phase 2, the non-mapped neurons were found from 2-dimensional neural network and

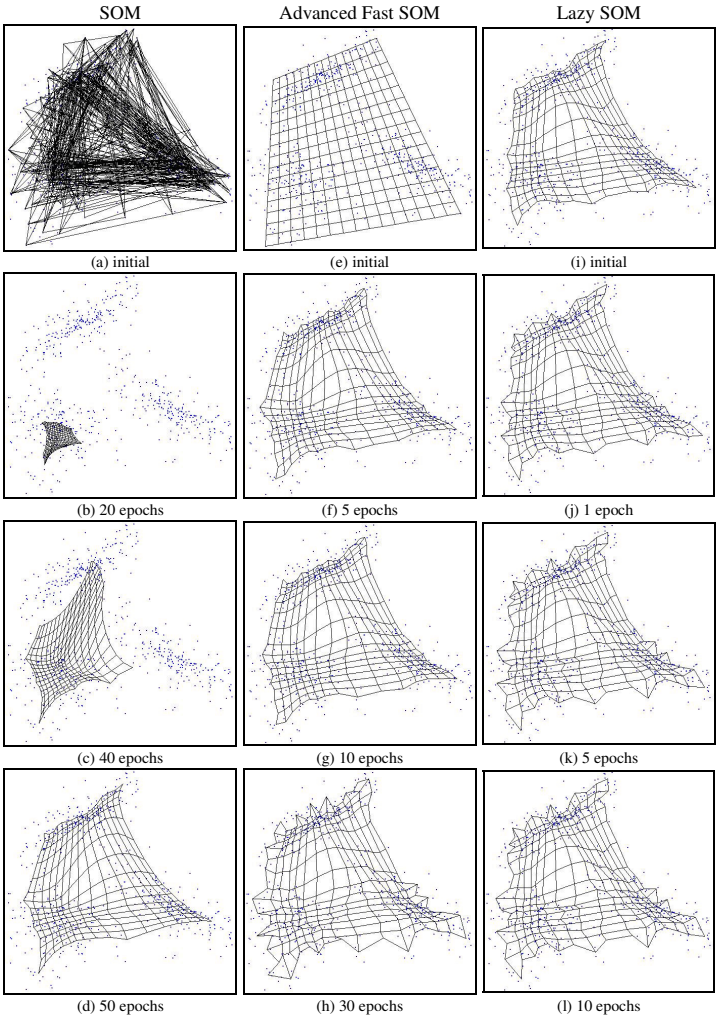


**Fig. 4.** Training process of 2-dimensional data for SOM, Advanced Fast SOM and LazySOM

randomly assigned data vectors to them to avoid dilapidation. Notably, many neurons were wasted while neuron number is large enough in SOM system. Due to neural network approximate data distribution, the final phase, not only trains neurons based on true data patterns, but also only determines the winning solution in fewer training epochs than previous schemes. The final neurons were quickly obtained as the codebook at the end of phase three. Fig. 4(l) presents the final configuration of the neural network (LazySOM) with 10 training epochs.

The procedure of LazySOM algorithm can be described step by step below:

Step 1. Initialize parameters:
$T$: training epoch, $N \times N$: neural array, $k$: number of cluster, $\eta_0$: initial learning rate ($\eta_0 > 0$).

Step 2. Estimate virtual patterns $y_k$ using **LBG** ($\varepsilon$=0.01).

Step 3. Train neural network for 100 training epochs by conventional SOM and virtual patterns.
Set $R = N$ and $\eta = 1$.
Search the winner ($w^*$) by comparing the distance between virtual patterns ($y_k$) and neurons, as in Eqn. (3).

$$w_j^* = \arg\min \|y_k - w_j(t)\|, \quad j = 1, 2, \ldots, N^2 \tag{3}$$

In Eqn. (3), notation $\| \cdot \|$ represents the Euclidean distance.
Weight adaptation: The weights of the neurons neighboring winner are adapted by Eqn. (4).

$$w_j(t+1) = w_j(t) + \eta(t) \times h_{ji}(t) \times [y_k - w_j(t)] \tag{4}$$

$$h_{ji}(t) = \exp(-\frac{r}{R}) \tag{5}$$

Eqn. (5) derives the neighborhood function, where $r :\| j - j^* \|$ and $R$ is the radius of neighborhood.

