Fractal Based VQ Image Compression Algorithm

Chuanfeng Lu and Qiangfu Zhao
The University of Aizu
Aizuwakamatsu, Japan 965-8580
Email: {m5061125,qf-zhao}@u-aizu.ac.jp

***Abstract*— In this paper we study VQ (vector quantization) based image compression. So far many methods have been proposed in the literature. Examples include LBG algorithm, self-organizing feature map, and so on. Although VQ based image compression is theoretically useful, it is still not practical mainly for two reasons. The first one is that the computation amount is too big, and the second is that the compression ratio will be very low if the code size is too small or too big. In this paper, we try to solve the first problem by accelerating the winner-take-all algorithm. The basic idea is to insert a new procedure in each learning cycle to reduce the approximation error. We also propose a method to solve the second problem by combining VQ and the IFS (iterative function system). The idea is to reduce the codebook size by encoding each code further using one or a few IFS codes. By so doing we can increase the compression ratio without increasing the error very much. The efficiency of the proposed methods will be shown through experiments with several well known images**

I. Introduction

Vector quantization (VQ) emerged as a powerful approach for image compression, after LBG (Linde, Buzo and Gray) algorithm was introduced. Kohonen's self-organizing feature map (SOFM) can also be used to design the codebook for vector quantization, which is known to have comparable performance to LBG algorithm. In this paper a new compression algorithm is proposed in order to improve the VQ algorithm. The basic idea is re-compress the codebook using the iterated function system (IFS) algorithm. We also propose a new self-organizing learning algorithm which can converge much more faster than the conventional one.

II. Review of VQ and Fractal Algorithm

Vector quantization (VQ) is a generalization of scalar quantization. For image code, adjacent data items in a image are correlated. There is a good chance that the near neighbors of a pixel P will have the same gray level as P or very similar. This can be expanded to block of pixels. Thus there is an opportunity to approximate many similar separate blocks with one $vector$. However there exist a so-called trade off relationship between compression ratio and compression quality. The higher the reconstruction quality is, the lower compression ratio will be, and vice versa.

Unlike VQ, Fractal algorithm is a relative new method in image compression. Fractal have been popular since the 1970s and have many applications. Applying fractals to image compression is done by means of iterated function system (IFS). IFS compression can be very efficient, achieving excellent compression factors. The IFS encoder partitions the image into parts called ranges, then match each range to some other part called a domain to find the most similar one, and produces affine transformation parameters. The compressed file maintains a sequence of these parameters instead of the small image blocks.

III. IFS based VQ

Although there are many references in the literature but the VQ algorithm still can not be used prevalently for image compression, since the initialization and storage space for codebook are two hard problems to be resolved. It is well known that the compressed stream after VQ compression consists of two parts: codebook and index. In order to improve the compression ratio further we need to reduce codebook or index size. However it is very difficult to re-encode index any more. Alternatively we can reduce the codebook. Codebook is a set of vectors, i.e. many small image blocks. The problem is which approach is appropriate for block image coding. IFS brings to mind. IFS encoder can denote blocks with affine transformation coefficients, the larger the block is, the higher compression ratio we can obtain. Strictly speaking this is a re-compression approach, and the distortion will be worse. Instead, the encoding performance of VQ based IFS should be improved.

Though a LBG is simple and fast algorithms, in order to study and exploit the neural networks application in image compression area, the Kohonen neural network (KNN) with self-organized (SO) learning is employed to construct the codebooks. To build a N member M dimensional codebook, the KNN required has M inputs and N outputs. Since the learning process of SO is very slow, a concept $centroid$ from LBG algorithm is used to accelerate the learning process. The learning process is as follows:

1) Initialize the weights (codebook) at random
2) Compare the training examples with each code-vector using Euclidean distance measure, and identify the closest one which is often called the $winner$.
3) A descending learning rule is employed to update the $winner$.
4) After each training iteration, focus on every code-vector within the current codebook, find all the neighbors in the

sample space. Calculate the $centroid$ of these neighbors, and replace the corresponding code-vector.

5) Go on next training iteration.

