O
E

L
E
T
T
E
R
S

Fast *K*-dimensional tree-structured vector quantization encoding method for image compression

A. Meyer-Bäse
K. Jancke
A. Wismüller
Florida State University
Department of Electrical and Computer Engineering
Tallahassee, Florida 32310-6046

M. Georgiopoulos
University of Central Florida
School of Engineering and Computer Science
Orlando, Florida 32816-2362

Abstract. This paper presents a fast *K*-dimensional tree-based search method to speed up the encoding process for vector quantization. The method is especially designed for very large codebooks and is based on a local search rather than on a global search including the whole feature space. The relations between the proposed method and several existing fast algorithms are discussed. Simulation results demonstrate that with little preprocessing and memory cost, the encoding time of the new algorithm has been reduced significantly while encoding quality remains the same with respect to other existing fast algorithms. [DOI: 10.1117/1.1683885]

Subject terms: fast vector quantization; large codebooks; still image compression.

Paper 03102 received Oct. 23, 2003; revised manuscript received Dec. 1, 2003; accepted for publication Dec. 9, 2003; appeared online Dec. 11, 2003.

1 Introduction

Nearest neighbor search is based on finding the closest point to a reference point among M such points in the K-dimensional (K-d) space. Reducing the complexity of nearest neighbor search is of considerable interest in vector quantization (VQ) encoding.

To overcome this problem, several fast quantization algorithms have been developed. We can classify previous work into two groups.

The first group seeks a suboptimal solution in the sense of a mean squared error. The second group addresses an exact solution of the nearest neighbor encoding problem with less computation than that of exhaustive search. In this context, K-d trees have been widely used for fast search. In Ref. 1 the emphasis lies on the optimal design of the K-d tree for efficient nearest neighbor search in multidimensional space under a bucket-Voronoi intersection search framework. The main disadvantage of this method is a relatively low vector dimension. The fast encoding method proposed in Ref. 2 exploits the topological structure of the codebook but is designed for relatively small codebooks.

In this paper, a new fast method is presented which significantly reduces the computations without large memory cost and is applicable for large codebooks and vector dimensions.

2 Description of the Encoding Method

Our proposed method is based on the idea to build clusters containing a certain number of reference vectors. Thus, only a certain number of clusters have to be searched and the closest reference vector will be found within these clusters. This reduces the number of necessary floating point operations significantly.

The clustering works as follows: first one empty cluster is generated, which contains the whole input space (R^n). A vector $\mathbf{x}$ belongs to a cluster $\mathbf{i}$, if for each dimension k, $k=1,\cdots,n$ the following relation holds $\min_k^i < \mathbf{x}_k < \max_k^i$. The mins and maxs denote the lower and upper bounds of the cluster $\mathbf{i}$ in each dimension. Now the reference vectors are put one after another in the appropriate cluster (at the very beginning in the empty initial cluster).

Once the number of the vectors in one cluster exceeds an application-based threshold value, this cluster is split into two child clusters. The range of the two child clusters in the input space is determined as follows: the dimension of the largest deviation of the reference vectors within the parent cluster is determined. Let $d_{\max}$ be the dimension of the largest deviation, then the ranges of the two child clusters for this specific dimension are for the left cluster $(\min_{d_{\max}}, \mathrm{mid}_{d_{\max}})$ and for the right cluster $(\mathrm{mid}_{d_{\max}}, \max_{d_{\max}})$, while the ranges for the other dimensions remain unchanged and are the same for both the left and right cluster. The mins and the maxs are the parent's bounds; $\mathrm{mid}_{d_{\max}}$ is the average of the two closest values of the reference vectors in this dimension. Each time a cluster is split into two child clusters, the parent cluster becomes a node in the tree, and the child-clusters are new leaves of the tree. This means the generated tree will be binary with the leaves containing all reference vectors. The nodes can be used to look up a certain cluster. Every nonterminal node is associated with a region and a partitioning hyperplane of the form $\mathbf{x}$: $x_{d_{\max}} = h$, which needs storage of two scalar quantities $(d_{\max}, h)$ at each node, and also the pointers to the child-clusters themselves. The quantity $d_{\max}$ is the coordinate axis orthogonal to the hyperplane and corresponds to the dimension of the largest deviation of the reference vectors within the parent cluster, while h is the location of the plane on this axis. The location is given by the average of the two closest values of the reference vectors in this dimension. Thus, a better discriminatory power can be achieved in a denser data distribution.

This process is repeated until all reference vectors of the codebook have been processed and put into the appropriate cluster.

Once all vectors are distributed to the clusters, the closest reference vector to a given input vector can be found without searching the whole table, but just a couple of clusters have to be searched. First the cluster where the input vector belongs is looked up. This takes only a few compares, because the clusters can be ordered in a tree-structure, which is generated while the clusters are built. Now the Euclidean distance to all reference vectors in the

found cluster is calculated and the closest codeword is found.

