Using Vector Quantization for Image Processing

Pamela C. Cosman, Karen L. Oehler, Eve A. Riskin, Robert M. Gray

Abstract

Image compression is the process of reducing the number of bits required to represent an image. *Vector quantization*, the mapping of pixel intensity vectors into binary vectors indexing a limited number of possible reproductions, is a popular image compression algorithm. Compression has traditionally been done with little regard for image processing operations that may precede or follow the compression step. Recent work has used vector quantization both to simplify image processing tasks – such as enhancement, classification, halftoning, and edge detection – and to reduce the computational complexity by performing them simultaneously with the compression. After briefly reviewing the fundamental ideas of vector quantization, we present a survey of vector quantization algorithms that perform image processing.

1 Introduction

Data compression is the mapping of a data set into a bit stream to decrease the number of bits required to represent the data set. With data compression, one can store more information in a given storage space and transmit information faster over communication channels. Suppose a source is producing symbols from an alphabet of size 2^b at a rate of R symbols per second. Each symbol can be described with an index that is b bits long. Because the rate of the source is Rb bits per second, the data would need to be compressed to be transmitted over channels with capacity less than Rb.

The two types of data compression are *lossless* and *lossy*. Lossless compression has the advantage that the original information can be recovered perfectly from the compressed data. Short indices (less than b bits) are assigned to high probability symbols, and long indices (more than b bits) are assigned to low probability symbols. Lossless codes are variable rate because the number of bits allocated to source symbols differs. Whereas the source would still produce R symbols per second, only $Rm \leq Rb$ bits per second are required on average to identify them. The compression ratio is defined as $b{:}m$, and is limited by the source entropy rate, a measure of the randomness inherent in the source. For example, if all the source symbols are independent and equally likely (maximum randomness), a lossless code must have a bit rate that is no lower than Rb bits per second. Alternatively, you could use a lossy code that operates at rates below the source entropy rate. Lossy compression introduces error into the data, so the original data cannot be perfectly recovered. Vector quantization (VQ) [1, 21, 59], described in the next section, is a popular algorithm for lossy compression.

An image compression system may serve as a front end for a digital image processor. Digital image processing is the processing of a two-dimensional set of data. Among others, it includes representation and modeling, enhancement, restoration, analysis and reconstruction. Images are often processed in different ways, and there are open questions about how the image processing operations interact with each other. Which operation should be performed first? Which operation makes another processing task simpler or more complex? This paper addresses the question: Do any image processing operations naturally and effectively combine with VQ algorithms? Two attributes of VQ suggest that such smart compression is possible. First, the VQ design techniques of clustering and classification trees have a long history of applications to image processing, including enhancement and classification. Second, since a VQ system uses a collection of possible image reproduction blocks, an image processing routine can be applied to this set of blocks ahead of time rather than to the compressed image itself.

By combining other signal processing goals into the design of the VQ, the compression system can be better customized for a particular application. Isolated examples of these ideas have appeared in the literature, but little attempt has been made to unify them as a common approach; this is our goal. We begin with a description of unstructured (full search) vector quantization and tree-structured implementations. We then examine a number of variations on the VQ design algorithms that allow for the incorporation of image processing into the compression system.

2 Vector Quantization

Vector quantization is an image compression algorithm that is applied to vectors rather than scalars, and it can be easily understood through scalar quantization. Scalar quantization maps a large set of numbers to a smaller one and includes such operations as "rounding to the nearest integer," although in general the quantization levels do not have to be neither integers nor evenly spaced. Vector quantization rounds off (or *quantizes*) groups of numbers together instead of one at a time. These groups of numbers are called *input vectors*, and the quantization levels are called *reproduction vectors*. To specify a vector quantizer, one needs the set of possible reproduction vectors and a rule for mapping input vectors to the reproduction vectors. A two-dimensional example of a VQ is shown in Figure 1. The dots represent the reproduction vectors and the mapping rule is indicated by the lines, which delineate the boundaries between regions. Any input vector lying in a given region maps to the reproduction vector in that same region.

Another way of depicting this system is in Figure 2, which shows a VQ that operates directly on image pixel blocks. The input image is parsed into a sequence of groups of pixels, possibly – 2×2 squares as shown in the figure, but larger squares and rectangles and other shapes are often used. The encoder views an input vector X and applies its mapping rule to select one of the N possible reproduction vectors from its codebook. The chosen reproduction vector Y_i is also called a *codeword* and is (usually) a grayscale pixel block of the same dimension as the input block. The index i of Y_i is binary. Notice that X and $\hat{X}$ are slightly different in Figure 2 to demonstrate that VQ is a lossy compression technique. If the code has a fixed rate of b bits per input vector, then i has length b. With a variable rate code, the indices i have variable length, and b is their average length. The compressed image is represented by these indices i, and the compressed representation requires fewer bits. For example, for an 8 bit per pixel (bpp) original image, the input block requires $4 \times 8 = 32$ bits. For a fixed rate code with 256 codewords in the codebook, each codeword has an 8-bit index. Thus the compression ratio is 32:8, or 4:1. The decoder also has a copy of the codebook, and it operates as a simple table look-up. Upon receiving an index i, the decoder puts

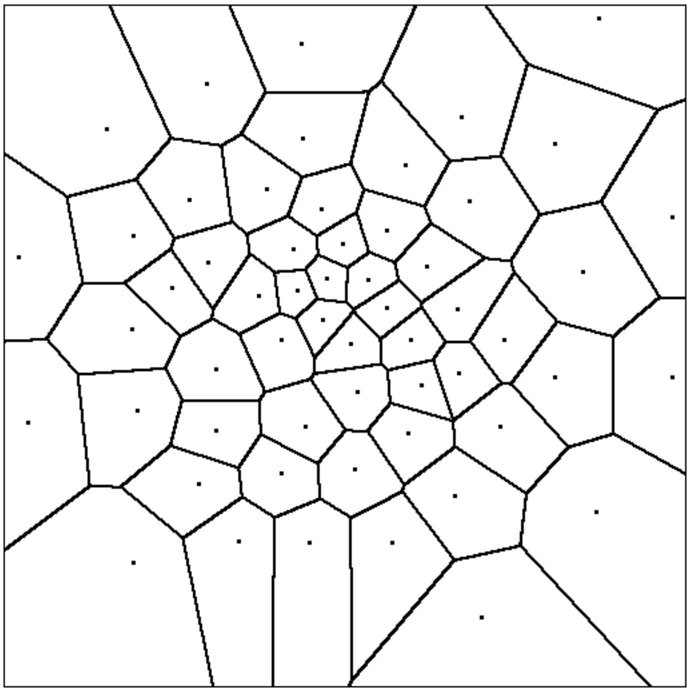

Figure 1: 2-D example of VQ.

out the stored codeword Y_i.

The operation of the decoder is thus completely described once we have specified the codebook. The operation of the encoder requires a choice of the mapping rule. The basic Shannon source code model provides an encoder that is optimal for a given codebook if the goal is to minimize an average distortion. If we assume $d(X, \hat{X}) \geq 0$ measures the distortion or the cost of reproducing an input vector X as a reproduction $\hat{X}$, and if we further assume that the overall distortion (or lack of fidelity) of the system is measured by an average distortion, the optimal encoder for a given codebook selects the vector Y_i if

$$d(X, Y_i) \leq d(X, Y_j), \text{ all } j.$$

In other words, the encoder operates in a nearest neighbor or minimum distortion fashion. A *full search* VQ is an unstructured collection of codewords. The encoder determines the closest one by an exhaustive search. A structured codebook uses a constrained search to speed up the encoding, but it is not guaranteed to find the overall nearest neighbor in the codebook.

The choice of distortion measure permits us to quantify the performance of a VQ in a manner that can be computed and used in analysis and design optimization. By far the most commonly used distortion measure for image compression is the mean squared error, in spite of its often cited shortcomings. Although there are many approaches to code design, the algorithms surveyed here are all based on clustering techniques, such as the Lloyd (Forgey, Isodata, k-means) algorithm. The Lloyd algorithm has been described in detail in a variety of places (see, for example [1, 21, 24]). It iteratively improves a codebook by alternately optimizing the encoder for the decoder (using a minimum distortion or nearest neighbor mapping) and the decoder for the encoder (replacing the

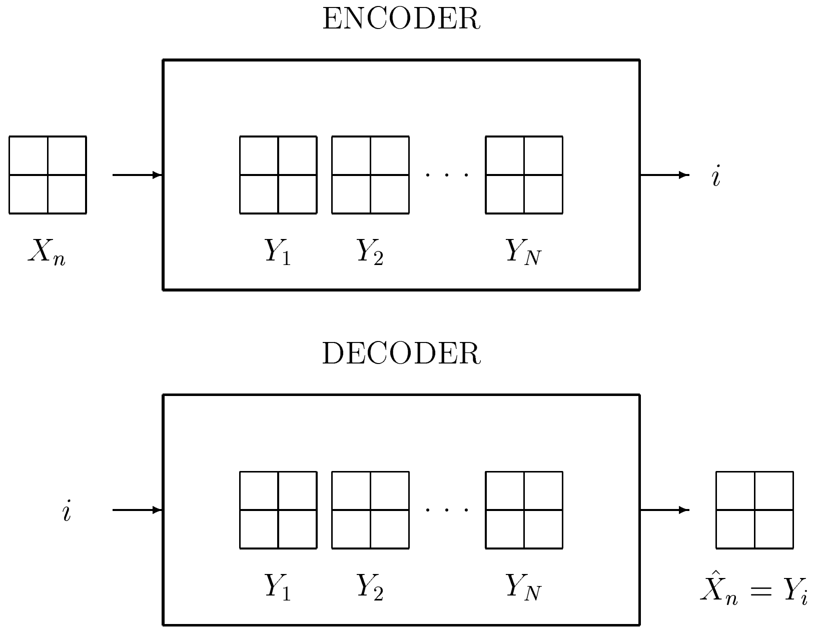

Figure 2: Vector Quantizer.

old codebook by generalized "centroids"). For squared error, centroids are the Euclidean mean of the input vectors mapping into a given index. Code design is usually based on a training set of typical data rather than on mathematical models of the data. For example, to produce a VQ for magnetic resonance chest scans, one begins with a set of 20 to 30 representative scans. These images are divided up into *training vectors,* and the clustering algorithm is run on this set.

