Proc. of the IASTED International Conference ARTIFICIAL
INTELLIGENCE AND SOFT COMPUTING, July 27-31, 1997,
Banff, Canada pp. 248-251

Self-Organizing Neural Network Domain Classification for Fractal Image Coding

Stephen Welstead
COLSA Corporation, 6726 Odyssey Drive, Huntsville, AL 35806, USA
email: swelstead@compuserve.com

Key Words: Neural Networks; Image Compression; Unsupervised Learning; Fractals

Abstract: This paper presents a scheme for improving encoding times for fractal image compression. The approach combines feature extraction with domain classification using a self-organizing neural network. Feature extraction reduces the dimensionality of the problem and enables the neural network to be trained on an image separate from the test image. The self-organizing network introduces a neighborhood topology for classification, and also eliminates the need to specify *a priori* a set of appropriate image classes. The network organizes itself according to the distribution of the image features observed during training. The paper presents results showing that this classification approach can reduce encoding times by two orders of magnitude, while maintaining comparable accuracy and compression performance.

1. Introduction

Fractal image coding presents a promising approach for image compression applications such as Internet Web sites [1]. However, the time requirement for the encoding stage of this approach has hindered its acceptance as a practical method. The encoding process maps domain subimage cells onto range subimage cells. For each range cell, the algorithm searches for the domain cell and corresponding transformation that provides the best fit to the range cell. Classification of domains can significantly speed up encoding performance by reducing the number of domains which must be searched. Most fractal image compression implementations incorporate some type of domain classification [2,3]. The use of self-organizing
neural networks to provide domain classification has also previously been explored [4,5]. These networks improve upon basic domain classification schemes by defining a topology for neighborhoods of classes. The contribution of the present paper is to combine self-organizing neural network classification of domains with feature extraction to provide even faster image encoding. A small set of image features measuring tone and texture qualities are computed for each domain and range cell. Each cell thus has an associated feature vector whose dimension is independent of cell size. Feature extraction is advantageous in two ways. The resulting lowering in dimensionality of the problem reduces the computations needed during domain search. Second, because the features are independent of the structure of particular domains, it is possible to train the self-organizing network on one image and apply the resulting network to classification on a different image. Thus, network training time is not part of the overall encoding time. Fractal image encoding times are usually reported in terms of performance on a Sun or Silicon Graphics workstation. The approach presented here provides comparable performance times while running on a 120 MHz Pentium PC.

2. Fractal Image Coding

Fractal image coding is based on the theory of iterated function systems (IFS) [6]. IFS theory has at its core the contraction mapping theorem (CMT) of classical analysis, and it has been used to iteratively produce fractal images. The fractal image is the fixed point in image space that is guaranteed by the CMT, and this image is called the attractor of the IFS. The inverse problem solved by fractal image coding is to start with a given image and compute an IFS that has an image close to the given image as its attractor. The fractal image code usually (though not always) requires less storage space than the original image, and so this represents a compression technique. Empirical results show that in many cases the fractal scheme is as good as JPEG, considered the current compression standard [2].

Fractal image compression uses a special type of IFS called a partitioned iterated function system (PIFS). A PIFS consists of a complete metric space $\mathbf{X}$, a collection of sub-domains $D_i \subset \mathbf{X}$, $i = 1,\ldots,n$, and a collection of contractive mappings $\tilde{w}_i: D_i \rightarrow \mathbf{X}$, $i = 1,\ldots,n$. We consider grayscale images as real-valued functions $f(x,y)$ defined on the unit square $I^2 = I \times I$. Let $\tilde{w}_i(x,y)$ be an affine transformation on $I^2 \rightarrow I^2$, that is,

$$w_i^{\sim}(x,y) = \mathbf{A}_i(x,y) + \mathbf{b}_i \tag{2.1}$$

for some 2×2 matrix $\mathbf{A}_i$ and 2×1 vector $\mathbf{b}_i$. Let $D_i \subset I^2$ be some subdomain of the unit square I^2, and let R_i be the range of $w_i^{\sim}$ operating on D_i, that is, $w_i^{\sim}(D_i) = R_i$. We can now define w_i operating on images $f(x,y)$ by