The complexity of the tree-structured vector quantization (TQ)-lookup algorithm can be determined as follows: the number of compares which are necessary for finding the right cluster depends on the level of the tree. If the reference vectors are uniformly distributed within the codebook, the tree is balanced, and the number of compares is given by the following equation (provided that each leaf of the tree contains the same number of reference vectors): $N_{\text{compares}} = \log_2 N_{\text{total}} / N_{\text{cluster}}$, in which N_{compares} is the number of compares, N_{total} is the total number of reference vectors, and N_{cluster} the average number of elements in each cluster. The number of elements per cluster varies from one to the maximum number of vectors which are allowed per cluster. Results for the average number of elements per cluster are given in Section 3. After the right cluster is determined, this cluster has to be searched to find the nearest reference vector within this cluster. The number of distance calculations is N_{cluster}. If the search is stopped at this point, quite good results can be achieved with a minimum of computations (see Table 1). The TQ algorithm which is executed just until this point will be called Fast TQ, the full TQ will be called Accurate TQ. If the search is continued to find the best approximation, the number of computations mainly depends on the distribution of the reference vectors within the codebook and on the dimension of the reference vectors. TQ will be compared to two other algorithms: exhaustive search and the quantization algorithm proposed in Ref. 2 (in the following referred to as Li and Salari algorithm). The exhaustive search does not need to store any further information, so there is no memory overhead. The Li and Salari algorithm works by calculating the distances from the reference vectors to some fixed vectors, and storing these distances associated with the reference vectors. The number of fixed vectors proposed in Ref. 2 is three, and this is the number of fixed vectors we used in our simulations.

3 Simulation Results

Computer simulations using six mammographic images from the Mammographic Image Analysis Society (MIAS) database were performed to evaluate the proposed method in comparison with some other fast algorithms. We use three codebook sizes of $N=256$, $N=1024$, and $N=65536$ codewords each. The codebooks were generated by a neural network algorithm based on the ''neural-gas'' network. The performance of the ''neural-gas'' network is better than the usually employed Lloyd algorithm. The vector codebook was trained by using five 1024×1024 mammograms using the ''neural-gas'' network. The images are monochrome with 256 graylevels. The vector dimension n is $4\times4=16$. After the codebook converged, the quantization was performed using images, which were not included in the training set. In our simulations the maximum number of vectors per cluster was set to 7 resp. 19, the average number of vectors per cluster was 5 resp. 14.

The simulation results are summarized in Table 1. The interpretation of the results yields that the Fast TQ is very fast even if used with large codebooks. If used with smaller maps, it is still faster than the Li and Salari quantizer. The memory usage depends on the number of reference vectors per cluster; for 256 vectors it is 1228.8 bytes, which is still less than the memory usage of the Li and Salari algorithm (6144 bytes). The Accurate TQ is fast for large codebooks and uses the same amount of memory as the Fast TQ. The resulting quantized image has the same peak signal to noise ratio of 21.24 dB as that for exhausting search, but the process of quantizing is much faster. For small codebooks (256 vectors) the Li and Salari algorithm performs better. The values stated in Table 1 are normalized to the number of vectors that were quantized, i.e., a value of 1024 in the column + means that in average 1024 additions were needed to quantize one vector.

Table 1 Comparison of computation performance for Fast TQ, Accurate TQ, Li and Salari algorithm, and exhaustive search. The overall performance is determined by multiplications, additions, comparisons (MACs) and is needed for the sorting of the codevector distances from the hyperplanes for different codebook sizes. (Codebook is tested with an image outside the training set.)

Size	Algorithm	*	±	Compares
256	Fast TQ (7)	21.24	42.48	5.32
	Acc. TQ (7)	54.88	108.72	133.96
	Acc. TQ (19)	91.87	182.73	100.66
	Li and Salari	34.96	65.96	101.08
	Exh. search	1024	2048	256
1024	Fast TQ (7)	21.33	42.67	5.33
	Acc. TQ (7)	176.03	351.05	390.84
	Acc. TQ (19)	279.65	558.3	304.55
	Li and Salari	667.62	1331.24	1704.11
	Exh. search	4096	8192	1024
65536	Fast TQ (7)	19.64	39.28	4.92
	Acc. TQ (7)	267.92	534.8	598.76
	Acc. TQ (19)	287.32	573.64	433.92
	Li and Salari	4995.72	9987.4	26542.52
	Exh. search	262144	524288	65536

4 Conclusion

A new fast vector lookup technique for large codebooks has been presented in this letter. It has several advantages. The main advantage of the TQ algorithm is that a region in the codebook, which is close to the best matching reference vector, can be found with just a couple of comparisons. If applied to large codebooks, it is the fastest algorithm so far known. No *a priori* knowledge is necessary, and it can be applied for high vector dimensions. It is flexible, i.e., it is applicable to every codebook without modifications. The memory usage is small, so that it can be used even in memory restricted environments. An important application area of the proposed method is telemedicine and it is expected to support the very high archiving and transmission requirements of digital medical images, as required for example in teleradiology.

References

1. V. Ramasubramanian and K. Paliwal, ''Fast *k*-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding,'' *IEEE Trans. Signal Process.* **40**(12), 518–531 (1992).
2. W. Li and E. Salari, ''A fast vector quantization encoding method for image compression,'' *IEEE Trans. Circuits Syst. Video Technol.* **5**(2), 119–123 (1995).