We selected two well known 256 gray level image: Lenna and Mandrill through out this experiment, the size is 512×512. The performance after ten learning cycles is shown in Table II, which is almost the same as the result obtained by the LBG algorithm (Table I). In each learning cycle, the neurons are moved to the centers of different clusters, as a result the neural network can learn much faster than conventional SO algorithm.

After VQ, an IFS algorithm is used to re-compress the codebook, in this experiment we select Fishers' Quatree method. Since it is a bench mark for many papers in this area, can represent the generality in some extend. The encoding process is shown below:

1) Set a small value to the tolerance, and the minimum partition size.
2) Correspond to the codebook construct a domain pool from original image.
3) For each code-vector, find the most similar domain from the domain pool. If the error rate is smaller then the tolerance output the affine parameters; else if current partition size is big than the minimum, divide it into 4 equal partitions for each partition repeat this step, until the tolerance is satisfied or reach the minimum partition.

The decoding process is a simple two-step process. First, reconstruct the codebook from affine parameters. Then, base upon the index sequence draw back the image. The final result is shown in table III. Fig 2,3 are used to show the reconstructed quality. To obtain a high reconstructed quality, we use a very small tolerance which cause that all partition reached the minimum size. By doing so the quality dose not change very much, but the compression ratio is improved, the increment gain is the largest for the case of 16×16, the reason is when vector size change to large, the index size becomes very small almost be considered as zero.

IV. Conclusion and Remarks

In this paper a hybrid image compression algorithm has been proposed, the originality is to re-encode the codebook using the IFS algorithm. Experimental results have shown that this method can increase the compression ratio without increasing the distortion too much. A new learning algorithm for KNN has also been proposed to improve the learning speed.

References

[1] Y. Linde, A. Buzo and R. M. Gray, "An Algorithm for Vector Quantization", IEEE Trans. Communications, Vol. 28, No. 1, 1980, pp. 84-95.

[2] Self-Organization and Associated Memory (context) - Kohonen - 1988.

[3] Fractal Image Compression: Theory and Application, Yuval Fisher (ed.), Springer Verlag, New York, 1995.

TABLE I

LBG compression performance

		128	256	512	1024
Lenna	4×4	31.0/18.2	32.0/12.7	33.2/9.8	34.4/7.1
	8×8	27.8/23.2	28.9/12.7	30.5/7.0	32.9/3.7
	16×16	25.6/7.8	28.0/3.9	31.3/1.9	35.8/0.95
Mandrill	4×4	23.5/18.2	24.1/12.7	24.8/9.8	25.6/7.1
	8×8	21.1/23.2	21.6/12.7	22.2/7.0	23.3/3.7
	16×16	20.3/7.8	21.2/3.9	23.2/1.9	26.6/0.95

TABLE II

Variation KNN performance after ten times learning

		128	256	512	1024
Lenna	4×4	31.0/18.2	32.0/12.7	33.2/9.8	34.4/7.1
	8×8	27.8/23.2	28.9/12.7	30.5/7.0	32.9/3.7
	16×16	25.6/7.8	28.0/3.9	31.3/1.9	35.8/0.95
Mandrill	4×4	23.5/18.2	24.1/12.7	24.8/9.8	25.6/7.1
	8×8	21.1/23.2	21.6/12.7	22.2/7.0	23.3/3.7
	16×16	20.3/7.8	21.2/3.9	23.2/1.9	26.6/0.95

TABLE III

VQ + IFS compression performance

		128	256	512	1024
Lenna	4×4	29.65/20.47	32.21/15.06	33.20/12.79	34.37/10.66
	8×8	27.53/51.15	28.68/31.98	30.57/20.47	32.84/12.19
	16×16	25.42/29.24	27.99/15.05	31.35/7.27	35.65/3.92
Mandrill	4×4	23.74/20.47	24.02/15.06	24.85/12.79	25.53/10.66
	8×8	21.04/51.15	21.57/31.98	22.16/20.47	23.14/12.19
	16×16	20.63/29.24	21.25/15.05	23.09/7.27	26.39/3.92

Fig. 1. 8×8 codebook:128 PSNR 27.53 ratio:51.15

Fig. 2. 8×8 codebook:128 PSNR 21.04 ratio:51.15