A Finite-State Multiresolution Approach to Image Compression Using Pruned Nested Tree-Structured Vector Quantization

Sharon M. Perlmutter & Robert M. Gray
Information Systems Laboratory
Stanford University
Stanford, CA 90435

Abstract

An algorithm is described for constructing a finite-state compression code that is both progressive and multiresolution. The codec consists of nested levels of tree-structured vector quantizers (TSVQs) where the codebook for each level of the nested TSVQs is constructed from the terminal leaves of the TSVQ from the previous level. The first level of the TSVQ represents a finite-state next state function. The codeword dimension at the subsequent levels is greater than or equal to those of the previous levels. This property allows the codec to produce a multiresolution output in a progressive manner. Pruning is performed on the nested TSVQs to achieve the bit allocation across the levels. The resulting pruned TSVQ decoder operates entirely by successive table lookups, with no arithmetic computation. Furthermore, it provides superior performance to ordinary pruned TSVQ at low bit rates.

1 Introduction

Shannon theory implies that the performance of a vector quantizer (VQ) can be improved by increasing the vector dimension. The computational complexity of an unconstrained VQ, however, increases exponentially with the block size. One method to reduce the computational complexity is to use some form of constrained VQ, such as a tree-structured VQ. However, even with a TSVQ the computational complexity increases linearly with increasing block size. Another technique for improving the performance of a VQ without greatly increasing its complexity is to incorporate memory into the VQ with a finite-state vector quantizer (FSVQ) [3]. An FSVQ consists of a finite state space, an initial state, an encoder, a decoder, and a next state function. The encoder and decoder contain identical copies of a finite set of subcodebooks, where each subcodebook belongs to a distinct state. The current input vector is encoded using the subcodebook designated by the next state function. The next state function uses previously encoded vectors to ascertain the state of the encoder/decoder pair. Thus, given an arbitrary initial state known to both encoder and decoder, the decoder can track the state sequence without using side information. An FSVQ thus allows the use of more codewords at a particular bit rate without the need to transmit additional bits to the decoder. Yet another approach to coding large block sizes is to use some form of hierarchical scheme. With a hierarchical VQ [3], a feature vector of reduced dimensionality is extracted from a large-dimensional vector. The feature vector is then coded by some VQ scheme. The decoded feature vector is subsequently used to generate an approximation to the large-dimensional vector. In this paper, we consider a combination of both a tree-structured FSVQ and a hierarchical VQ. We first code the coarsest possible resolution and then recursively code the finer resolutions of the pyramid. Unlike traditional Gaussian pyramid methods, no differences are formed and coarser levels affect finer levels only through codebook selection.

The codec presented in this paper possesses several useful properties. Since the algorithm is based on pruned tree-structured vector quantization (PTSVQ), it has a natural progressive quality. As such, the decoder is able to construct increasingly better quality images as bits arrive. This attribute speeds data base browsing and image selection. The use of a hierarchical scheme with PTSVQ also permits the code to be progressive in spatial resolution. Images are thus transmitted progressively from small image to large. This combination of progressive and multiresolution transmission provides a solution to the problem of mismatch between the image being transmitted and the screen dimensions of the receiver. This, in turn, allows receivers of different screen dimensions to receive information from the same bit stream.

1058-6393/95 $4.00

A variety of schemes exist to provide both progressive and multiresolution codes, including subband and wavelet methods. Although the technique considered in this paper does not provide the performance of the best wavelet methods, it provides similar properties with a very simple code. In particular, the decoder requires no arithmetic operations. This makes it a useful approach to applications requiring fast decoding with low complexity in software.

2 Algorithm

There are three basic goals to the algorithm. First, we would like to exploit the advantages of using large-dimensional vectors. Second, we would like to avoid the large computational complexity of a large-dimensional VQ, and third, we desire a spatially multiresolution output that is transmitted progressively from small image to large. We can accomplish these goals by designing a variable dimension (or multiresolution) PTSVQ. This "multiresolution PTSVQ" will be in the form of nested levels of TSVQs which are subsequently pruned according to [2] to achieve the bit allocation across levels. The codewords at each level of the multiresolution PTSVQ, except for level 0, have dimension greater than or equal to those of the levels of lower resolution. Each coordinate in a low resolution codeword can correspond to a block of pixels in the original image and in higher resolution codebooks. Thus level 1 of the tree corresponds to the lowest resolution (smallest dimension) codewords, and the bottom level of the tree corresponds to the finest resolution (largest dimension) codewords. The top level (level 0) of the PTSVQ consists of a finite-state next state function. This design algorithm combines the multiresolution structure of [5] with a finite-state code for lowest resolution, incorporating memory into the code in a simple but effective way.