Step 4. Update parameters $R$ and $\eta$, as follows:
$R = R \times 0.95$
IF $R < 0.1$ THEN $R$=0.1
$\eta$=$\eta \times 0.975$
IF $\eta < 0.01$ THEN $\eta$=0.01

Step 5. Repeat Steps 3-5 until training epoch is achieved.

Step 6. Search non-mapped neurons, and assign data patterns to them randomly.
Set $t$=0, where $t$ indicates the current epoch.

Step 7. Update parameter $\eta$, as follows:

$$\eta(t+1) = \frac{\eta_0}{1+t} \tag{6}$$

Step 8. Search winner ($w_j^*$) by comparing distance between true patterns ($x_i$) and neurons, as in Eqn. (7).

$$w_j^* = \arg\min \|x_i - w_j(t)\|, \quad j = 1, 2, \ldots, N^2 \tag{7}$$

Only update the weight of winner $w_{j^*}$, as in Eqn. (8):

$$w_j^*(t+1) = w_j^*(t) + \eta(t) \times [x_i - w_j^*(t)] \tag{8}$$

Step 9. Repeat Steps 7-8 until all patterns $x_i$ are trained.

Step 10. $t = t + 1$.
IF $t < T$ THEN goto Step7.

The LazySOM algorithm can be illustrated in detail as follows:

```
LazySOM(DataList, N, k, η0, T )
    /*Assign neurons randomly.*/
    NeuralArray ← InitializeNeuralNetwork(N);
    VirtualPatterns ← LBG( k, ε);   /* k: number of virtual patterns.*/
    /* Employ virtual patterns and Kohonen-SOM to train neurons.*/
    SOM(VirtualPatterns, NeuralArray, 1, 0.975, 0.01, N, 0.95, 0.1, 100);
    List ← Search_Non-mapped_Neurons(NeuralArray);
    AssignVectorsRandomly(List);
    t ← 0;   /* t: current training epoch. */
    WHILE t < T DO    /* T: Max. training epoch. */
        η ← η0 / (1 + t);
        t++;
        FOR i ←1 TO K ← DataList.size()    /* K: number of true patterns.*/
            /*Eqn. (7)*/
            Winner ← SearchWinner(DataList[i], NeuralArray);
            /*Utilize Eqn. (8) to update weight of winner.*/
            OnlyUpdateWeightOfWinner(DataList[i], NeuralArray[Winner], η);
        END FOR
    END WHILE
End LazySOM

SOM(DataPatterns, NeuralArray, η, ηrate, ηmin, R0, Rrate, Rmin, Cycles)
    t ← 1;    /* t: Current training epoch.*/
    WHILE t < Cycles DO    /* Cycles: Max. training epoch.*/
        /* k: number of data patterns */
        FOR i ← 0 TO k ← DataPatterns.size()
            /* Eqn.(3) */
            Winner ← SearchWinner(DataPatterns[i], NeuralArray);
            /* Utilize Eqns. (4)-(5) to update weight of neurons of
               neighborhood */
            UpdateWeightOfNeighborhood (DataPatterns[i], NeuralArray,
                                        Winner,R0, η);
        END FOR
        η ← η × ηrate;
        IF η < ηmin
            η ← ηmin;
        END IF
        R0 ← R0 × Rrate;
        IF R0 < Rmin
            R0 ← Rmin;
        END IF
        t + +;
    END WHILE
END SOM
```

## 4 Experiment and Analysis

The experiment comprising quality of compressed images and time cost of the presented LazySOM algorithm were demonstrated. The experiment was con-

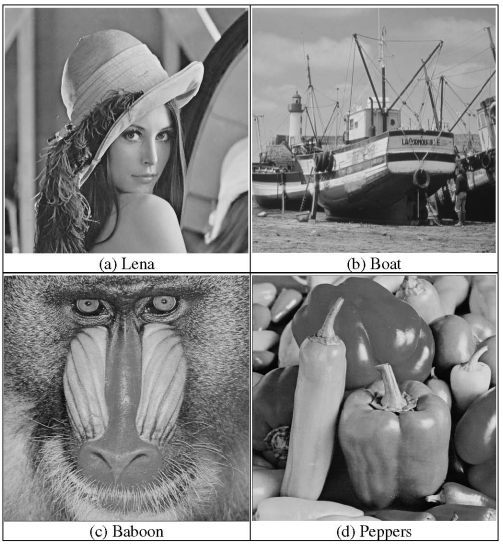


**Fig. 5.** Test images

**Table 1.** The simulation results (PSNR and time cost) for LazySOM, LBG, SOM, HSOM and FastSOMs using four gray Lena, Boat, Baboon and Peppers images with 128, 256, 512, and 1024 test codebook sizes. Notably, the left-hand side of the table indicates the PSNR comparison, while the right-hand side of the table represents the time cost comparison. In addition, Boldface depicts the best one, while N/A denotes not-available.