3 Tree-Structured Vector Quantization

Shannon theory states that VQ can perform arbitrarily close to the theoretical optimal performance for a given rate if the vectors have sufficiently large dimension. Unfortunately code complexity grows exponentially with vector dimension. The practical solution to this "curse of dimensionality" is to constrain the code structure. This solution results in codes that are not mathematically optimal; but it will likely provide better performance with implementable codes for a given rate. There are many common constrained code structures [1, 21], including lattice-based codes, classified VQ, multistep VQ, product codes (gain/shape and mean removed), predictive VQ, finite-state VQ, and tree-structured VQ. The only constrained code structure we describe here is tree-structured VQ (TSVQ) because it is used in several of the combined systems for compression and image processing.

TSVQ avoids the full search of an unstructured codebook. Figure 3 depicts two simple binary trees. In both cases, the codeword is selected by a sequence of binary decisions. Vector reproductions are stored at each node in the tree. The search begins at the root node. The encoder compares the input vector to two possible candidate reproductions, chooses the one with the minimum distortion, and advances to the selected node. If the node is not a terminal node (leaf) of the tree, the encoder continues and chooses the best available node of the new pair presented. The encoder produces binary symbols to represent its sequence of left/right decisions. The stored index is then a path map through the tree to the terminal node, which is associated with the final codeword. The two trees differ in that the one on the right is *balanced* and all indices have the same length R. This tree yields a fixed rate code. The other is *unbalanced* and has indices of differing length. Here the instantaneous bit rate – the number of bits per input vector or pixel – changes, but the average rate is constrained. Like a lossless code, an unbalanced tree gives the code the freedom to allocate more bits to active areas and fewer bits to less active areas such as background. The goal in lossy compression, however, is to choose long or short codewords to minimize average distortion for a given bit rate, not to match improbable or probable vectors as in lossless coding. The search complexity of a balanced tree is linear in the bit rate instead of exponential but at the cost of a roughly doubled memory size. For unbalanced trees the search complexity remains linear in the *average* bit rate, but the memory can be considerably larger unless constrained.

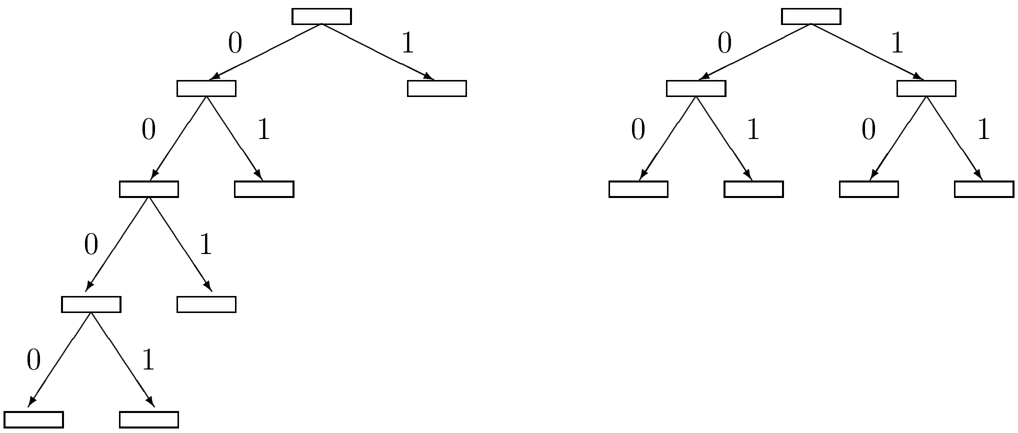

Figure 3: Unbalanced and balanced trees.

Growing Trees

Tree-structured codes are designed by combining clustering with ideas from classification and regression tree design [3]. Classification trees apply a sequence of tests to an input to classify it. The general philosophy of classification tree design is a gardening metaphor: First grow a tree and then prune it. To grow a TSVQ, one begins with a set of training vectors and calculates their centroid. This centroid is the optimum rate 0 codeword and is associated with the root node. The cluster of data is then split into two subclusters. The split can involve perturbing the root node centroid slightly and associating the root node centroid and its perturbation with the two new child nodes. The iterative clustering algorithm is then run on the pair. The data points shift back and forth between the two subclusters until the subclusters stabilize, and their centroids represent the codewords at the first level of the tree. There are now two quite different options.

Split all terminal nodes (balanced trees):

The tree can be extended by simultaneously splitting the two clusters associated with the two current terminal nodes and running the iterative clustering algorithm on each pair. For each current terminal node, only the training vectors in that node's cluster are used to design the node's children. The clustering algorithm will eventually converge for all the split clusters to give a new balanced tree with four nodes in the second level. One continues to grow the tree in this manner. This technique is depicted in Figure 4.

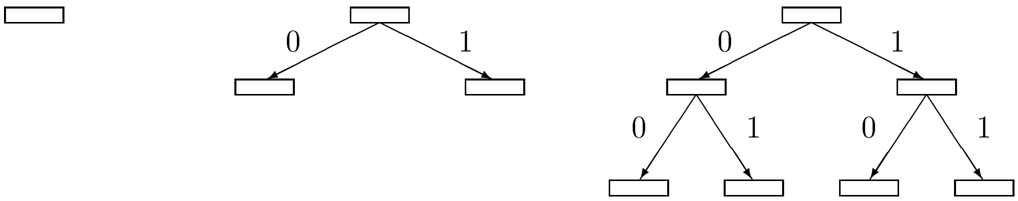

Figure 4: Growing a balanced tree.

Balanced trees have two clear drawbacks. As the tree grows, some nodes may become sparse in training vectors, and the resulting clusters may not generalize well to outside data. Some nodes may even be empty and will waste bits. Also, there will likely be too many codewords to represent inactive vectors and too few to represent active vectors.

Split one node at a time (unbalanced trees):

An alternative design paradigm is to split nodes one at a time rather than an entire level at a time. After the level one codebook has converged, one chooses one of the two child nodes to split and run a clustering algorithm on that node alone to obtain a new unbalanced tree with three terminal nodes. One then repeats the procedure, choosing one of the three nodes. This continues until the tree reaches the desired average rate. The two common methods of choosing which node to split are not optimal. The first is to split the node with the largest contribution to the average distortion [42]. The second approach is optimal in an incremental or greedy fashion. Splitting a node causes an increase in average rate, ΔR, and a decrease in average distortion, ΔD. We split the node that maximizes the magnitude slope $|\Delta D/\Delta R|$ to get the largest decrease in average distortion per increase in average bit rate [62]. This splitting algorithm is a natural extension of a fundamental design technique for classification and regression tree (CART$^{\mathrm{TM}}$) design [3].

Pruning Trees

Whether balanced or unbalanced, the growing algorithm is greedy and does not consider the impact that the current split has on future splits. Furthermore, even the unbalanced tree can result in sparsely populated or improbable nodes that cannot be fully trusted to typify long run behavior. A solution to both these problems is to prune the tree [6]. Pruning removes a node and all its descendents and hence reduces the average bit rate and increases the average distortion. The idea is to minimize the magnitude of the increase of average distortion per decrease in bit rate, $|\Delta D/\Delta R|$ (the same quantity we maximized in growing the tree). Because we have the entire tree, we can consider the effect of removing entire branches rather than individual nodes. Thus, we can find the optimal subtrees of an initial tree that provide the best distortion-rate tradeoff. The optimal TSVQs of decreasing rate formed by pruning an initial tree are nested (form embedded codes) which makes the pruning algorithm work efficiently. These codes have a successive approximation character in that the distortion decreases on average as the bit rate increases.

The advantages of variable rate TSVQ are that it usually yields lower distortion than fixed rate full search VQ for a given average rate and block size, encodes with a sequence of binary decisions, and has a simple design algorithm. It also has a natural successive approximation (progressive) property and is well matched to variable rate environments such as storage or packet communications.

4 Digital Image Processing and VQ

Digital image processing can be divided into several different classes of applications, including representation and modeling, enhancement, restoration, analysis, and reconstruction. Typically, when one processes a compressed image, the steps are cascaded as shown in Figure 5. The original data are first compressed and stored as a list of codeword indices. The decoder reads in the indices and generates the decompressed data set (reconstructed image), thereby expanding the compressed data file back to its original size. In Figure 5, the decompressed data look slightly different from the original data to indicate that although the file sizes are the same, the decompressed data are not the same as the original due to the VQ. The processor operates on the decompressed data to generate the processed result. The decompressed data and the processed result are shown differently in Figure 5 to indicate that the processed information may not even be an image.

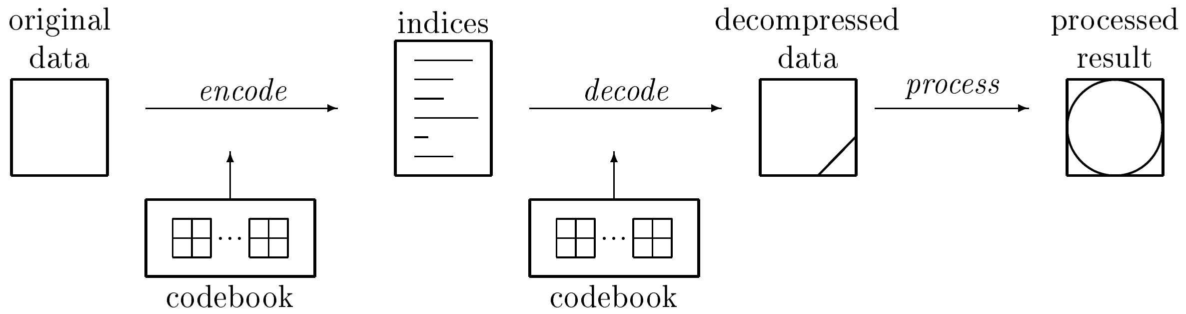

Figure 5: Separate sequential decoding and processing steps with intermediate file generation.

Several ways of combining the decoding and processing steps have appeared in the literature. The simplest of these ways merely eliminates the need for the intermediate decompressed data file. As the decoder retrieves blocks from the codebook, the blocks are immediately sent for processing. This situation is diagramed in Figure 6. As we discuss in Section 8, this combination might be useful when the original data set is very large (e.g., 3-D data of size $512 \times 512 \times 512$ pixels at 8 bpp = 134 Mbytes) and the processed result is much smaller. In this case, one may not be able to accommodate the entire decompressed data set even briefly on the system.