The small-dimensional codewords of the lower resolution levels of this multiresolution PTSVQ will be used to encode small-dimensional feature vectors extracted from large-dimensional pixel vectors. Here we consider sample means as the small-dimensional feature vectors. By coding these low-dimensional features of large pixel blocks, we are both coding for low spatial resolution and performing a "classification" of the observed large block for purposes of selecting good codebooks for finer resolution.

In general, it is difficult to obtain medium to high bit rates with very large block sizes. As such, a quadtree decomposition (as implemented in [6]) will be performed on the large-dimensional pixel vectors

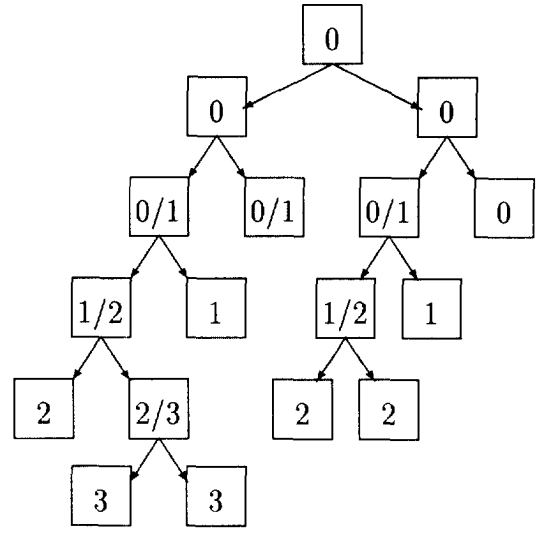


Figure 1: Example of the structure of the finite-state multiresolution PTSVQ

(say, of size $M \times M$) in order to form four smaller dimensional pixel vectors (each of size $M/2 \times M/2$) at some level of the multiresolution PTSVQ. We here consider a single decomposition made after the second level of the multiresolution PTSVQ, whereby we decompose a 64-dimensional (64-D) pixel vector into four 16-D pixel vectors.

The basic structure of this finite-state multiresolution PTSVQ is illustrated in Figure 1. The nodes labeled 0 belong to the finite-state next state function. The nodes labeled 1 have 1-D codewords and belong to the level 1 PTSVQ. These 1-D codewords are used to code the 1-D sample means that are extracted from 8×8 pixel blocks in the original image. The nodes labeled 2 have 4-D codewords and belong to the level 2 PTSVQ. These 4-D codewords are used to code a 4-D feature vector that is extracted from each 16-D pixel block. The nodes labeled 3 have 16-D codewords and belong to the level 3 PTSVQ. These 16-D codewords are used to code the actual 16-D pixel blocks. Note that some terminal leaves of the full multiresolution PTSVQ belong to the lower resolution levels of the tree. This is because the higher resolution level TSVQs that extended from these nodes have been pruned. The details of the implementation of this algorithm are given below.

Let $X^{8\times8}$ be the input pixel vector to be encoded. Let y_n represent the codeword at node n and τ_i indicate the set of terminal nodes in the level i PTSVQ. The complete encoding of $X^{8\times8}$ consists of the follow-

ing:

1. Form a 1-D level 0 sample mean $m_0(X^{8\times8})$ from the 8×8 pixel block. This is the arithmetic average of all the pixels labeled 0 in the 8×8 block shown in Figure 2(a).

2. Form a "state predictor" vector, $(\hat{m}_1(X_L), \hat{m}_1(X_D), \hat{m}_1(X_A))$, from the concatenation of the reconstructed means of the vectors to the left, diagonal (in the northwest direction), and above the current input vector. These means have been reconstructed using the level 1 PTSVQ.

3. Find the state of the current vector $X^{8\times8}$ by performing a binary search with the state predictor vector through the level 0 PTSVQ. The terminal node of the level 0 PTSVQ to which the state predictor vector is mapped represents the state of the current input vector $X^{8\times8}$. Note that no bits need to be sent to the decoder since the decoder has the previously decoded vector $(\hat{m}_1(X_L), \hat{m}_1(X_A), \hat{m}_1(X_D))$.

4. Encode the mean $m_0(X^{8\times8})$ with the level 1 PTSVQ that extends from the terminal node that determined the state of $X^{8\times8}$. The output codeword at this node, $\hat{m}_1(X^{8\times8})$, will be used to determine the state of future vectors.

