Expert Systems with Applications 39 (2012) 2427–2432

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Speedup of color palette indexing in self-organization of Kohonen feature map

Kuo-Liang Chung [a,1], Yong-Huai Huang [b,*,2], Jyun-Pin Wang [a,b], Ming-Shao Cheng [a,b]

[a] Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei 10672, Taiwan, ROC
[b] Institute of Computer and Communication Engineering, Department of Electronic Engineering, Jinwen University of Science and Technology, No. 99, An-Chung Road, Hsin-Tien Dist., New Taipei City 23154, Taiwan, ROC

ARTICLE INFO

Keywords:
Color palette indexing
Learning process
Lookup table
SOFM
Speedup

ABSTRACT

Based on the self-organization of Kohonen feature map (SOFM), recently, Pei et al. presented an efficient color palette indexing method to construct a color table for compression. Taking the palette indexing method as a representative, this paper presents two new strategies, the pruning-based search strategy and the lookup table (LUT)-based update strategy, to speed up the learning process in the SOFM. Based on four typical testing images, experimental results illustrate that our proposed two strategies have 35% execution-time improvement ratio in average. The practical improvement ratio is very close to that in the theoretical analysis.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

In order to achieve good compression performance by using image compression standards, such as JPEG-LS (Information Technology, 1999; Weinberger, Seroussi, & Sapiro, 2000), JPEG-2000 (JPEG, 2000; Skodras, Christopoulos, & Ebrahimi, 2001), and PNG (Roleof, 2003), how to design a good color palette table is an important issue. In 1981, Kohonen presented a pioneer work, self-organization of Kohonen feature map (SOFM) (Kohonen, 1981), which is a powerful unsupervised neuron learning model. The SOFM has been studied extensively in the applications such as color quantization (Pei & Lo, 1998), color palette indexing design (Pei, Chuang, & Chuang, 2006), vector quantization (Cho, Paiva, Kim, Sanchez, & Príncipe, 2007; Laha, Pal, & Chanda, 2004), clustering (Hsu & Halgamuge, 2008; Hsieh, Jeng, Yang, Chen, & Lin, 2007; Hu, Chen, Hsu, & Tzeng, 2002; Pandya & Macy, 1995; Ritteer, Martinez, & Schulten, 1992; Zhang, Chai, & Yang, 2010), data mining, pattern recognition (Ripley, 1996), graph processing (Hagenbuchner, Sperduti, & Tsoi, 2009), feature selection (Huang & Tsai, 2009), and so on. Recently, Pei et al. presented an efficient SOFM-based color palette indexing method (Pei et al., 2006) for JPEG-LS and JPEG2000. Based on the constructed color palette table, some palette re-indexing methods (Memon, Venkateswaran, & Stanco, 1996; Pinho & Neves, 2004; Zeng, Li, & Lei, 2001) have been developed in order to improve the compression performance further.

In this paper, we take Pei et al.'s color palette indexing method (Pei et al., 2006) as the representative to point out two computational bottlenecks existed in the SOFM. In order to relax the two bottlenecks, for each training vector, we first present a pruning-based search strategy to speed up the process for finding the winning neuron in each iteration; further, a lookup table (LUT) strategy is presented to speed up the lateral update interaction between the winning neuron and its neighboring neurons. Based on four typical testing images, experimental results illustrate that our proposed two strategies have 35% execution-time improvement ratio in average while preserving the same result as in the SOFM. The practical improvement ratio is very close to that in the theoretical analysis. Precisely speaking, the first strategy has 19% execution-time improvement ratio; the second strategy has 16% execution-time improvement ratio. In fact, our proposed two strategies could be used to speed up the other SOFM-based learning processes in different applications.

The rest of this paper is organized as follows: In Section 2, the two computational bottlenecks existed in the learning process of Pei et al.'s method is introduced. In Section 3, our proposed faster learning process is presented. In Section 4, some experimental results are demonstrated. Some concluding remarks are addressed in Section 5.