What characteristics of the processing let it be combined with the decompression step? Clearly the processor must be able to operate on subblocks of the image that are no larger than the size of the codewords. In addition, the codewords must be processed independently of each other or at least of the blocks which have not yet been decompressed. In general, a subblock can be anything from a single point to the whole image. The key issues in combining processing with VQ are

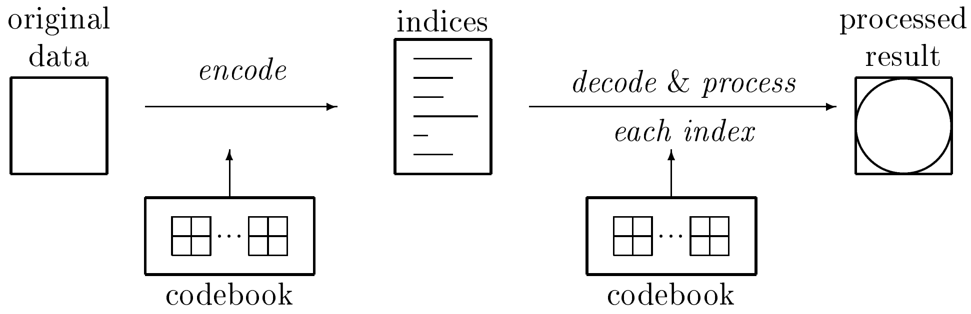

Figure 6: Sequential decoding and processing steps with no intermediate file.

the size of the operational subblocks and whether the subblocks are processed in an independent (nonoverlapping) or dependent manner. If, in addition to operating independently on codeword-sized subblocks, the processor also depends only on information available at the time of codebook design, then the system in Figure 6 can be further simplified. At the time of codebook design, each codeword is processed as if it were a subblock of an image. Each codeword is then linked to the result of processing the codeword, as shown in Figure 7. This result may be another image subblock (a processed version of the codeword), or it may be a piece of information (e.g., a class assignment for the image subblock or a determination of edge orientation within the subblock). We use a circle to represent the result of processing the codeword to differentiate it from the original codewords.

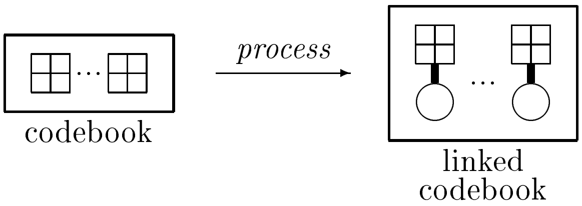

Figure 7: Perform processing on the codebook.

The processing is done off-line at the time of codebook design, and the decoder codebook contains the codewords and processed results linked together. When the system decompresses an image, the decoder can put out the codewords, the processed result, or both, and no additional time is required. This system appears in Figure 8; most of the image compression/processing algorithms we will describe are based on this figure.

A trivial example of combining VQ and image processing is thresholding, which operates on a grayscale image in a pointwise manner. The VQ is designed as usual, and each pixel in each codeword is thresholded to produce the linked codebook. When the codebook is used for decompression, the decoder can select the output from either the original or the thresholded codewords. Thus, a thresholded compressed image can be obtained for no postprocessing costs.

What are the costs in combining VQ and image processing? First, the decoder must store the processed codewords, which requires varying amounts of storage space. Second, if the process-

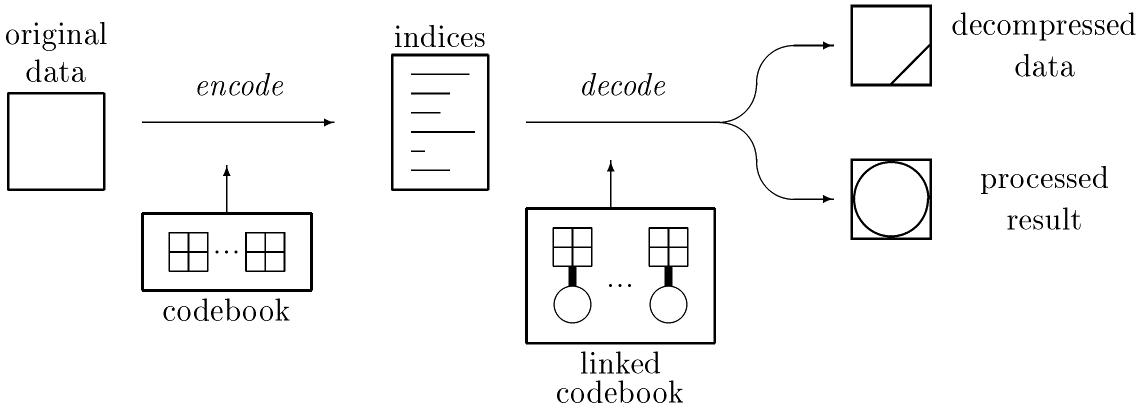

Figure 8: Combined decoding and processing steps.

ing step normally depends on the entire decompressed image, the results obtained by processing the codewords off-line would be different from the results usually obtained. Thresholding, however, works on independent blocks (pixels) and therefore the results obtained by thresholding the codewords in advance or thresholding the decompressed data file are clearly the same.

Even the simple thresholding case may not be as straightforward as described. One may want to modify the standard VQ design algorithm based on the processing operation that follows. With thresholding, many codewords may map to the same thresholded vector, and one could modify the design algorithm not to waste those codeword indices. This situation is diagramed in Figure 9, and classification applications based on this model are discussed in Section 6.

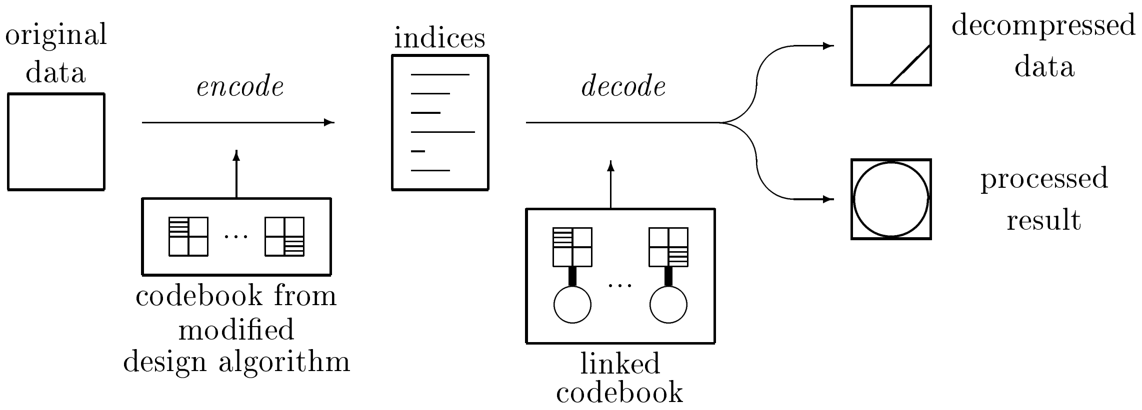

Figure 9: Combined decoding and processing steps, where encoder's codebook has been modified by knowledge of processing.

All the examples in the literature follow one of these models. In the sequel we introduce a variety of image processing operations that can be combined with VQ. We consider the key issues of operational block size, block independence, and the need for altering the VQ design.

Enhancement

The goal of image enhancement is to accentuate certain features of the image for subsequent analysis or display. Examples include contrast enhancement, pseudocoloring, noise filtering, sharpening, and magnifying. VQ has the innate ability to remove "speckle" noise because of the smoothing or averaging performed by the centroid operation. For medical images, it was suggested in [53, 9] that slightly compressed images yielded marginally better diagnostic accuracy and subjective quality ratings than did original images. Of course, such smoothing can be considered as enhancement only if the speckle is indeed undesirable noise and not the signal of importance. A discussion of the use of data compression (not specifically VQ) for filtering random noise is given in [45].

Histogram equalization (HE) is a powerful tool for contrast enhancement. Global HE remaps pixel intensities in a pointwise fashion (thus subblock size $= 1 \times 1$) with a remapping function based on the histogram of the entire image (size $= N \times N$). Despite the discrepancy in the two block sizes, neither of which is suitable for VQ, the operation can be made compatible with VQ by substituting the histogram of the training images used to design the VQ for the histogram of the decompressed input image. Global and adaptive histogram equalization are described in Section 5.

Classification and Analysis

Scene analysis and image understanding range from character recognition and medical image analysis to automatic defect analysis and cartography. Most such problems require examination of blocks considerably larger than a VQ vector, but the algorithms usually begin with low-level operations on small blocks. A VQ can be used for this low-level classification or detection, and the low-level classification itself may be useful. Examples of classification and edge detection are given in Sections 6 and 7. Classification and analysis problems are often solved with tree-structured methods, and thus a natural meld with tree-structured VQ may exist.

Visualization

Scientific visualization is the use of computer graphics techniques for displaying experimental or simulated data. Visualization includes techniques for displaying volumetric (3-D) scalar fields as 2-D images. Such 3-D arrays of digital data representing spatial volumes arise in many scientific applications, including medicine, nondestructive evaluation, astrophysics, and meteorology. Isosurface generation is the construction of a polygonal model which approximates a contour surface in the volume, thereby extracting and displaying a single surface from the volumetric data. Another example is volume rendering, which treats the entire 3-D scalar field as a collection of sources and attenuators and integrates these contributions along the viewing direction to form a projected image of the translucent volume. Section 8 presents an algorithm for combined VQ and volume rendering that provides storage savings and may yield faster rendering.

Reconstruction and Representation

Halftoning is the conversion of a grayscale image to a bilevel image suitable for display on a binary device. A combination of compression and halftoning would be useful for transmitting images by facsimile, transmitting images to printers, and storing images for display on monochrome monitors.

This problem has been examined by Vander Kam *et al.* [34, 35] and is discussed in Section 9. One particular halftoning process, error diffusion, is particularly challenging to combine with VQ because it is a neighborhood process.

Image reconstruction encompasses many methods of generating images from raw data. Examples include reconstruction of medical CT images from 1-D projections and reconstruction of synthetic aperture radar (SAR) images from complex radar returns. Because of the nature of the signal processing required by the SAR processing, VQ can be used to simplify reconstruction computations following compression. This technique is described in Section 10.

5 Contrast Enhancement

Histogram equalization is a contrast enhancement method for increasing the dynamic range of images to bring out features hidden in dark regions or washed out by light areas. Histogram equalization remaps each pixel to an intensity proportional to its rank among surrounding pixels [32]. Transforming each pixel according to the inverse of a cumulative distribution function alters the histogram or empirical distribution of the pixel intensities. If the cumulative distribution function is the empirical distribution of the image, the result is a more uniform distribution of pixel intensities. Histogram equalization effectively widens the perceived dynamic range of the image.