| Image | CodeBook Size | PSNR (dB) | | | | | Time cost (second) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | LazySOM | LBG | SOM | HSOM | Fast SOMs | LazySOM | LBG | SOM | HSOM | Fast SOMs |
| Lena | 128 | 29.661 | 29.569 | **29.776** | N/A | N/A | **5.570** | 8.394 | 146.235 | N/A | N/A |
| | 256 | 30.680 | 30.468 | **30.782** | 30.636 | N/A | **7.646** | 16.693 | 296.515 | 56.080 | N/A |
| | 512 | **31.843** | 31.272 | 31.567 | N/A | N/A | **14.579** | 28.215 | 600.047 | N/A | N/A |
| | 1024 | **33.171** | 32.106 | 32.428 | 32.973 | N/A | **19.999** | 45.065 | 1,195.875 | 113.786 | N/A |
| Boat | 128 | 29.305 | 29.132 | **29.382** | N/A | N/A | **5.633** | 14.178 | 147.516 | N/A | N/A |
| | 256 | 30.306 | 29.935 | **30.334** | 30.166 | N/A | **7.725** | 21.082 | 297.453 | 55.771 | N/A |
| | 512 | **31.415** | 30.754 | 31.254 | N/A | N/A | **14.607** | 30.458 | 596.156 | N/A | N/A |
| | 1024 | **32.626** | 31.643 | 32.160 | 32.455 | N/A | **19.949** | 42.124 | 1,199.422 | 114.080 | N/A |
| Baboon | 128 | 23.251 | 23.202 | **23.267** | N/A | N/A | **5.363** | 8.855 | 146.812 | N/A | N/A |
| | 256 | 23.906 | 23.828 | **23.948** | 23.738 | N/A | **7.520** | 15.295 | 302.687 | 56.014 | N/A |
| | 512 | 24.635 | 24.494 | **24.656** | N/A | N/A | **14.284** | 24.151 | 595.562 | N/A | N/A |
| | 1024 | **25.466** | 25.247 | 25.363 | 25.242 | N/A | **19.304** | 39.655 | 1,204.813 | 114.195 | N/A |
| Peppers | 128 | 29.788 | 29.674 | **29.858** | N/A | N/A | **5.653** | 9.019 | 149.609 | N/A | N/A |
| | 256 | 30.697 | 30.488 | **30.702** | 30.627 | N/A | **7.741** | 16.013 | 294.953 | 55.847 | N/A |
| | 512 | **31.666** | 31.223 | 31.498 | N/A | N/A | **14.585** | 26.168 | 591.672 | N/A | N/A |
| | 1024 | **32.723** | 31.985 | 32.293 | 32.573 | N/A | **19.927** | 38.181 | 1,194.093 | 114.173 | N/A |