2. The palette indexing method by Pei et al. and two computational bottlenecks

In this section, we first introduce the palette indexing method by Pei et al. (2006), and then we point out two computational bottlenecks in the SOFM-based learning model. A full color image usually has 24 bits, each color channel with 8 bits. The goal of color palette indexing method is to construct a color palette table such

* Corresponding author.
E-mail addresses: k.l.chung@mail.ntust.edu.tw (K.-L. Chung), yonghuai@ms28.hinet.net (Y.-H. Huang).

[1] Supported by the National Science Council of ROC under contract NSC99-2221-E011-078-MY3.
[2] Supported by the National Science Council of ROC under contract 99-2221-E-228-006.

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.08.092

that the input color image can be converted to an indexed image. In essence, the indexed image associated with the palette table constitutes the quantized color image. Pei et al.'s method has been shown to have good compression performance.

2.1. SOFM-based learning model

As shown in Fig. 1, the 1-D SOFM used in Pei et al.'s method has two layers, namely the input layer and the output layer. The output layer has N neurons, u_1, u_2, ..., and u_N; the initial triple weight of the ith neuron u_i in the neural network, $1 \leqslant i \leqslant N$, is set to

$$\begin{aligned}\mu_i &= [r_i(0), g_i(0), b_i(0)]^T \\ &= [(i-1)*256/N, (i-1)*256/N, (i-1)*256/N]^T, \end{aligned} \quad (1)$$

where $r_i(0)$, $g_i(0)$, and $b_i(0)$ indicate the red value, green value, and blue value at the 0th iteration, i.e. at the initial iteration in the ith neuron; '256' indicates the maximal allowable number of indices in the palette. From the input layer, we feed each pixel $x = [r_x, g_x, b_x]^T$ in the input color image as the training vector into the SOFM for training a good color palette table.

In the training process of the SOFM, M training vectors are required in each sweep. In order to maximize the randomness of the input training vectors, avoid biased training, and avoid some clusters being overtrained during the training process, Pei et al. presented an effective butterfly-jumping sequence to generate the input training vectors for each training sweep. Based on the butterfly-jumping sequence, all pixels in the $W \times H$ input image is separated into $\frac{W \times H}{M}$ training sets S_1, S_2, ..., and $S_{\frac{W \times H}{M}}$. Each set contains M training vector to be used in a training sweep.

For the mth training sweep, $1 \leqslant m \leqslant \frac{W \times H}{M}$, each training vector x in S_m is fed into the SOFM to search the best matched neuron u_c, $1 \leqslant c \leqslant N$, i.e. the winning neuron, in the output layer based on the following square Euclidean distance function:

$$\begin{aligned} c &= \arg\min_{1 \leqslant i \leqslant N} \|x - \mu_i\|^2 \\ &= \arg\min_{1 \leqslant i \leqslant N} [(r_x - r_i)^2 + (g_x - g_i)^2 + (b_x - b_i)^2]. \end{aligned} \quad (2)$$

Based on the input vector x and the winning neuron u_c with weight $\mu_c = [r_c, g_c, b_c]^T$, the output layer therefore updates the weight of the winning neuron and simultaneously performs the lateral update interaction between the winning neuron u_c and its neighboring neurons by the following learning function:

$$\mu_i(m, n+1) = \begin{cases} \mu_i(m,n) + \alpha(m) \cdot g(i,c,\sigma(m)) \cdot [x - \mu_i(m,n)], \\ \quad \text{if } |i-c| \leqslant \lfloor \sigma(m) \rfloor, \\ \mu_i(m,n), \qquad \text{otherwise}, \end{cases} \quad (3)$$

where m and n denotes the sweep number and the iteration number, respectively; $g(i,c,\sigma(m))$ is the neighboring function providing the lateral interaction between u_i and u_c; $\sigma(m)$ and $\alpha(m)$ denote the width of neighboring function and scalar gain function used in the mth sweep, respectively. $g(i,c,\sigma(m))$, $\sigma(m)$, and $\alpha(m)$ are defined by

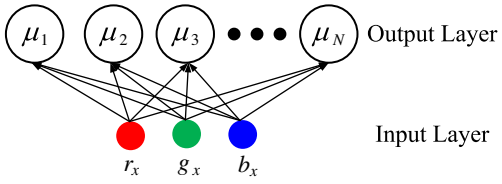

Fig. 1. The used 1-D SOFM.