In global histogram equalization, one calculates the intensity histogram for the entire image and then remaps the intensity of each pixel to be proportional to its rank in this global histogram. A single, global remapping function does not provide much flexibility. For example, in the original image of Figure 11, the dark areas of the cortex have intensities in the range of 70–80, and the brighter pixels have intensities > 110. A function that maps pixels in the range of 70–80 to *lower* values, and maps pixel intensities > 110 to *higher* values would enhance the contrast between the various structures in the cortex. In the spinal column area, however, the vertebrae have intensities in the range of 70–80, and the intensities of the darker interstices between the vertebrae range from 30–40. In this area of the image, the remapping function should map 70–80 to *higher* values and 30-40 to *lower* values, to make the vertebrae more clearly distinguishable from their surroundings. Global histogram equalization, however, uses only one mapping function for the entire image.

This problem is addressed by adaptive histogram equalization (AHE), in which the histogram is calculated for pixels in a context region (usually a square) and the remapping is done only for the center pixel of the square. This can be called *pointwise* histogram equalization because, for each point in the image, the histogram is calculated only for the square context region centered on the point and that point alone is remapped. Because this is computationally intensive, the bilinear interpolative version is an alternative to lower the computational complexity [57]. It calculates the histogram for only a set of nonoverlapping context regions that cover the image; the remapping of pixel intensity values is then exact for only the centers of these context regions. For all other pixels, a bilinear interpolation from the nearest context region centers determines the appropriate remapping function.

With the bilinear interpolative AHE, the remapping function for a given pixel of intensity i at location (x, y) is determined from the nearest four context regions as shown in Figure 10. If m_{+-} denotes the mapping at the grid pixel (x_+, y_-) to the upper right of (x, y) and similar subscripts are used for the other surrounding context regions, the interpolated AHE result is given by

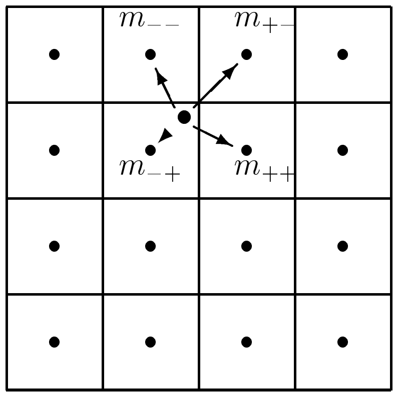

Figure 10: Bilinear interpolative adaptive histogram equalization.

$$m(i) = a[bm_{++}(i) + (1-b)m_{-+}(i)] + \tag{1}$$

$$[1-a][bm_{+-}(i) + (1-b)m_{--}(i)],$$

where

$$a = \frac{y - y_-}{y_+ - y_-}, \qquad b = \frac{x - x_-}{x_+ - x_-}. \tag{2}$$

Pixels in the border regions of the image are handled separately by using a linear interpolation from the two nearest context region centers or, in the corners, using only a single remapping function. A 256×256 pixel image typically has sixteen 64×64 context regions.

Combined VQ and Histogram Equalization

Instead of performing the decoding and equalizing operations sequentially, one can perform them simultaneously by equalizing the decoder's codebook off-line at the time of codebook generation [8]. Although the histograms of the future test images are not available at the time of codebook generation, the histogram of the training sequence is available. The intensity distribution of the training set must be very similar to that of the test set for a VQ codebook to work well, and the histograms would likely be quite similar.

We implemented combined VQ and global histogram equalization by constructing the global histogram for the training set and equalizing the codebook with this global histogram. Each pixel of each terminal node remaps to a new intensity proportional to its rank in the global histogram. The new codewords are stored at the decoder along with the original codewords. The resulting system follows the model of Figure 8. In this application, the original data, the decompressed data, and the processed result are all 256×256 grayscale images. The radiologist can view either the equalized or the unequalized series of compressed scans (or both); both ways require the same amount of time to reconstruct the image.

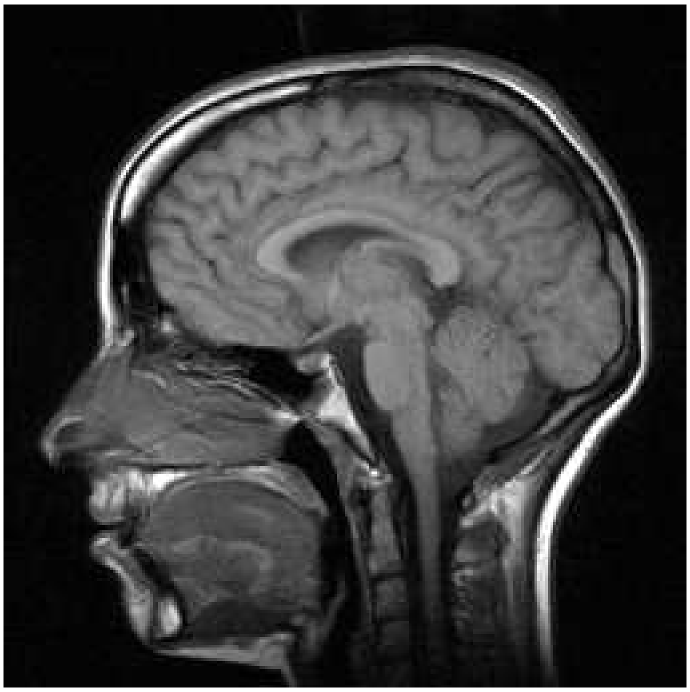

Figure 11: Original image.

The simultaneous combination of VQ and AHE is not straightforward. AHE remaps a pixel's intensity using a histogram local to that pixel, so one must know the pixel's spatial location in addition to its intensity to determine the appropriate remapping function. AHE cannot be simply applied to VQ, because although the intensity of any pixel in a codeword is known, its "location" is unknown. Because the codewords represent centroids of clusters of training sequence vectors, the concept of a codeword's "location" within the training images is vague, because the codeword likely does not exist in any of the images. We divided each training image into the sixteen context regions shown in Figure 10 [7]. The pixels from the corresponding regions of the training images were pooled to form sixteen different intensity histograms. The codewords were equalized using each of the sixteen different histograms, and the resulting equalized versions of the codewords were stored at the decoder along with the original codewords. Because the input image is scanned in a fixed raster order, the spatial location of each vector was known to the decoder. The decoder automatically generated the coefficients a and b from Eq. 2 for the input vector location and selected and linearly combined the four equalized versions of the codeword. The resulting system follows the model in Figure 8, with the reproduction vectors linked to sixteen different equalized versions of themselves. The system differs slightly from the model in that a small amount of post-processing (the linear combination) is required at the decoder.

To demonstrate combined AHE and VQ, an unbalanced tree was grown to an average depth of 2 bpp on a training sequence of 10 magnetic resonance (MR) mid-sagittal brain scans of 10 different subjects. The 256×256 training images were blocked into 2×2 vectors. The tree was pruned back to 1.7 bpp and each leaf codeword equalized over the sixteen histograms from the training sequence. Figure 11 shows the original test image, which is not in the training set. Figures 12 and 13 show the regular compressed and adaptively histogram equalized compressed images. The image quality of the equalized compressed image is very high, and its contrast is enhanced. The invaginations of the cortex are more obvious, and the vertebrae are more clearly differentiated from the interstitial spaces.

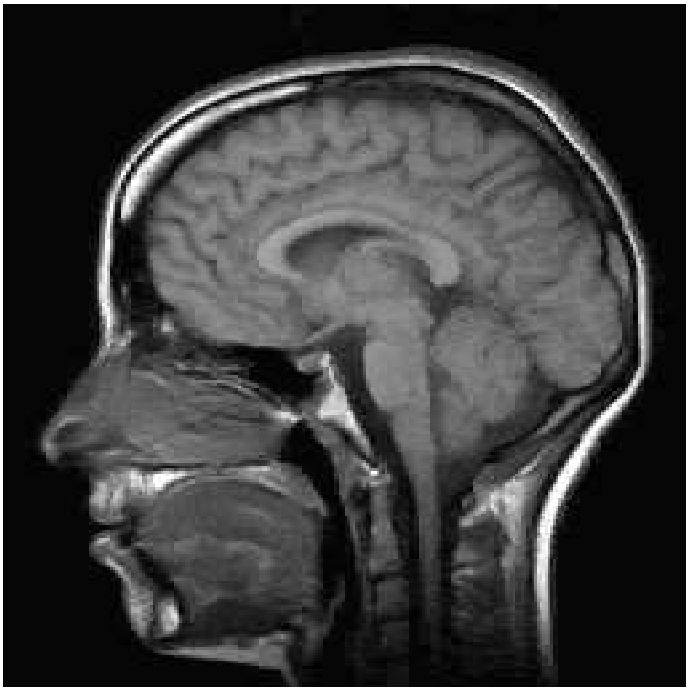

Figure 12: Regular compressed image at 1.78 bpp.

6 Classification

Combining vector quantization with classification is natural because both techniques can be designed and implemented using methods from statistical clustering and classification trees. The goal of such a combination is to incorporate classification information into the codewords by classifying the codewords themselves during code design. By combining VQ and low-level classification, certain simple features in an image can be classified automatically as part of the compression process. The classification requires no more bits to describe than those required for compression alone, an important feature in low memory or low bandwidth situations. Such a combination can be used to highlight regions in the reconstructed image belonging to a specific class or to provide an efficient front end to more sophisticated full-frame recognition algorithms. If the VQ output is intended for classification, the compression design can provide better performance than if the quantization and classification algorithms were designed independently.

The notion of using a VQ to classify is implicit in the classical nearest neighbor (NN) classification algorithms [17, 10]. The NN classifier is constructed by labeling every training vector by its class and then using the entire training set with the corresponding classes as a VQ for future vectors. A new vector is classified by finding its nearest Euclidean neighbor in the training set and then assigning the label of that nearest neighbor to the new vector. In this case, the entire training set is the codebook, which can be extremely large. Although this can be considered as an application of

Figure 13: Adaptively histogram equalized compressed image at 1.78 bpp.

VQ to classifier construction, little compression of the input is realized if the training set is large. One can reduce the codebook size by by eliminating a subset of codewords whose removal causes the least damage to the classifier performance [28, 20, 12]. Instead of applying explicit clustering to minimize the squared error of a reduced set of labeled templates, the nearest neighbor literature considers classification error (or Bayes risk) when reducing the codebook size.