5. If the encoder has reached a terminal leaf in the multiresolution tree, stop. Otherwise, perform a quadtree decomposition to obtain $X^{8\times8} = (w_0^{4\times4}, w_1^{4\times4}, w_2^{4\times4}, w_3^{4\times4})$. This quadtree decomposition is illustrated in Figure 2(b). Calculate the local mean in each quadrant of each of the $w_k^{4\times4}$, $k = 0, 1, 2, 3$, vectors. Figure 2(c) indicates the pixels from which each mean is taken (each mean is taken from all pixels with the same label). Note that from each of the four 16-D $w_k^{4\times4}$ vectors we have extracted four local means. Thus, we now have 4 separate 4-D vectors, $m_2(w_k^{4\times4})$, $k = 0, 1, 2, 3$. Encode each of the resulting 2×2 mean vectors, $m_2(w_k^{4\times4})$, $k = 0, 1, 2, 3$, with the 4-D level 2 PTSVQ that extends from the terminal node that produced its respective level 1 mean, $\hat{m}_1(X^{8\times8})$.

6. We can now either interpolate each of the 4-D codewords $\hat{m}_2(w_k^{4\times4})$, $k = 0, 1, 2, 3$ (here we use a constant interpolation function) to obtain four 16-D reproductions, $\hat{w}_k^{4\times4}$, $k = 0, 1, 2, 3$, or encode the actual 16-D pixel blocks, $w_k^{4\times4}$, $k = 0, 1, 2, 3$, with the level 3 PTSVQ. The performance of both of these options is presented in Section 3.

Note that once we have encoded the means from each of the 64-D pixel blocks in the image with the level 1 PTSVQ, we have a low spatial resolution representation of the image that has $\frac{1}{64}$ of the original number of pixels. Once we have encoded each of the $m_2(w_k^{4\times4})$, $k = 0, 1, 2, 3$, means from each of the 64-D pixel blocks, we have a finer representation of the image that is $\frac{1}{4}$ of the original number of pixels. Similarly, once we have encoded each of the $w_k^{4\times4}$, $k = 0, 1, 2, 3$, vectors from each of the 64-D pixel blocks, we obtain the full reconstructed image.

The codebook is designed as follows:
Let $\beta(\alpha(V))_{i,j}$ be the reproduction vector produced by encoding a feature vector V with the level i TSVQ that originated from the terminal node j of the level $i-1$ TSVQ. In the case where $i = 0$, j is the root node. Let S be the set of training vectors and y_n be the codeword at node n. Here we generalize to an input vector of size $M \times M$, where M is an integer assumed to be a power of 2, and we let τ_i indicate the set of terminal nodes in the level i TSVQ.

1. For each input vector, $X_t^{M\times M}$, $t \in S$, form a 4-D vector, $(m_0(X_{L_t}), m_0(X_{D_t}), m_0(X_{A_t}), m_0(X_t^{M\times M}))$ that consists of the concatenation of the state predictor vector with the mean of the current input vector. Construct a TSVQ using the generalized Lloyd algorithm on these vectors. The codewords at node j in the level 0 TSVQ will be referred to as $(y_{p,j}, y_{m,j})$, where $y_{p,j}$ is the 3-D codeword associated with the state predictor vectors that map to node j and y_m is the codeword associated with the means of the actual input vectors that map to node j. Note that the design is "open loop" in that the means of the original pixel values are used by the predictor TSVQ, while the actual encoder will "close the loop" by using reconstructed means so that the decoder will be able to track the state. Although the open loop codebook design was chosen for simplicity, the design can naturally be extended to the closed loop case. The input vectors are partitioned with respect to the mean squared error (MSE) between the three-dimensional state predictor codeword of node j, $y_{p,j}$, and the three-dimensional state predictor vector. The splitting criterion for the TSVQ is based on the MSE between the full-sized input vectors that map to a node and the interpolation of the codeword associated with the means