$$w_i(f)(x,y) = s_i f(w_i^{\sim -1}(x,y)) + o_i \tag{2.2}$$

provided $w_i^{\sim}$ is invertible and $(x,y) \in R_i$. The constant s_i expands or contracts the range of values of f, and so, in the context of grayscale images, it controls contrast. Similarly, the constant o_i raises or lowers the grayscale values, and so controls brightness. The transformation $w_i^{\sim}$ is called the spatial part of w_i.

The basic algorithm proceeds as follows. Partition the image into non-overlapping rectangular range cells $\{R_i\}$. The R_i may be of equal size, but more commonly some type of variable sizing is used to allow a concentration of small range cells in parts of the image that contain detail. The results presented here were produced with a quadtree partition scheme as described in [2]. Cover the image with a sequence of possibly overlapping domain cells. The domains occur in a variety of sizes, and there typically may be hundreds or thousands of domain cells. The affine transformation given by (2.1) is a spatial contraction if $|\det \mathbf{A}_i| < 1$. We will limit the choices for $w_i^{\sim}$ to spatial contractions with rigid translation and one of eight basic rotations and reflections, as described in [2]. For each range cell R_i, we attempt to find the domain D_i, the spatial transformation $w_i^{\sim}$, the contrast s_i and brightness o_i such that $w_i(f)$ is close to the image f on R_i. That is, we want to find w_i such that the quantity

$$\int_{R_i} \left| w_i(f)(x,y) - f(x,y) \right|^2 dxdy \tag{2.3}$$

is small. For a digitized image, the integral in (2.3) represents a summation over pixels. If the quantity (2.3) is not less than some preset tolerance after the best w_i has been found, then the quadtree scheme subdivides the range into four smaller rectangles, and the search for an optimal transformation is repeated on these smaller ranges. This process continues until the quantity (2.3) can be brought within tolerance, or until the maximum preset quadtree depth has been reached. The image is decoded by iteratively applying the transformation W to the image f, where

$$W(f)(x,y) = w_i(f)(x,y) \text{ for } (x,y) \in R_i.$$

If the transformations $\{w_i\}$ have been correctly chosen, the iterate $W^{\circ n}(f)$ will be close to the original image f for some reasonable value of n. The encoding phase is computationally intensive because of the large number of domains which must be searched for each range cell, and because of the computations which must be performed for each domain-range comparison. This paper addresses the computational demands of the encoding phase in two ways. Frist, we introduce the notion of image features defined for each domain and range cell. Candidate domain cells can then be selected based on a small number of feature values, rather than actual pixel values. Second, we introduce a neighborhood topology on the domain cells by means of a self-organizing neural network. This further reduces encoding time by allowing one to quickly locate domain cells which look like the range cell in feature space.

3. Feature Extraction

One way to improve the basic fractal image coding process is to extract a small number of features which characterize the domain and range subimages. The comparison of domains and ranges is then based on these features rather than on individual pixels, thus reducing the complexity of the problem. This paper uses six features that measure image texture and contrast distribution. Feature extraction by itself provides a significant speedup in the encoding process.

The following six features are used here: (i) standard deviation, σ; (ii) skewness, which sums the cubes of the differences between pixel values and the cell mean, normalized by the cube of σ ; (iii) neighbor contrast, which measures the average difference between adjacent pixel values; (iv) beta, which measures how much the pixel values change in relation to the value at the center of the cell; (v) horizontal gradient, which measures change in pixel values across the cell; and (vi) vertical gradient, which measures change in pixel values from top to bottom. Mean is not used as a feature, since the contrast and brightness are varied during the domain-range cell matching process. The feature vector is normalized when comparing distance in feature space so that the larger feature values do not dominate the comparison.