An early example of using VQ for joint classification and compression was described by Hilbert [30]. Hilbert applied clustering techniques to training vectors to produce a code in a manner similar to the Lloyd algorithm. The resulting codewords were labeled using a maximum likelihood classifier developed from the training data. When these codewords were used for encoding, the labels provided simultaneous classification. An adaptive clustering algorithm produced variable size codebooks that better represented variations in the training data. Lossless source coding applied to the encoder output further reduced the bit rate. The algorithm was applied to multispectral Landsat images, which were both compressed and classified into eight classes of crop types. This method follows the model in Figure 8, in which the VQ decoder can either put out codewords for the input vector or the classification of the input vector. In Hilbert's case, the data classification was not incorporated into the clustering algorithm. Thus, the design algorithm was essentially a cascade of separate compression and classification steps, although the overall system provided simultaneous compression and classification.

More recent methods modify the clustering algorithm used to design a VQ to improve classification ability. Such a system has the form of Figure 9. Perhaps the best known such technique is that of Kohonen *et al.* [36, 38, 37], who proposed a variety of learning vector quantizers (LVQ) to perform classification using a VQ encoder and codebook. The encoder operates as an ordinary minimum mean squared error selection of a representative from the codebook but the codebook is designed to attempt to reduce classification error implicitly rather than reducing mean squared error. Kohonen argued that for the case of Gaussian data, the partition induced by a VQ can approximate that required for a Bayes estimator; his algorithm is based on this intuition. Kohonen's approach has been widely used for classification of such disparate applications as the classification of speech sounds [14], of objects in clutter in synthetic aperture radar [27], of proteins [44], of bird songs [47], of oceanic signals [22, 4], and of texture [43].

Kohonen's algorithm is similar to Stone's general formulation of nearest neighbor methods for parametric regression, in which a general weighting dependent on class membership of several nearest neighbors can be applied to the classifier [64]. Viewed in this way, Kohonen's algorithm can be considered a clustered simplification of the nearest neighbor approach. As in the nearest neighbor classifier, compression ability is not explicitly considered in LVQ. Kohonen's general goal was to imitate a Bayes classifier with less complexity than other neural network approaches, but there is no explicit minimization of classification error in the code design.

We describe a method of explicitly combining classification and compression using TSVQ in [51]. The training set is classified by determining those features to be recognized in subsequent images. In this example, aerial photographs are hand-labeled by a human observer as regions of man-made or natural objects. This *a priori* knowledge is used when designing the TSVQ codebook to improve the classification accuracy over that of ordinary TSVQ.

Because the classification and compression are done simultaneously, one tree-structured search both encodes and classifies each image subblock. Stored with each codeword in the codebook is a label for the best class prediction for image subblocks that map into it. The best predictive class for a given codeword is determined by a majority vote of the *a priori* class assignments of the training vectors represented by that codeword. This classification rule is essentially an empirical maximum a posteriori (MAP) classifier based on the VQ output, where the necessary conditional probabilities are estimated by the relative frequencies when the VQ is applied to the training set. Once the encoder selects the best codeword, the preliminary classification of the subblock is a simple table lookup; no other computations are required. In effect, this classification comes "for free" once the codebook has been designed off-line.

The classification accuracy can be explicitly incorporated into the VQ design through the node splitting strategy. Three splitting strategies are natural:

Criterion 1 (Ordinary TSVQ) Split the node with the largest $|\Delta D/\Delta R|$ as in Section 3, yielding ordinary greedily grown TSVQ for comparison.

Criterion 2 Split the node that is "worst" in terms of *percentage* of misclassified training vectors.

Criterion 3 Split the node that is "worst" in terms of *number* of misclassified training vectors.

The encoder and the centroid reproduction levels are chosen to minimize squared error. The first splitting criterion represents an ordinary TSVQ codebook design method, but the latter two splitting criterion represent a modified design method for constructing a codebook with reduced classification error.

In [51] the training set consisted of five images. The images were 512×512 8-bit grayscale aerial photographs of the San Francisco Bay area. Each 16×16 pixel subblock in the training set was classified as either man-made or natural. The original test image is shown in Figure 14.

Figure 14: Original aerial image (8 bpp).

Criterion:	1 - Ordinary TSVQ	2	3
Test PSNR (dB)	23.4	22.8	22.7
Test classification ability	0.71	0.74	0.75

Table 1: PSNR and classification ability using TSVQ codebooks grown using various splitting criteria.

Each of the splitting criteria was used to construct a TSVQ codebook at an average bit rate of 0.5 bpp. A test image outside the training sequence was encoded with the resulting codebooks. Compression and classification ability results are shown in Table 1. The compression ability is measured by the peak signal to noise ratio (PSNR). The classification ability is measured by the fraction of vectors classified correctly by the encoder to the classification standard created by the human observer. In general, the first splitting criterion provided the lowest mean squared error in the encoded images at the expense of reduced classification ability. The other splitting methods provided poorer compression ability (the encoded images were slightly more blocky in appearance) but better classification ability for the test image. Choosing the splitting criterion involves a tradeoff between compression and classification quality; some splits serve one purpose better than the other. This example demonstrates how knowledge of the subsequent processing can be used to modify the codebook design algorithm to improve the subsequent processing. Again, this is the situation we have illustrated in Figure 9.

Ideally, the compressed images would be viewed as a color image so that the classification information could be indicated by color superimposed on the grayscale reconstructed image. Such a contrast makes the natural and man-made features of the image easier for a human viewer to differentiate. Because it is difficult to display the compression and classification results simultaneously in a grayscale image, the results are shown separately here. Experimental data are shown for images encoded and classified with a codebook grown using the second criterion. Figure 15 shows the image compressed to 0.5 bpp. In Figure 16, the subblocks classified as natural are displayed directly while the subblocks classified as man-made are replaced by solid white blocks. The classification ability was modest; at 0.5 bpp the best classification encoder still had 25% misclassification error on the training sequence. This large error partly reflects the quality of the training sequence, however. The hand-labeling was affected by the human observer's resolution and consistency limitations. The classification results were comparable to results from the CART$^{\text{TM}}$ algorithm, a traditional tree-structured classification technique [3].

Figure 15: Image compressed at 0.46 bpp using a compression/classification encoder.

Another approach to using VQ to compress and classify explicitly incorporates a Bayes risk component into the distortion measure used for code design; this trades off mean squared error with classification error [52]. Suppose that one wants to classify the input signal as being in one of two classes (say 0 or 1) and that the cost of misclassifying a vector in class k as being in class j is $C_{k,j}$. Assume that $C_{k,k} = 0$. Given a classifier $h(i)$, which assigns a VQ index i to a class 0 or 1, the Bayes risk is given by

$$B = C_{0,1} \Pr(h = 1 \text{ and } X \text{ is in class } 0) +$$

$$C_{1,0} \Pr(h = 0 \text{ and } X \text{ is in class } 1).$$

One can replace the usual design goal of minimizing an average distortion $E[d(X, \hat{X})]$, such as mean squared error, by a modified average distortion $E[d(X, \hat{X})] + \lambda B$, where λ is a Lagrange multiplier by which the relative importance of average distortion and Bayes risk can be adjusted. The average distortion depends only on the VQ encoder and decoder, whereas the Bayes risk depends only on

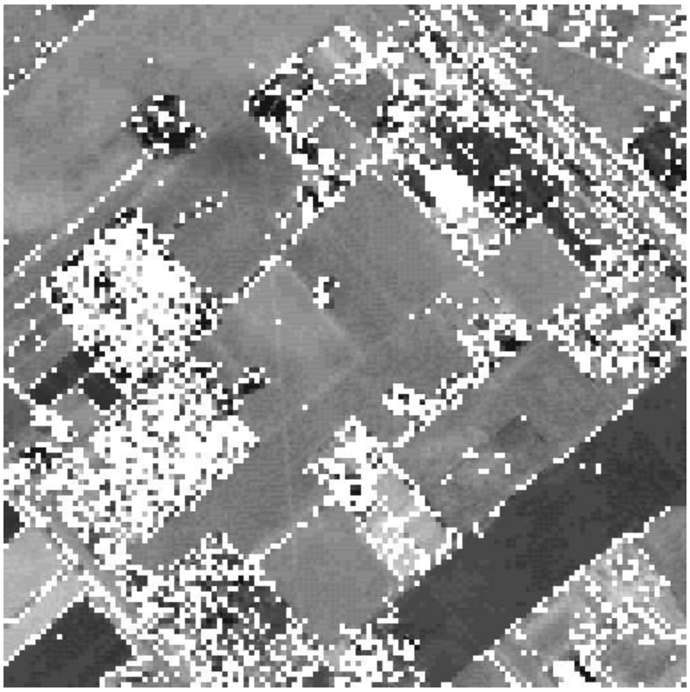

Figure 16: Image compressed at 0.46 bpp using a compression/classification encoder where man-made subblocks are replaced by solid white subblocks.

the VQ encoder and on the classifier function h. A variation of the Lloyd algorithm that resembles the optimization used in entropy-constrained VQ [5] can be run to minimize the modified average distortion: Given an initial codebook (possibly designed to minimize squared error alone) and a classifier (possibly an empirical Bayes classifier based on conditional probabilities in the labeled training set), the following iteration is performed:

- Encode the training data by choosing the VQ index i for each input vector that minimizes the modified distortion, (squared error) + ($\lambda \times$ conditional Bayes risk), resulting from mapping the input into VQ index i.

- Update the reproduction vectors corresponding to the VQ indices by computing the Euclidean centroids of the training vectors mapping into them.

- Update the classifier decision for each VQ index by using a Bayes classifier on the labeled training data.

The iteration is continued until further improvements are negligible. Like the Lloyd algorithm, this is a descent algorithm, and results in a reduction of the average modified distortion at each step.

The conditional probabilities $\Pr(X \text{ is in class k}|X)$ needed to compute the Bayes risk within the training set are simple fractions depending on how many times the observed vector occurs with a specific class label. Outside the training set these probabilities must be based on a model or empirically estimated. Alternatively, outside the training set one can use the simple but suboptimal encoder, which simply minimizes Euclidean distance, as LVQ does.

Figure 17: CT image compressed at 2 bpp using a compression/classification encoder.