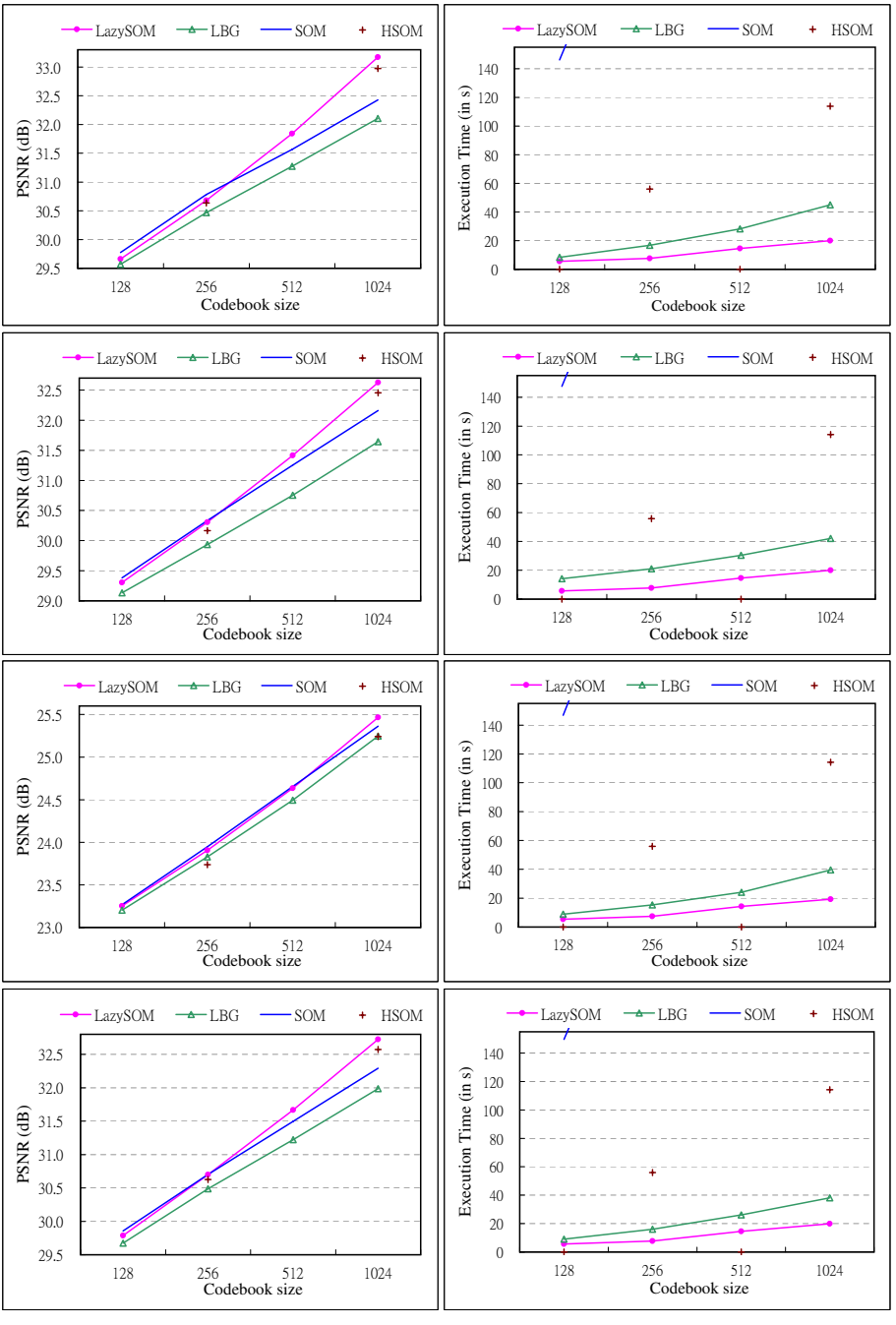


**Fig. 6.** The comparison of PSNR (in dB) and time cost (in second) for LazySOM, LBG, SOM and HSOM using four gray Lena, Boat, Baboon and Peppers images with 128, 256, 512, and 1024 test codebook sizes


**Fig. 7.** The system interface of the LazySOM image compressor, using a compressed image of Lena with a vector = 2×2 and codebook=64

ducted in a Java-based program and ran on a desktop computer with 2GB RAM, an Intel T7300 2.0GHz CPU on Microsoft Windows XP professional operational system. Four gray Lena, Boat, Baboon and Peppers images were employed. Simulation results were calculated with the average of 30 rounds.

For fair comparison, the parameters of HSOM approach were set as in paper [8]. Moreover, the stopping threshold ($\varepsilon$) of LBG was set to 0.0001, while the training epoch was set to 200 (for conventional SOM). For our proposed LazySOM, the number of clusters, namely virtual patterns in codebook size$\leq$512 and codebook size=1024 was set to 64 and 128 respectively. Learning rate in codebook size of 128, 256, 512 and 1024 was set to 0.3, 0.4, 0.5 and 0.7 respectively, while the training epoch was set to 30, 20, 20 and 10 respectively. Notably, Fast SOM and Advanced Fast SOM were only designed for pattern recognition problem, and they are not available for vector quantization. Hence, they were not shown in the experiment for comparison.