4. Self-Organizing Neural Networks

The second improvement which can be made is to classify the domains and ranges based on these features, and to compare ranges only with domains in similar image classes. Here we use a classification scheme based on Kohonen self-organizing neural networks [7] to classify the domains. This type of network consists of a lattice of node positions. Attached to each node is a weight vector whose dimension matches the dimension of the feature vectors. The dimension of the feature vectors used here is 6. The lattice used here consists of 10 rows and 10 columns of nodes. Each node represents a class of domain image cells, so we want to keep the overall number of nodes fairly small. It is possible to consider other node arrangements, such as higher dimensional matrices.

Network training is through unsupervised learning. The network of weight vectors is initialized to random values. An input feature vector is then presented to the network and we find the weight vector closest to the input vector. That is, find *i',j'* such that

$$\|\mathbf{v} - \mathbf{w}_{i',j'}\| \le \|\mathbf{v} - \mathbf{w}_{i,j}\| \text{ for all } i,j$$

where $\mathbf{v}$ is the input feature vector and $\mathbf{w}_{i,j}$ is the weight vector at node i,j. Adapt the weights in the lattice neighborhood of the winning weight $\mathbf{w}_{i',j'}$ to look more like the input vector. This adaptation is summarized as:

$$\mathbf{w}_{i,j}^{\text{new}} = \mathbf{w}_{i,j}^{\text{old}} + \varepsilon \exp\left(\alpha \left\| \mathbf{v} - \mathbf{w}_{i,j}^{\text{old}} \right\|^2\right)\left(\mathbf{v} - \mathbf{w}_{i,j}^{\text{old}}\right)$$

where i,j range over a neighborhood of i',j'. The size of this neighborhood is reduced during the course of the training iterations. The parameter ε is the iteration stepsize, and α is inversely proportional to the stepsize. The code implementation for the results presented here is that given in [8].

Once the network has been trained, the domain cells for the given image are classified by assigning them to the weight vector to which they are closest in feature space. When a range cell feature vector is presented to the network, it is similarly assigned to a network weight vector. The range cell is then compared to those domain cells assigned to this weight vector, as well as to those domains assigned to the neighboring weight vectors on the network lattice. An advantage of the self-organizing network approach to classification is this concept of neighboring image classes which is provided by the network lattice. Another advantage is that this clustering of image types occurs without the need to pre-define image classes.

5. Results

Table 1 compares fractal image compression results using three different methods applied to the images shown in Figures 1 and 2. The baseline ("Base") method is the standard quadtree method as discussed in [2], with no domain classification. The quadtree level indicates the maximum quadtree depth allowed. A large number here allows smaller range rectangles, which leads to better quality in the decoded image, but worse compression. "Error threshold" is a parameter which controls when the range rectangles are subdivided into smaller rectangles at the next highest quadtree level. Error measures are computed by comparing the original bitmap image with an image that has been decoded using 6 iterations (more iterations would have resulted in slightly smaller errors). "Average Error %" is the average error per pixel, while "PSNR" is the peak signal-to-noise ratio, as computed in [3]. The features-only ("FO") method compares domain and range cells based on the six features discussed in section 3. The final method ("SO") classifies domains using the self-organizing neural network approach with feature extraction, as discussed above. In each case, a total of 320 domain cells were used. A larger number of domains would have increased encoding times and provided marginally better compression ratios.

Compression ratios are estimates based on an average of 4 bytes per range rectangle compared with the 66614 bytes of the original bitmap. The self-organizing method is roughly twice as fast as the features-only method and one hundred times as fast as the baseline method ("Time per Rect" provides a time performance measure that is independent of the final image fidelity; times reported here are for a 120MHz Pentium PC). The self-organizing network was trained off-line on an image separate from the two images shown here, so that training time is not included in the total for this method. The decoded image quality and compression for the faster methods are comparable to the baseline method.