This method has been used to analyze simulated data, identify pulmonary tumor nodules in computerized tomography (CT) images, and identify man-made regions in aerial images [52]. For CT images, the algorithm was used to design a VQ codebook that both compressed the images and classified vectors in the images as tumor or nontumor. The locations of the tumors were determined by radiologists, and 2×2 training vectors were labeled accordingly. A 2 bpp full search codebook was constructed using unbalanced classification costs, making missed tumors 100 times more detrimental than false alarms. Compression results on a CT image outside the training set are shown in Figure 17. This image contains three circular tumors in the left lung. The resulting classification of this image is shown in Figure 18. The algorithm correctly identifies substantial parts of each of the three tumors. The classification here involves no extra decoding complexity when implemented and is based entirely on small blocks with no context. It is significantly simpler than many other recognition algorithms and can provide a useful front end to more sophisticated algorithms yielding an overall performance improvement and complexity reduction.

Recent work by Owsley and Atlas used multidimensional ordering of VQ codebooks in classification of vector-series patterns (such as spectrograms) [54]. By structuring the codebooks so that the relative positions of the codevectors in the codebook correspond to distances between codevectors, they made the codevector indices a meaningful representation of the information in the vectors. They were then able to classify based on the series of indices. Classifiers such as neural networks, which would be overloaded by the presentation of the entire spectrogram, benefited from the more efficient representation.

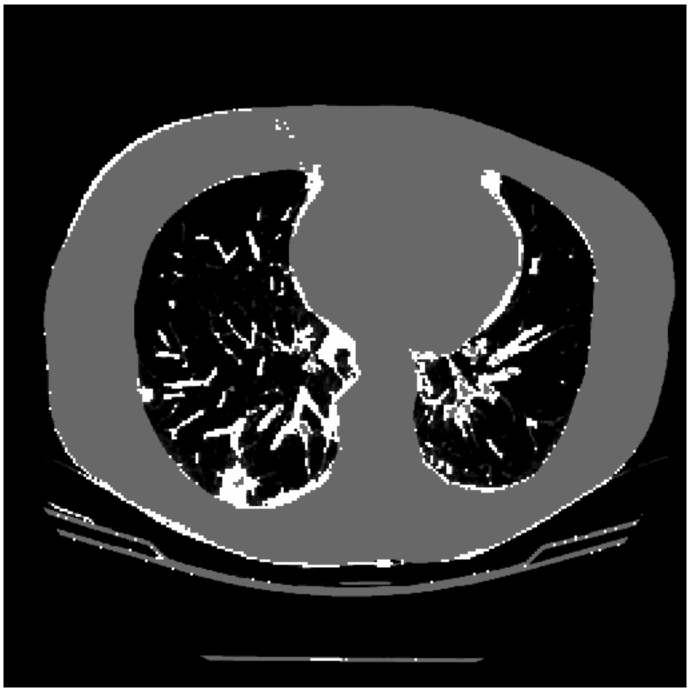

Figure 18: CT image compressed at 2 bpp using a compression/classification encoder where light regions represent vectors classified as tumor.

7 Edge Detection

Variable rate VQ has some inherent edge-detecting properties. A variable rate TSVQ usually uses more bits for regions of greater activity (such as edges) and fewer bits for homogeneous or inactive regions. The number of bits allocated to a particular block provides information about the block. This is illustrated in the instantaneous rate image of Figure 19. The original image of Figure 11 has been encoded with 2×2 blocks to an average rate of 1.8 bpp, with the encoding depth from 4 to 18. The blocks at the greatest depths are shown in white, those with the shortest path lengths are displayed in black, and the depths in between are in shades of gray. The edges around the head as well as some of the internal edges are clearly visible from the instantaneous rate picture.

In another example of using VQ for edge detection, VQ has been used to implement gradient magnitude edge detection [33]. The second directional derivative edge detector described in [25, 26] combines naturally with VQ because it works on small image blocks (typically 5×5 or 7×7). This VQ-based edge detector also fits into the scheme diagramed in Figure 8.

The second directional derivative edge detector estimates the gradient magnitude at each pixel location in the image with a facet. The facet models a digital image as being derived by sampling a continuous underlying grayscale intensity surface. This surface can be represented by a low-degree (usually quadratic or cubic) bivariate polynomial. The gradient direction and magnitude, the second directional derivative in the direction of the gradient, and the contrast are all obtained in terms of the facet polynomial coefficients. The center pixel of the block is labeled as an edge pixel if the second directional derivative in the direction of the gradient has a negatively sloped zero crossing within a threshold radius of the center of the pixel and if the edge contrast exceeds a threshold value.

Figure 19: Instantaneous rate image.

The VQ-based edge detector is applied to each codeword in a codebook and the edge/no edge information is stored with the codeword. Edge detection is then performed by VQ encoding and table lookup of the edge information. The VQ encoding differs from ordinary VQ in that a sliding block is used so that every pixel in the image appears as the center of a block. Usually, the image is tiled in a nonoverlapping manner. This algorithm is a "trainable edge detector," which has lower computational complexity than a conventional gradient edge detector. Further complexity can be saved by encoding with larger VQ blocks to determine edge information for multiple pixels at one time.

The VQ-based edge detector was applied to a motion sequence obtained from a camera mounted on a mobile robot in an outdoor environment, using a greedily grown unbalanced TSVQ with 5×5 vectors at a rate of 6 bits per vector [33]. Because we are using a sliding block VQ so that every pixel appears in 25 different vectors, the effective bit rate is 6 bits per pixel. The VQ was trained on the first frame of the motion sequence and tested on the next 20 images of the sequence. The result was compared to the case in which the second directional derivative edge detector was applied directly to the original test image (with no encoding). The comparison showed that all dominant edges in the original image are detected in the compressed image at this rate and, in fact, the algorithm gave fewer false positive edges by rejecting many high frequency low contrast texture edges. At the same time, it preserved the low frequency high contrast edges. Recent extensions to this work include a supervised classification scheme that uses human input to reclassify the output of the VQ-based edge detector; this is similar to using VQ for classification as described in Section 6.

8 Volume Rendering

3-D scalar fields arise in many applications. Empiric measurements of sonic waveforms or tomography radiation are processed to obtain density samples of a solid over a 3-D volume. In other cases, such as the stress distribution over a mechanical part or the pressure distribution within a fluid reservoir, the 3-D scalar fields are calculated numerically using finite difference or element techniques. The data set typically represents volume elements (voxel) samples of some scalar field f that are available on a cubical grid. The goal of visualizing the data is to understand the spatial distribution of the field values over the domain on which the function is defined. This can be done with volume rendering, which generates a 2-D image from the 3-D data set. The volume renderer computes an image by assuming a translucent material model and a lighting model and then rendering from a given point of view. The volume renderer forms an image in two stages: *shading* and *ray tracing.*

Shading requires the assumption of a material model that assigns a color, c, and an opacity, α, to each point in the volume. In a simple material model, color and opacity are user-defined functions of the scalar field value alone. In a more complicated implementation, the volume is modeled as a composition of one or more materials [13], and either the material composition of each voxel is provided directly or classification is used to estimate these percentages from the original data. In this model, values of c and α are assigned to each material and are calculated from the material composition in each voxel. In addition, the surface physics within the volume are approximated by assigning to each material a density characteristic, ρ. A surface is considered to occur within the volume whenever two or more materials with different ρ values meet. The strength of the surface is proportional to both the magnitude of the difference in ρ and the sharpness of the transition from one material to the other. The surface normal vector (n_x, n_y, n_z) can be required by the volume renderer so that the color value can incorporate directional shading and the gradient magnitude $\| \bigtriangledown f \|$ can influence the opacity level.

The ray tracer integrates the color and opacity values along each of a set of rays passing from a chosen viewpoint though the volume. It resamples the c and α values along each ray and composites the samples. The computation consists of incrementing the position, resampling c and α with a trilinear interpolation from the surrounding voxel values, and blending the sample into the pixel of the final 2-D image.

Such a volume rendering system typically processes the data in raw, uncompressed form. For large data sets, the storage used by these methods is very high and rendering speed is generally slow. For example, even if only a single 8-bit value of the scalar field f is stored for each voxel, the complete original (or decompressed) file will amount to 134 Mbytes for a $512 \times 512 \times 512$ voxel array. Depending on the complexity of the material and lighting models, and depending on whether the user wants to render the same material model from different viewpoints without having to recompute values, the file may be expanded to include the various derived quantities, yielding as many as seven values for each voxel: $\{f, n_x, n_y, n_z, \| \bigtriangledown f \|, c, \alpha\}$.

Combined VQ and Volume Rendering

Ning *et al.* showed VQ to be a particularly useful technique for compressing volumetric data [49]. In addition to providing a respectable amount of compression, VQ meets two criteria specifically important for pipelining it with volume rendering. VQ provides fast (table lookup) decoding so that rendering is not slowed by the decompression. VQ also allows random access to voxel values, which is advantageous because voxel access patterns are viewpoint dependent. Capitalizing on these advantages, Ning *et al.* [49] implemented a system in the form of Figure 6. A separate VQ codebook was designed for each volume to be compressed. In this case, the codebook must be stored in addition to the list of indices and the compression ratio cited must include the codebook. When the volume is visualized, the renderer performs the shading and ray tracing steps directly on the decompressed data as they arrive. The algorithm was tested on a fluid flow data set. Only the scalar field values were stored for each voxel. This combined system provided excellent image quality with 17:1 compression and only a modest 5% increase in rendering time. Recreating the entire decompressed file was not necessary, and thus the 17:1 storage savings were retained throughout the entire processing operation.

In further work [48, 50], Ning *et al.* showed that the shading and ray tracing steps could be accelerated by performing precomputations on the codebook before decoding. The system differs slightly from the model in Figure 8, in which the processing on the codebook is only done only once at codebook generation time. In volume rendering, because the user typically wants to select which colors and opacities correspond to the different features in the volume, c and α cannot be calculated at the time of codebook generation but must wait for the user to specify the correspondence immediately prior to decoding. At this point, the rendering can be sped up by performing the c and α calculations on only the codebook rather than by decompressing the full volume and calculating the values for each voxel in the decompressed volume. The ray tracing step can also be accelerated. Each codeword in the codebook can be ray traced from the user-specified viewpoint to form a set of projected bit maps. This step involves trilinear interpolating of the c and α values and combining them for the samples within each codeword. When the volume is decompressed, the entire volume can be ray traced by stepping from block to block and compositing the samples from the appropriate precomputed bit maps. Compared to the standard ray tracer for full volumes, time is saved in two ways: Fewer steps are taken because the steps are on a block basis rather than on a sample basis, and each step is less costly because the trilinear interpolation has already been done. The algorithm was tested on a data set of computerized tomographic scans and on an air jet study. At a compression ratio of 6:1, the time savings for the shading step was approximately 1000:1, and the speed-up for ray tracing was approximately 10:1.

