Browse Level Compression of AVIRIS Data Using Vector Quantization on a Massively Parallel Machine

M. Manohar
Universities Space Research Association
NASA Goddard Space Flight Center
Greenbelt, Maryland

James C. Tilton
NASA Goddard Space Flight Center
Greenbelt, Maryland

Abstract. This paper describes and evaluates two Vector Quantization (VQ) approaches for compressing high spectral resolution AVIRIS (Airborne Visible/Infrared Imaging Spectrometer) image data by a factor of over 22. This highly compressed data is useful for "browsing" through several data sets in order to select the best data sets for a particular application, or for preliminary analysis. The two VQ approaches differ in the codebook generation methods, which are Linde-Buzo-Gray (LBG) in one case and a Self Organizing Feature Map (SOFM) based on neural models in the other. Both spectral and spatial correlations have been exploited to improve on coding efficiency. The codebook generation and coding portions of these VQ algorithms have been implemented on a massively parallel SIMD machine, the MasPar MP-1. However, the reconstruction of the compressed data is a simple table lookup which is efficiently performed on a sequential machine. The results are evaluated for accuracy by comparing the results from detecting minerals with the SPAM (Spectral Analysis Manager) software package in the original and reconstructed data sets.

I. Introduction

Each scene from the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) consists of over 140 megabytes of data. This large amount of data leads to significant difficulties in data archival, access and dissemination of AVIRIS data. One area that data compression can be of immediate assistance is in the access and dissemination of browse quality versions of AVIRIS data for the selection and preliminary analysis of data from the AVIRIS data archive.

AVIRIS data contains a great deal of redundancy that data compression can exploit to reduce data values by as much as 20 to 50 times. When compressed with appropriate techniques, this highly compressed AVIRIS browse data appears visually very similar to the original data, and still retains much of its scientific information content, making possible a preliminary analysis of the data. Of course, the final analysis will require the original data, or data reconstructed from a lossless compression of the data.

In earlier experiments [1], browse data with a compression factor of 18 was produced from AVHRR (Advanced Very High Resolution Radiometer). This data was processed to obtain Sea Surface Temperature (SST) maps. (SST maps are a standard product generated from AVHRR data and ancillary data. These maps are regularly used by scientists who study the Earth's oceans and climate.) For the best compression method, the mean error in the SST maps was only 0.5°C.

Unlike AVHRR data with 5 spectral bands, the AVIRIS data has as many as 224 spectral bands. In performing VQ on this data, it must be judiciously grouped into vectors so both spatial and spectral correlations are utilized. A small data cube of 128 rows by 256 columns of 32 spectral bands was used in these experiments. Training and coding were made computationally efficient by (i) dividing the spectrum into four parts of eight bands each, and (ii) parallel implementation on MasPar MP-1 massively parallel computer.

This paper describes and evaluates two Vector Quantization (VQ) compression approaches for producing browse versions of AVIRIS data. In the first VQ approach the optimal codebook is generated by well known Linde-Buzo-Gray (LBG) algorithm that is based on k-means clustering algorithm. The second employs a more innovative technique called Self Organizing Feature Maps (SOFM) based on the neural models due to Kohonen. The compression

approaches are evaluated by compression ratio (or data rate), computational requirements, and the image and analysis errors introduced due to lossy compression. The SPAM (Spectral Analysis Manager) package was used to produce mineral maps from the original and reconstructed data sets. These mineral maps were then analyzed for error.

II. Compression Based on Vector Quantization

Vector Quantization (VQ) is motivated by rate-distortion theory [2] which states that better performance can be achieved by coding vectors rather than scalars. To effectively exploit spatial and spectral correlations among image pixels in VQ, the image is partitioned into two dimensional cells of fixed size in all spectral bands. The pixels in each $k = w_r x w_c$ of a rectangular cell of each spectral band are scanned in raster scan order and are taken to be a one-dimensional vector of size k in band sequential (BSQ) order. As the image is scanned, each vector is compared to standard vectors in a codebook. The address of the standard vector in the codebook that most closely matches the input vector is transmitted or stored as the code for that cell. Compression is obtained because the address bits are much fewer than the number of bits required for representing each vector, and because the number of elements transmitted or stored is less than the original number of pixels by a factor equal to the vector size. Thus, the smaller the codebook size (m), and the larger the vector size (k), the more efficient is the compression (fewer bits are required per pixel).

This relationship of codebook size and vector size to compression efficiency is illustrated by the expression for the number of bits required per pixel, or compression rate (r).

$$r = 1/k * \log_2 m \text{ bits/pixel} \tag{1}$$

A particular rate can be accomplished by several possible pairs of m and k. Selecting small values of the codebook size, m, is not very effective because r changes as logarithm of m. Increasing the vector size is more effective since the rate, r, increases linearly with 1/k. Thus, the change in rate, r, is more rapid with k than m, suggesting that large vector dimensions should be used.

In designing a VQ compressor, one must also consider the distortion introduced by the VQ process. The most commonly used distortion measure is the mean squared error between input image and reconstructed image. Although this measure does not directly show how close visually the reconstructed image is to the original image, it is used because it is mathematically tractable, and does give some sense of difference between the two images.