6. Conclusions

Feature extraction combined with self-organizing neural network classification of domains provides a significant speedup of encoding times. Basing image classification on a small number of features not only reduces the dimensionality of the computations, but also allows network training on an image separate from the test image, thus removing network training time from the total encoding time. The self-organizing network establishes its own set of representative domain classes, as well as defining a neighborhood topology on the domain classes.

References

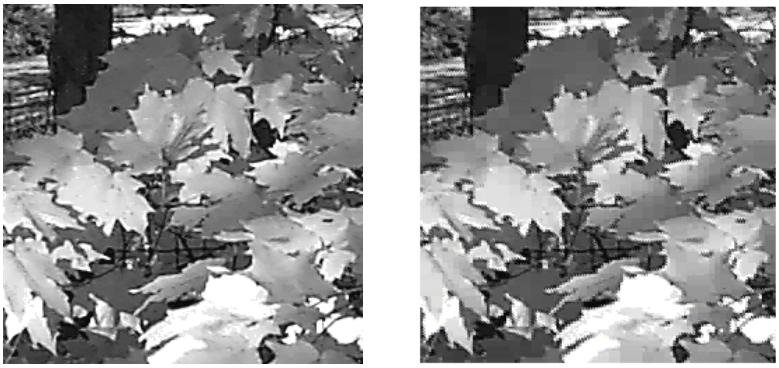
[1] E. DeJesus, "Walking, Talking Web", *Byte*, January, 1997, 81-84.
[2] Y. Fisher, ed., *Fractal Image Compression*, (New York: Springer-Verlag, 1995).
[3] A. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations", *IEEE Trans. Image Proc*. 1, 1992, 18-30.
[4] A. Bogdan and H. Meadows, "Kohonen neural network for image coding based on iteration transformation theory", Proc. SPIE 1766, 1992, 425-436.
[5] R. Hamzaoui, "Codebook clustering by self-organizing maps for fractal image compression", NATO ASI Conf. On Fractal Image Encoding and Analysis, July, 1995.
[6] M. Barnsley, *Fractals Everywhere*, 2nd ed. (Boston: Academic Press, 1993).
[7] T. Kohonen, *Self-Organization and Associative Memory*, (Springer-Verlag, 1989).
[8] S. Welstead, *Neural Network and Fuzzy Logic Applications in C/C++*, (New York: John Wiley and Sons, 1994).

Method	Image	Error Threshold	Quadtree Level	No. of Rect's	Time (sec)	Time per Rect (sec)	Average Error %	PSNR (dB)	Compression Ratio
Base	Rose	0.05	6	1492	3433	2.30094	4.0919	23.55	11.16
FO	Rose	0.05	6	2683	132	0.04920	3.1624	24.78	6.21
FO	Rose	0.05	7	8902	421	0.04729	1.7223	29.68	1.87
SO	Rose	0.05	6	1738	46	0.02647	3.8358	24.01	9.58
SO	Rose	0.05	7	3484	101	0.02899	3.0371	26.36	4.78
SO	Rose	0.025	6	2620	57	0.02176	3.0551	25.21	6.36
SO	Rose	0.025	7	6649	174	0.02617	1.8006	30.48	2.50
SO	Leaves	0.05	6	2608	60	0.02301	6.1114	19.83	6.39
SO	Leaves	0.025	7	10369	297	0.02864	3.0455	24.66	1.61

Table 1. Results of fractal image compression using self-organizing domain classification (method "SO"), domain comparison using features only ("FO"), and the baseline fractal image compression method ("Base").

(a) **(b)**

Figure 1. (a): The original "Rose" bitmap (256 × 256, 256 gray levels); (b): Decoded image (6 iterations, quadtree level 7, 1.8% average pixel error, 2.5:1 compression)



(a) **(b)**

Figure 2. (a): The original "Leaves" bitmap (256 × 256, 256 gray levels); (b): Decoded image (6 iterations, quadtree level 7, 3.05% average pixel error, 1.6:1 compression). Higher detail level in this image leads to poorer compression performance.