9 Halftoning

Halftoning is the process of rendering grayscale (usually 8-bit) images to print them on binary (black and white) devices such as laser printers. Although the pixels in a halftoned image must be either black or white, the illusion of continuous shades of gray can be created by appropriate choices of the percentage and patterns of black pixels in each region of the image. The data rate of an uncompressed halftoned image is 1 bit per pixel which is already reduced relative to the rate of the original grayscale image. However, large halftoned images still require several megabytes of data to be transmitted to the printer. Data compression can be used to reduce transmission delays and printer memory requirements.

Ordered dither [65, 46] is a simple halftoning technique that operates independently on blocks. In Eq. 3, X is a 4×4 dither matrix for dithering 8-bit grayscale images [46]. The entire grayscale image is scanned in a nonoverlapping manner with the dithering matrix, and each grayscale pixel is compared to the appropriate threshold value in the matrix. If a pixel's intensity is above the threshold, it is set to white; otherwise, it is set to black. Thus the decision for each pixel depends on its position in the block as well as its original grayscale intensity value. The quality of the resulting halftoned image depends on the size and entries of the dithering matrix.

$$X = \begin{pmatrix} 8 & 136 & 40 & 168 \\ 200 & 72 & 232 & 104 \\ 56 & 184 & 24 & 152 \\ 248 & 120 & 216 & 88 \end{pmatrix} \tag{3}$$

Because each block is processed independently, ordered dither can easily be implemented with VQ if the dithering matrix and the VQ block are the same size (or the VQ block size is an integer multiple of the dithering matrix size). One applies the dithering matrix to each vector in the codebook to form linked VQ codebooks, as in Figure 8. The main codebook contains grayscale vectors, and the processed codebook contains binary vectors.

Error diffusion [18] is a neighborhood halftoning process. It is more complex than ordered dither but usually leads to higher halftoned image quality when the printing resolution is low (300 dpi and below). The error between the input grayscale pixel and its binary output is spread out over neighboring pixels by adding to the input a weighted combination of output errors from pixels above and to the left of the input. The updated input is compared to a fixed threshold to make the black or white decision. Combining error diffusion with VQ is challenging because it is a neighborhood process; however, it can lead to a significant decrease in the computational complexity. We are still exploring this problem.

Vander Kam *et al.* present an algorithm for a joint vector quantizer and halftoner design [34]. A frequency-weighted distortion measure based on the human visual response is used both to design the VQ and to select the binary reproduction for an input grayscale vector. Full search VQ is used to select the binary vector that measures a low frequency-weighted distortion compared to the input grayscale block. The halftoning algorithm is block-based (like ordered dither) so that it is easily compatible with VQ but is more complex and higher quality than ordered dither. The compressed halftoned image quality is then improved by including contextual information to allow more output reproductions. Vander Kam *et al.* found that their joint VQ/halftoner gave better results than either first compressing and then halftoning or vice versa (for their halftoning algorithm). In later work, they further improved the system's performance by replacing the full search VQ with an entropy-constrained VQ [5]. They obtained good image quality at bit rates as low as 0.1 bpp [35].

10 Synthetic Aperture Radar Reconstruction

Synthetic aperture radar (SAR) is used to obtain high resolution images of the earth and other planetary bodies from aircraft and spacecraft [11]. The system illuminates the target with microwaves, which are relatively unaffected by cloud cover and precipitation, so illumination by the sun is not required for proper exposure. The returning radar signals are processed to determine the reflectivity of the target at various range locations (perpendicular to direction of flight) and azimuth locations (along direction of flight). SAR uses these consecutive radar pulses at incremental antenna positions along the flight path to synthetically model an extremely long antenna aperture. Doing so improves the spatial resolution in the azimuth direction. The reflectivity of each point in the reconstructed image is obtained by correlating the raw data returns with a spatially dependent 2-D reference function, which performs a focusing operation (also called range/azimuth pulse compression). The results can be displayed as an image with intensity corresponding to reflectivity. Because the computational demands of processing raw SAR data are high, the radar returns are usually digitally sampled and transmitted from the airborne source to ground processing facilities. The high data rates required for transmission make compression desirable. Traditionally, the compressed data are decompressed before correlation with the location-dependent reference function for reconstruction.

Combined VQ and SAR Reconstruction

Vector quantization can be combined with the subsequent image processing task of reconstruction to significantly reduce both the bandwidth requirements and the computational demands of image reconstruction at the cost of higher memory requirements and somewhat degraded image quality [2, 60]. To reduce the reconstruction complexity, the reference function can be considered to be spatially invariant so that a single function can be used to reconstruct multiple points in the final image. The VQ codewords can be correlated with this spatially invariant reference function to form preprocessed codewords. This allows direct SAR reconstruction of the image from vector quantized blocks. Reconstruction requires only summation of the appropriately shifted preprocessed codewords. Eliminating the need for real-time multiplications significantly speeds up the reconstruction process. This method is similar to the model of Figure 8, in which the VQ decoder can put out either codewords representing the raw data or processed image segments corresponding to the reconstructed image.

The method has two main drawbacks. First, because the correlation step produces preprocessed codewords of significantly larger dimension than their unprocessed counterparts, significantly more memory is required. The memory requirements can be reduced somewhat by using a related technique that performs the range and azimuth correlation separably using two codebooks (for 1-D vectors in the range and azimuth directions) processed with corresponding 1-D reference functions [2, 60]. Second, image quality is degraded because of the approximation of the location-dependent reference function by a spatially invariant one. Inaccuracy in this focusing operation blurs the images. Using the spatially varying functions in a combined VQ/reconstruction system would require cross-correlating each of the codewords with each of the possible functions and storing all the results at the decoder; the memory requirements for this are prohibitive.

Although this algorithm was demonstrated on the SAR application, it is applicable to many other processing algorithms such as computing the discrete Fourier transform [61].

11 Related Work in Speech Processing

Vector quantization has been extensively used for speech compression, but it can also be used for other speech processing tasks including speech and isolated word recognition, speaker recognition and verification, and noise suppression. The ideas involved are related to the image processing examples. Abut devoted an entire section in his IEEE VQ reprint collection [1] to the use of VQ for speech processing.

VQ has found wide use in speech recognition systems as a front end, but it can also be used as a pattern matching technique to perform all or part of the recognition itself. Isolated utterance recognition can be performed by constructing a separate VQ codebook for each word in the vocabulary. In the recognition phase, average distortions between inputs and the various codebooks are computed and the utterance corresponding to the codebook with the smallest distortion is chosen. Such codebooks can also be used as a preprocessor to a word recognition system by eliminating those word candidates with high distortion [39, 19]. Performance can be improved by adding temporal information to the distortion measure used to select the VQ codewords [55, 31]. The modified distortion penalizes the selection of codewords unlikely in the given context, and improves the overall performance.

In other speech recognition systems, VQ has been combined with Hidden Markov Models (HMM). In the SPHINX system at Carnegie Mellon University [40], for example, VQ is used to discretize continuous data. The SPHINX system uses a size 256 codebook of 12-dimensional linear predictive coding cepstrum coefficients, and the VQ codewords are the symbols used in the system's HMMs. When an input sequence is presented, the cepstral coefficients are vector quantized, and the probability of the VQ codeword sequence given each HMM in the system is calculated. The utterance represented by the most likely HMM is selected. The SPHINX system recorded speaker-independent accuracy rates of 94% on a large vocabulary. For a review of the use of HMMs in speech recognition systems, see [56].

In recent related work, Pook and Ballard [58] have used VQ for HMMs that are applied to robot sensor data. The VQ is used to classify four different types of teleoperated robot manipulations. The classification is used both to filter and symbolize sensor data for later recognition schemes and to capture the salient characteristics of the sensor data so that they may later be mapped to autonomous control space. VQ is used on spectral coefficients for an HMM in a robotics system for learning human skill, by Yang, Xu, and Chen [66].

VQ can also be used for speaker recognition using a small VQ codebook consisting of highly representative speaker-specific feature vectors [29, 41, 63]. To differentiate between two speakers, training utterances from the two speakers are used to train two separate codebooks. Some clusters may overlap from one codebook to the other. A set of feature vectors is generated from an unknown speaker and encoded with each codebook. The speaker's identity is considered to be verified if the cumulative distortion between the input and a codebook is less than some preset threshold. Finally, VQ can be used as part of a noise reduction algorithm which maps noisy speech features into clean ones [16, 23, 15].

12 Conclusion

Our goal has been to describe the fundamental ideas of vector quantization and to survey ways in which design algorithms based on clustering and classification trees can be modified to incorporate image processing into VQ. We have provided examples of some of these methods including combining compression with histogram equalization, classification, edge detection, halftoning, and reconstruction. The goals of these algorithms are to reduce the complexity of the processing that follows the decompression step or to provide better overall quality by jointly optimizing the two operations instead of cascading them independently. Some of these techniques are recent in origin and the results are preliminary, but they suggest that the inherent enhancement and classification capabilities of clustering and classification trees can yield compression algorithms that perform a variety of signal processing functions.

Acknowledgements

The authors gratefully acknowledge the financial support of NIH Grant No. CA49697-02, NSF Grant Nos. MIP-9110508 and MIP-9016974, and of ESL, Incorporated for the research that led to many of the algorithms reported here. We also thank Professor Richard Olshen, Rick Vander Kam, M. Y. Jaisimha, and Jill Goldschneider for their helpful cooperation and contributions to this work.

References

[1] H. Abut, editor. *Vector Quantization.* IEEE Reprint Collection. IEEE Press, Piscataway, NJ, May 1990.

[2] D. V. Arnold, D. M. Chabries, and P. L. Jackson. Reduction of SAR digital processing time using vector quantization. In *Proceedings of 1988 IEEE National Radar Conference*, pages 134–139, Ann Arbor, Michigan, April 1988.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[4] D. Burton. Acoustic transient classification of passive sonar signals by using vector quantization. In *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 1493–6, Toronto, Ontario, Canada, 1991. IEEE.

[5] P. A. Chou, T. Lookabaugh, and R. M. Gray. Entropy-constrained vector quantization. *IEEE Trans. Acoust. Speech Signal Process.*, 37:31–42, Jan. 1989.

[6] 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.

[7] P. C. Cosman, E. A. Riskin, and R. M. Gray. Combined vector quantization and adaptive histogram equalization. In *Proceedings Medical Imaging VI*, Newport Beach, CA, Feb 1992. SPIE.

[8] P. C. Cosman, E. A. Riskin, and R. M. Gray. Combining vector quantization and histogram equalization. *Information Processing and Management*, 28(6):681–686, Nov-Dec 1992.