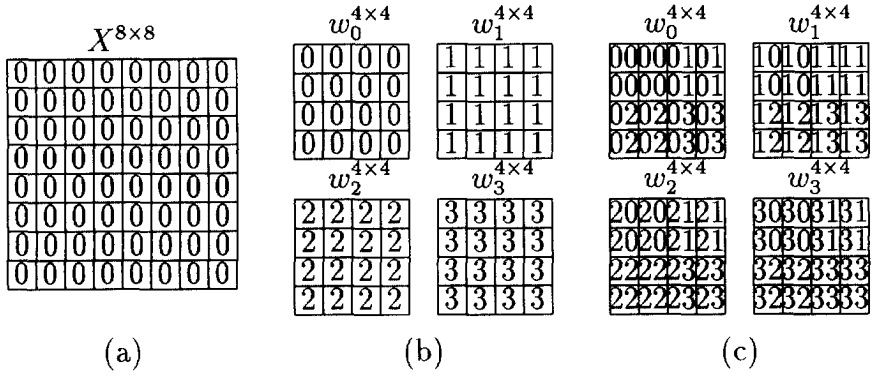


Figure 2: (a) Pixels from which the level 1 mean is computed. (b) Example of decomposition of $X^{8\times8}$ into $w_k^{4\times4}$, $k = 0, 1, 2, 3$. (c) Pixels from which four separate 4-D level 2 mean vectors are computed.

of the vectors that map to the node, $y_{m,j}$. Here we use a constant interpolation function of the form $g(y_{m,j}) = (\underbrace{y_{m,j}, ..., y_{m,j}}_{M\times M})$. All nodes of this TSVQ are referred to as level 0 nodes.

2. From each $j \in \tau_0$, construct a 1-D TSVQ from $\{m_0(X_t^{M\times M}),\ t \in S : \beta(\alpha(m_0(X_{L_t}), m_0(X_{D_t}), m_0(X_{A_t})))_{0,root} = y_{p,j}\}$. These level 0 terminal nodes are also labeled as level 1 since they are root nodes for the level 1 TSVQs. Each of the nodes in the newly constructed TSVQs is considered a level 1 node.

3. Perform a quadtree decomposition of each input vector to obtain $X_t^{M\times M} = (w_{t,0}^{m\times m}, w_{t,1}^{m\times m}, w_{t,2}^{m\times m}, w_{t,3}^{m\times m})$, where $m = \frac{M}{2}$. Form a local mean in each of the quadrants of each of the $w_{t,k}^{m\times m}$, $k = 0, 1, 2, 3$, vectors. Each of the $w_{t,k}^{m\times m}$ now consists of a 4-D mean vector, $m_2(w_{t,k}^{m\times m})$.

4. From each $j \in \tau_1$, construct a 2×2 TSVQ from $\{m_2(w_{t,k}^{m\times m}), k = 0, 1, 2, 3, t \in S : \beta(\alpha(m_0(X_{L_t}), m_0(X_{D_t}), m_0(X_{A_t})))_{0,root} = y_{p,c}$ and $\beta(\alpha(m_1(X_t^{M\times M})))_{1,c} = y_j\}$. At this point we have a 3-level TSVQ.

5. Form quadrants in each of the quadrant subblocks identified in step 3. Form a local mean in each of these new quadrants. Each $w_{t,k}^{m\times m}$ now has sixteen local means which together constitute a 16-D vector, $m_3(w_{t,k}^{m\times m})$. Note that if the original input vector was of dimension 8×8 these means would be the actual pixel values.

6. From each $j \in \tau_2$, construct a 4×4 TSVQ from $\{m_3(w_{t,k}^{m\times m}), k = 0, 1, 2, 3, t \in S : \beta(\alpha(m_0(X_{L_t}), m_0(X_{D_t}), m_0(X_{A_t})))_{0,root} = y_{p,c}$, $\beta(\alpha(m_1(X_t^{M\times M})))_{1,c} = y_n$, and $\beta(\alpha(m_2(w_{t,k}^{m\times m})))_{2,n} = y_j\}$. At this point we have a 4-level TSVQ.

   The previous two steps can be omitted or repeated depending on the dimension of the original input vector and complexity constraints. The initial rate to which each level's TSVQ is grown can be determined empirically or by a lookahead procedure.

7. Prune the multi-level TSVQ to form a multiresolution PTSVQ according to the technique described in [2]. This operation determines the final bit allocation across the multi-level TSVQ.

By pruning across different resolution TSVQs, it is possible for terminal nodes of the pruned TSVQ to be nodes from any resolution level. Thus, if a region consists of very low activity, when the nested TSVQs are pruned, the terminal nodes that correspond to the codewords of this region can be level 0 or level 1 nodes. These nodes also contain interpolated codewords, which will provide the appropriate reconstruction at the higher resolution levels. Thus no additional bits need to be sent to the decoder to code the higher resolution levels. Here we use a simple constant interpolation function, but optimal nonlinear interpolation [3] could also be performed. Note that by pruning across the different TSVQs we eliminate the need for the threshold detector common in some multiresolution systems [6].