Fig. 5 presents the original images involving Lena, Boat, Baboon and Peppers. The test block was 512×512, and the test codebook sizes were 128, 256, 512, and 1024. Moreover, the color level is gray level, i.e. RGB value is equivalent (3-color model). Table 1 summarizes the simulation results (PSNR and time cost) for

**Fig. 8.** The compressed images of Lena, Boat, Baboon and Peppers generated by LazySOM with codebook size 1024 and vector 4×4

LazySOM, LBG, SOM, HSOM and FastSOMs (including Advanced Fast SOM) using four gray Lena, Boat, Baboon and Peppers images with 128, 256, 512, and 1024 test codebook sizes. Notably, the left-hand side of the table indicates the PSNR comparisons, while the right-hand side of the table represents the time cost comparisons. Boldface depicts the best one, and N/A denotes not-available. For HSOM, N/A is caused by codebook setting in HSOM must be $N^2$. Thus, only 256 and 1024 can be set. Moreover, Fast SOM adopts 2-dimensional concept to initialize neural network, making it difficult to apply to high-dimensional data. Notably, the authors (M. C. Su *et al.*) proposed it just for pattern recognition problem (not for VQ).

Fig. 6 demonstrates the comparison of PSNR (in dB) and time cost (in second) for LazySOM, LBG, SOM and HSOM using four gray Lena, Boat, Baboon and Peppers images with 128, 256, 512, and 1024 test codebook sizes. It is observed that the proposed LazySOM can generate compressed image with best quality and has the lowest computation cost. Fig. 7 illustrates the system interface of the LazySOM image compressor, using a compressed image of Lena with a vector = 2×2 and codebook=64. Fig. 8 shows the compressed images of Lena, Boat, Baboon and Peppers generated by LazySOM with codebook size 1024 and vector 4×4. These images reveal that the compressed and original images look very similar to human eyes. These images, which have high quality images and low storage space, would probably be acceptable to most end user. For paper length limitation, there were only several figures and tables to demonstrate the compressed images and time cost.

## 5 Conclusion

LazySOM was found to be the fastest algorithm compared to the tested LBG, SOM and HSOM approaches, since it utilizes few virtual patterns to train the neural network, and only updates the weight of winner. The compressed image quality was slightly lower in LazySOM than in SOM with codebook $\leq 256$. However, the difference was not significant, and LazySOM had a much lower computation effort than SOM. Particularly, training in LazySOM is fewer significantly than SOM. Furthermore, LazySOM with codebook=1024 had the highest PSNR. According to our simulation results, it is observed that the proposed LazySOM can generate a compressed image efficiently and with better quality than several existing well-known SOM related approaches.

**Acknowledgement.** The author would like to thank the National Science Council of Republic of China, Taiwan for financially supporting this research under contract no. NSC 96-2221-E-020-027.

## References

1. Gray, R.M.: Vector Quantization. IEEE ASSP 1(2), 4–29 (1984)
2. Sayood, K.: Introduction to Data Compression, 2nd edn. Morgan Kaufmann, San Francisco (2000)
3. Linde, Y., Buzo, A., Gray, R.M.: An algorithm for vector quantization design. IEEE Trans. Commun. 28, 84–95 (1980)
4. Kohonen, T.: Self-organizing map, Berlin (1995)
5. Kohonen, T.: Self-organizing map. Proceedings of the IEEE 78(9), 1464–1480 (1990)
6. Madeiro, F., Vilar, R.M., Neto, B.G.A.: A Self-Organizing Algorithm for Image Compression. In: Proceedings of Vth Brazilian Symposium on Neural Networks, pp. 146–150 (1998)
7. Kangas, J., Kohonen, T.: Developments and applications of the self-organizing map and related algorithms. Mathemathics and Computers in Simulation 41, 3–12 (1996)
8. Barbalho, M., Duarte, A., Neto, D., Costa, A.F., Netto, L.A.: Hierarchical SOM applied to image compression. In: Proceedings of International Joint Conference on Neural Networks, pp. 442–447 (2001)
9. Su, M.C., Chang, H.T.: Fast self-organizing feature map algorithm. IEEE Trans. on Neural Networks 13(3), 721–733 (2000)
10. Su, M.C., Liu, T.K., Chang, H.T.: Improving the self-organizing feature map algorithm using an efficient initialization scheme. Tamkang Journal of Science and Engineering 5(1), 35–48 (2002)
11. Tsai, C.-F., Jhuang, C.-A., Liu, C.-W.: Gray Image Compression Using New Hierarchical Self-Organizing Map Technique. In: International Conference on Innovative Computing, Information and Control, Paper No. 2858 (2008)