[9] P.C. Cosman, C. Tseng, R.A. Olshen, K.C.P. Li, and R. M. Gray. Predicting perceptual quality from SNR in compressed medical images. In *SID '93 Digest of Technical Papers*, pages 309–312, Seattle, Wa, May 1993. Society for Information Display.

[10] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inform. Theory*, 13:21–27, 1967.

[11] J. C. Curlander and R. N. McDonough. *Synthetic aperture radar: systems and signal processing.* Wiley series in remote sensing. John Wiley & Sons, Inc., New York, 1991.

[12] P. A. Devijver and J. Kittler. On the edited nearest neighbor rule. In *Proc. of 5th Int. Conf. on Pattern Recongition*, pages 72–80, 1980.

[13] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics*, 22(4):65–74, Aug. 1988. Proceedings of SIGGRAPH '88.

[14] A. Duchon and S. Katagiri. A minimum-distortion segmentation/LVQ hybrid algorithm for speech recognition. *Journal of the Acoustical Society of Japan (E)*, 14(1):37–42, 1993.

[15] Y. Ephraim. Statistical-model-based speech enhancement systems. *Proc. IEEE*, 80(10):1526–1555, October 1992.

[16] Y. Ephraim and R. M. Gray. A unified approach for encoding clean and noisy sources by means of waveform and autoregressive model vector quantization. *IEEE Trans. Inform. Theory*, 34:826–834, July 1988.

[17] E. Fix and J.L. Hodges, Jr. Discriminatory analysis, nonparametric discrimination, consistency properties. Project 21-49-004, Report No. 4, USAF School of Aviation Medicine, Randolph Field, Texas, Feb. 1951.

[18] R. Floyd and L. Steinberg. An adaptive algorithm for spatial grey scale. *SID Int. Sym. Digest of Tech. Papers*, pages 36–37, 1975.

[19] S. Furui. A VQ-based preprocessor using cepstral dynamic features for speaker-independent large vocabulary word recognition. *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-36:980–987, 1988.

[20] G. W. Gates. The reduced nearest neighbor rule. *IEEE Trans. Inform. Theory*, 18:431–433, 1972.

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

[22] J. Ghosh, S. Chakravarthy, Y. Shin, C.-C. Chu, L. Deuser, S. Beck, R. Still, and J. Whiteley. Adaptive kernel classifiers for short-duration oceanic signals. In *Proceedings IEEE Conference on Neural Networks for Ocean Engineering*, pages 41–8, 1991.

[23] H. Gish, Y. Chow, and J. R. Rohlicek. Probabilistic vector mapping of noisy speech parameters for HMM word spotting. In *Proceedings ICASSP*, volume 1, pages 117–120, Albuquerque, NM, April 1990.

[24] R. M. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, April 1984.

[25] R. M. Haralick. Digital step edges from zero crossings of second directional derivatives. *IEEE Trans. Pattern Anal. and Mach. Int.*, 6:58–68, Jan. 1984.

[26] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision.* Addison-Wesley, 1991.

[27] R.O. Harger. Object detection in clutter with learning maps. In *Proceedings of the SPIE - The International Society for Optical Engineering*, volume 1630, pages 76–86, 1992.

[28] P. E. Hart. The condensed nearest neighbor rule. *IEEE Trans. Inform. Theory*, 14:515–516, 1968.

[29] E. H. Helms. *Speaker recognition using linear prediction vector codebooks.* PhD thesis, Southern Methodist University, 1988.

[30] E. E. Hilbert. Cluster compression algorithm: a joint clustering/data compression concept. Publication 77-43, Jet Propulsion Lab, Pasadena, CA, Dec. 1977.

[31] S.-S. Huang and R. M. Gray. Spellmode recognition based on vector quantization. *Speech Communication*, 7:41–53, 1988.

[32] R. Hummel. Image enhancement by histogram transformation. *Computer graphics and image processing*, 6(2):184–195, April 1977.

[33] M. Y. Jaisimha, E. A. Riskin, and R. M. Haralick. Fast facet edge detection in image sequences using vector quantization. In *Proceedings ICASSP*, volume 3, pages 441–444, March 1992.

[34] R. A. Vander Kam, P. A. Chou, E. A. Riskin, and R. M. Gray. An algorithm for joint vector quantizer and halftoner design. In *Proceedings ICASSP*, volume 3, pages 497–500, March 1992.

[35] R.A. Vander Kam, P.A. Chou, and R. M. Gray. Combined halftoning and entropy-constrained vector quantization. In *SID '93 Digest of Technical Papers*, Seattle, Wa, May 1993. Society for Information Display.

[36] T. Kohonen. An introduction to neural computing. *Neural Networks*, 1:3–16, 1988.

[37] T. Kohonen. *Self-organization and associative memory.* Springer-Verlag, Berlin, third edition, 1989.

[38] T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural networks: benchmarking studies. In *IEEE International Conference on Neural Networks*, pages I–61–68, July 1988.

[39] G. E. Kopec and M. A. Bush. Network-based isolated digit recognition using vector quantization. *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-33:850–867, 1985.

[40] Kai-Fu Lee. *Automatic Speech Recognition: The Development of the SPHINX System.* Kluwer Academic Publishers, Boston, MA, 1989.

[41] K.-P. Li and E. H. Wrench, Jr. An approach to text-independent speaker recognition with short utterances. *Proceedings ICASSP*, 2:555–558, 1983.

[42] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Proc. IEEE*, 73(11):1551–1587, Nov. 1985.

[43] G.F. McLean. Texture classification using transform vector quantization. In *Proceedings of the SPIE - The International Society for Optical Engineering: Visual Communications and Image Processing*, volume 1360, pages 1332–43, Lausanne, Switzerland, 1990.

[44] J.J. Merelo, M.A. Andrade, C. Urena, A. Prieto, and F. Moran. Application of vector quantization algorithms to protein classification and secondary structure computation. In A. Prieto, editor, *Artificial Neural Networks. International Workshop IWANN '91 Proceedings*, pages 415–21. Springer-Verlag, 1991.

[45] B.K. Natarajan. Filtering random noise via data compression. In J.A. Storer and M. Cohn, editors, *Proceedings of the 1993 IEEE Data Compression Conference (DCC)*, pages 60–69, Snowbird, Utah, March 1993. IEEE Computer Society Press.

[46] A. N. Netravali and B. G. Haskell. *Digital Pictures: Representation and Compression.* Plenum Press, New York, 1988.

[47] E.K. Neumann, D.A. Wheeler, A.S. Bernstein, J.W. Burnside, and J.C. Hall. Artificial neural network classification of drosophila courtship song mutants. *Biological Cybernetics*, 66(6):485–96, 1992.

[48] P. Ning. *Applications of Data Compression to 3-D Scalar Field Visualization.* Ph.D. Dissertation, Stanford University, 1993.

[49] P. Ning and L. Hesselink. Vector quantization for volume rendering. In *Proc. ACM Workshop on Volume Visualization*, pages 69–74, Boston, MA, Nov. 1992.

[50] P. Ning and L. Hesselink. Fast volume rendering of compressed data. In *Visualization 1993*, pages 11–18. IEEE, Oct. 1993.

[51] K.L. Oehler, P.C. Cosman, R.M. Gray, and J. May. Classification using vector quantization. In *Proc. Twenty-Fifth Annual Asilomar Conference on Signals, Systems, and Computers*, pages 439–445, Pacific Grove, CA, Nov. 1991.

[52] K.L. Oehler and R.M. Gray. Combining image classification and image compression using vector quantization. In J.A. Storer and M. Cohn, editors, *Proceedings of the 1993 IEEE Data Compression Conference (DCC)*, pages 2–11, Snowbird, Utah, March 1993. IEEE Computer Society Press.

[53] R. A. Olshen, P. C. Cosman, C. Tseng, C. Davidson, L. Moses, R. M. Gray, and C. Bergin. Evaluating compressed medical images. In *Proceedings COMCON III*, pages 830–840, Victoria, B.C., Canada, Oct. 1991.

[54] L.M. Owsley and L.E. Atlas. Ordered vector quantization for neural network pattern recognition. In *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, 1993. to appear.

[55] K. C. Pan, F. K. Soong, and L. R. Rabiner. A vector quantization based preprocessor for speaker-independent isolated word recognition. *IEEE Trans. Acoust. Speech Signal Process.*, 33(6):546–560, June 1985.

[56] J. Piccone. Continuous speech recognition using Hidden Markov Models. *IEEE ASSP Magazine*, 7(3):26–41, July 1990.

[57] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. H. Romeny, J. B. Zimmerman, and K. Zuiderveld. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 39:355–368, 1987.

[58] P. Pook and D. H. Ballard. Recognizing teleoperated manipulations. In *Proceedings IEEE Robotics and Automation*, volume 2, pages 578–585, Atlanta, GA, May 1993.

[59] M. Rabbani and P. W. Jones. *Digital Image Compression Techniques*, volume TT7 of *Tutorial Texts in Optical Engineering*. SPIE Optical Engineering Press, Bellingham, WA, 1991.

[60] C. J. Read, D. V. Arnold, D. M. Chabries, P. L. Jackson, and R. W. Christiansen. Synthetic aperture radar image formation from compressed data using a new computation technique. *IEEE Aerospace and Electronic Systems Magazine*, 3(10):3–10, October 1988.

[61] C. J. Read, D. M. Chabries, R. W. Christiansen, and J. K. Flanagan. A method for computing the DFT of vector quantized data. In *Proceedings ICASSP*, pages 1015–1018, Glasgow, Scotland, May 1989.

[62] E. A. Riskin and R. M. Gray. A greedy tree growing algorithm for the design of variable rate vector quantizers. *IEEE Trans. Signal Process.*, 39:2500–2507, Nov. 1991.

[63] F. K. Soong, A. E. Rosenberg, L. R. Rabiner, and B.-H. Juang. A vector quantization approach to speaker recognition. *AT&T Tech. J.*, 66:14–26, 1987.

[64] C. J. Stone. Consistent nonparametric regression. *Annals of Statistics*, 5:595–645, 1977.

[65] R. Ulichney. *Digital Halftoning*. MIT Press, Cambridge, MA, 1987.

[66] J. Yang, Y. Xu, and C. S. Chen. Hidden Markov Model approach to skill learning and its application to telerobotics. In *Proceedings IEEE Robotics and Automation*, volume 1, pages 396–402, Atlanta, GA, May 1993.