An advantage of the above system is that the computational complexity is asymmetrical at the decoder and the encoder. In particular, unlike most multiresolution systems [1, 6], the decoder contains no arithmetic operations. The average computational complexity at the encoder consists of the calculations to

perform the binary Euclidean nearest neighbor test at each node and the mean calculations. These calculations are on the order of those needed with an ordinary tree-structured finite-state VQ [4].

3 Results

The training sequence used to design the codebook consisted of ten 12 bit 512×512 computerized tomography (CT) images. We chose two CT test images that were not in the training sequence. Results are reported on the average of these two images. Performance was measured by $SNR = 10 * \log_{10}(D_0/distortion)$, where D_0 is the distortion obtained with the best zero rate code, and the distortion measure is MSE. During the codebook design, the initial rate to which each level's TSVQ was grown (i.e., before pruning) was determined empirically. We looked at two variations of our algorithm. In both cases the original sized input vectors are 8×8. In the first method, we designed a 4-level multiresolution codebook as described by the above design algorithm. Thus, the largest codeword dimension is 16. In the second method, we designed a 3-level multiresolution codebook where steps 5 and 6 of the above design algorithm are not implemented. Thus, the largest codeword dimension is 4. The performance of these two coders was compared to an ordinary PTSVQ and to an "ordinary multiresolution PTSVQ" (one that does not contain a finite-state component). Figure 3(a) compares the SNR vs. bit rate performance of the 4-level finite-state multiresolution PTSVQ with an ordinary multiresolution PTSVQ of greatest codeword dimension 16 and with ordinary 16-D PTSVQ. Figure 3(b) compares the SNR vs. bit rate performance of the 3-level finite-state multiresolution PTSVQ with an ordinary multiresolution PTSVQ of greatest dimension 4 and with ordinary 4-D PTSVQ. Since the system does not include entropy coding, the bit rates reported are real bit rates, not entropy rates. The 4-level finite-state multiresolution system outperformed ordinary 16-D PTSVQ at bits rates below 0.44 bpp. For example, we obtained more than 3 dB improvement at 0.16 bpp. The 3-level finite-state multiresolution system outperformed ordinary 4-D PTSVQ at bit rates below 0.70 bpp. In particular, we obtained more than 8 dB improvement at 0.25 bpp. We note that the introduction of the finite-state component in both the 4-level and 3-level systems improved the performance at the lower bit rates by as much as 1 dB. Figure 4(a) shows an original uncompressed CT chest image. Figure 4(b) shows the CT chest image compressed to 0.36 bpp using the 4-level finite-state multiresolution algorithm.

4 Conclusions

We have presented a novel algorithm for progressively generating images from small to large. This algorithm is asymmetrical in computational complexity at the encoder and the decoder. In particular, the decoder contains no arithmetic operations. Simulations on CT images demonstrated that the algorithm outperforms ordinary PTSVQ at low bit rates.

4.1 Acknowledgements

This work was partially supported by the National Science Foundation under grant NSF MIP-9016974. The authors gratefully acknowledge the assistance of Keren Perlmutter.

References

[1] P. J. Burt and E. H. Adelson. The Laplacian pyramid as a compact image code. *IEEE Trans. Comm.*, COM-31:552–540, April 1983.

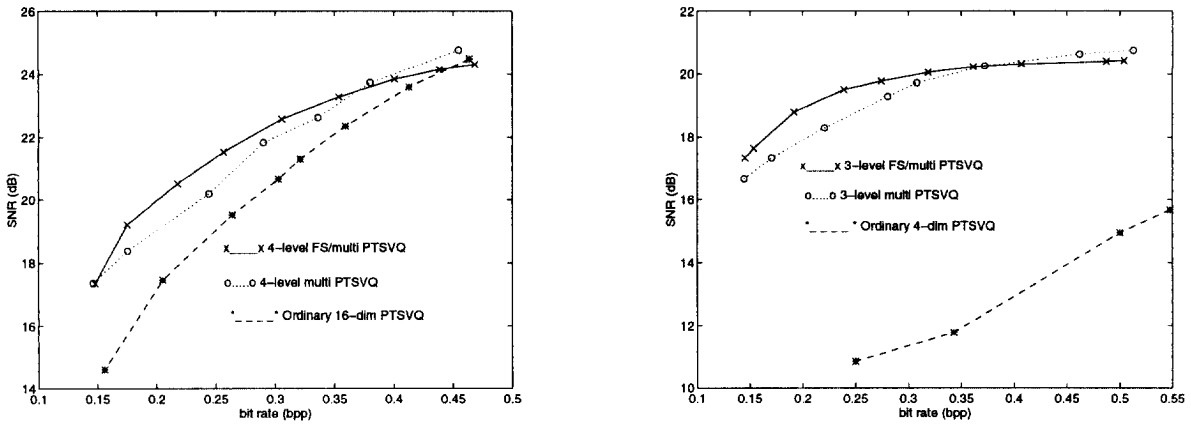
[2] P. A. Chou, T. Lookabaugh, and R. M. Gray. Optimal pruning with applications to tree-structured source coding and modeling. *IEEE Trans. Inform. Theory*, 35(2):299–315, March 1989.