$$g(i,c,\sigma(m)) = \exp(-|i-c|^2/\sigma^2(m)), \quad (4)$$

$$\sigma(m) = \sigma(0) \times k_1^m, \quad \text{and} \quad (5)$$

$$\alpha(m) = \alpha(0) \times k_2^m, \quad (6)$$

respectively, where k_1 and k_2 are set to values in [0.8, 0.99]. By Eqs. (5) and (6), $\sigma(m)$ and $\alpha(m)$ are decreasing functions in terms of sweep number m. $\sigma(0)$ is set to a value in [1, 10] and $\alpha(0)$ is set to a value less than one initially. In our implementation, we set $\sigma(0) = 10$, $\alpha(0) = 0.1$, $k_1 = 0.8$, and $k_2 = 0.8$ initially.

After updating the weights of neurons in the output layer, the next training vector in S_m will be fed into the SOFM to search the best matched neuron by Eq. (2) and update the weights of relevant neurons by Eq. (3). The above training process is performed iteratively until $\alpha(m)$ becomes small enough. The stopping rule of the learning process is set to that if the condition $\alpha(m) \leqslant 4.05648 \times 10^{-5}$ is held in the mth sweep. From the stopping rule, we find that the number of sweeps N_s can be set to 35 since $\alpha(35)$ satisfies the stopping rule.

2.2. Two computational bottlenecks

From the description in last subsection, we now point out two concerned computational bottlenecks. The first computational bottleneck occurs in the process for searching the winning neuron. In each sweep, we have M training vectors and for each training vector, Eq. (2) is performed to calculate N square Euclidean distances to select the minimal one, and then determine the winning neuron. In Eq. (2), it needs three subtractions, two additions, and three square operations for calculating each square Euclidean distance and from the N calculated Euclidean distances, $N-1$ comparisons are required to find the winning neuron with minimal Euclidean distance. Overall, each sweep needs to calculate Eq. (2) M times. Let T_{add}, T_{sub}, T_{sq}, and T_{cmp} denote the time required to perform one addition, one subtraction, one square operation, and one comparison, respectively. From the state of the art in VLSI technology, one addition, one subtraction, and one square operation can be performed using almost the same time. Therefore, we can use T_{add} to represent each one of T_{sub} and T_{sq} due to $T_{add} \cong T_{sub} \cong T_{sq}$. Further, one comparison is composed of one subtraction and one sign test. Thus, we assume $T_{cmp} = 2T_{add}$ for convenience. By setting the number of sweeps performed in the SOFM to be 35, i.e. $N_s = 35$, we have the following proposition.

Proposition 1. *For each training vector, finding the winning neuron takes $T_W = (10N-2) \times T_{add}$ ($=N \times (3T_{sub} + 2T_{add} + 3T_{sq}) + (N-1) \times T_{cmp}$) time. For each sweep, it takes $M \times T_W$ time to find M winning neurons. For an input color image, it takes $T_W^I = N_s \times M \times T_W$ time to find all winning neurons. For the case $N_s = 35$, we have $T_W^I = (350MN - 70M) \times T_{add}$.*

Proposition 1 indicates the first computational bottleneck in the SOFM-based palette indexing method. Therefore, reducing the computational effort required in finding winning neurons is an important issue and it leads to the first motivation of our research. In Section 3.1, an efficient pruning-based search strategy will be presented to speed up the process for finding the winning neuron in each iteration.

The second computational bottleneck occurs in the process for updating the weights of winning neuron and its neighbors. For each determined winning neuron, $(2\lfloor \sigma(m) \rfloor + 1)$ neurons must be updated by Eq. (3) where $\lfloor \cdot \rfloor$ denotes the floor operation. From Eq. (3), it needs three additions and four multiplications to sum up the four terms $\mu_i(m,n)$, $[x - \mu_i(m,n)]$, $g(i,c,\sigma(m))$, and $\alpha(m)$; the term $[x - \mu_i(m,n)]$ needs three subtractions; the neighboring function $g(i,c,\sigma(m))$ defined in Eq. (4) needs one subtraction, one

Download English Version:

https://daneshyari.com/en/article/10322490

Download Persian Version:

https://daneshyari.com/article/10322490

Daneshyari.com