[3] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Boston, 1992.

[4] Y. Hussain and N. Farvardin. Variable rate finite-state vector quantization. Presented at the Twenty-Fourth Annual Conference on Information Sciences and Systems, Princeton, NJ, March 1990.

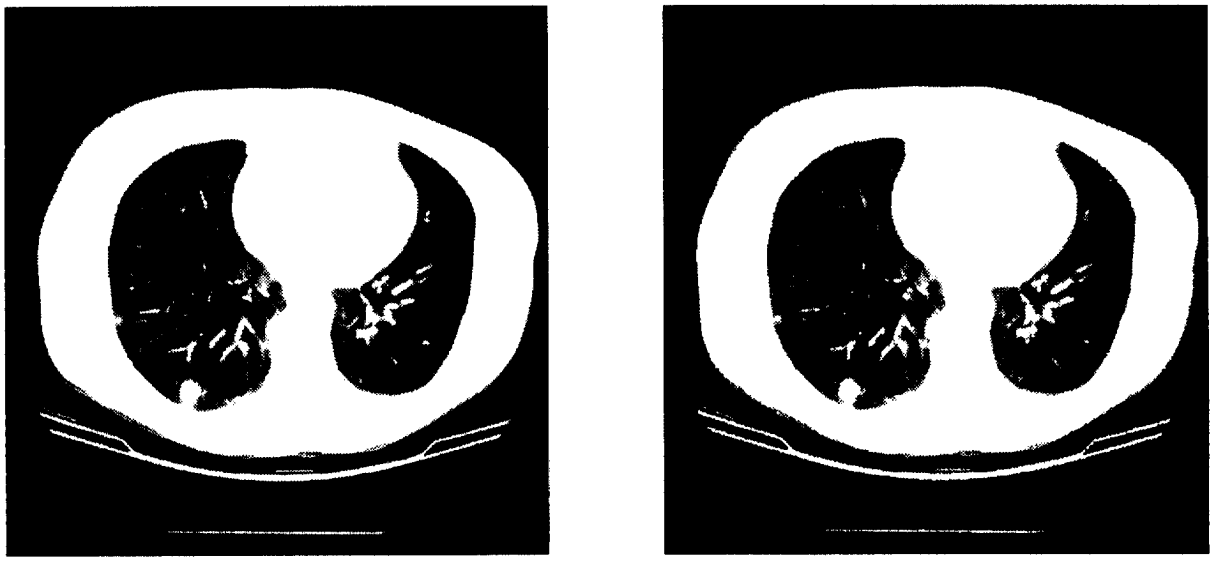
[5] S. M. Perlmutter and R. M. Gray. A low complexity multiresolution approach to image compression using pruned nested tree-structured vector quantization. In *Proc. International Conference on Image Processing*, Austin, Texas, Nov. 1994.

[6] Y. Yamada and T. Tazaki. Recursive vector quantization for monochrome video signals. *IEICE Trans.*, E 74(2):399–405, Feb. 1991.



(a) (b)

Figure 3: (a) SNR vs. bit rate for 4-level finite-state multiresolution PTSVQ, 4-level ordinary multiresolution PTSVQ, and ordinary 16-D PTSVQ and (b) SNR vs. bit rate for 3-level finite-state multiresolution PTSVQ, 3-level ordinary multiresolution PTSVQ, and ordinary 4-D PTSVQ



(a) (b)

Figure 4: (a) Original uncompressed CT image and (b) CT image compressed with 4-level finite-state multiresolution PTSVQ to 0.36 bpp