Here we use the average mean squared error as a distortion measure for quantifying the performance of the encoder. If Xi, and χ_i are the input vector and reproduced vector respectively, the average distortion, d, is given by

$$d = 1/n \sum (X_i - \chi_i)^2 \tag{2}$$

where n is the number of vectors from the input image. For given rate, r (bits/pixel), and given distortion measure, d, the vector dimension, k (pixels), and the codebook size, m, can be optimally determined [3].

Another common distortion measure frequently used is the signal to noise ratio (or signal to quantization noise ratio [4]), defined as:

$$SNR = 10 \log_{10} E(\|x\|^2) / d \tag{3}$$

where $E(\|x\|^2)$ is the average signal energy, and d is as given in Eq. 2.

As noted earlier, large vector dimensions should be used to achieve low compression rates. Large vector dimensions, however, are difficult to handle even in parallel machines. For a given

codebook size, the computational requirements are directly proportional to the vector dimension. Since larger vector dimensions are also recommended to exploit the image spatial correlations effectively, trade-offs must be made between computational requirements and compression performance (rate and distortion).

The greatest drawback of VQ is that it is computationally very demanding. The bulk of the time is consumed exhaustively searching the codebook to find the closest match to a given input vector from among the code words in the candidate set. Different methods have been developed to speed the search by structuring the codebook, at the cost of reducing the compression factor. Parallelization is an obvious solution to this search problem in both codebook generation as well as in data encoding.

The VQ can be conveniently described by assuming an array of cells configured in two-dimensional lattice. This array holds the representative samples to the input set, also called the codebook. The most important function of the VQ is to learn from the input samples the most optimal codebook containing fewer number of samples than the input set. Let the input event space be represented by

$$\xi = \{X_1, X_2, .., X_i, ..., X_n\} \text{ where } X_i = (x_{i1}, x_{i2}, ..., x_{ik})^t$$

and the output space or codebook by

$$\psi = \{Y_1, Y_2, ..., Y_j, ..., Y_m\} \text{ where } Y_j = (y_{j1}, y_{j2}, ..., y_{jk})^t$$

In all practical situations the codebook size, m << n (the number of vectors extracted from the input image). The compression ratio is given by $(n*b)/\log_2 m$, where b is number of bits per vector.

In the training process ψ is learned from ξ such that the many to one mapping of all the vectors $X_i \in \xi$ onto $Y_j \in \psi$ results in minimum error. The learning starts from some initial state, $Y_j(0)$. After X_i is compared with all samples in ψ, the most closely matching vector, Y_c, is called the *winner*. The *winner* Y_c is updated such that X_i and Y_c match even more closely in subsequent iterations. This is competitive learning. Using the Euclidean distance measure for matching, the identification of the winner can be mathematically represented as:

$$\| X_i - Y_c \| = \min_{\forall j \in m} \| X_i - Y_j \| \tag{4}$$

Y_c is the closest match to the input sample X_i from mean squared error point of view.

Next, Y_c is updated according to the following rule in order to make the match even closer between successive iterations, t and t+1:

$$\begin{aligned} Y_j(t+1) &= Y_j(t) + \alpha(t)(X_i - Y_j(t)) && \text{for } j = c \\ Y_j(t+1) &= Y_j(t) && \text{for } j \neq c \end{aligned} \tag{5}$$

where $\alpha(t)$ is a monotonically decreasing sequence of scalar values in t, making the error minimization converge locally. When the updating rule of Eq. 5 is applied, the error minimizes asymptotically. The input sample X_i is selected randomly from the event set ξ.

A. VQ using LBG algorithm

The first of the two approaches to VQ explored here uses the well known Linde-Buzo-Gray (LBG) algorithm, based on the k-means clustering algorithm, to train the codebook[4]. In this approach, the vectors extracted from the input multispectral images are clustered into a finite number of classes (equal to the size of the code book) in k-dimensional space, where k is the dimension of the vector.

In the LBG algorithm, the initial state, $Y_i \in \psi$ for all i $\in$ (1..m), is selected randomly from input sample set, ξ. Next, the competitive learning process is carried out by first accumulating the subsets of training samples from ξ that map onto each location of the codebook, ψ, based on Eq. 4. Then the updating operation is performed in a single step by using the mean of the samples that mapped onto each of the codebook locations. The accumulating and updating steps are then carried out iteratively until convergence [4].

The LBG algorithm maps input samples onto one of m different codebook vectors. The mapping is done regardless of the index value, i, of the codebook vector, so that the m cells act independently. Thus, the input vectors are mapped to the index i in haphazard order. The impact of this in pattern recognition accuracy is debatable, but it certainly slows the classification speed of incoming features, because of the exhaustive search required to determine the matching pattern in the index range. This haphazard mapping is also contrary to the way in which biological cells or neurons interact spatially to provide a topologically ordered map of input features. Kohonen's work on Self Organizing Feature Maps (SOFM) achieves this topological property of the neural systems by modifying the update rule in Eq. 5, as discussed in the following section.

B. VQ using SOFM algorithm

In the SOFM training process, the values $Y_i \in \psi$, are initialized with random values. The values $Y_i \in \psi$ are then treated as a two-dimensional neural lattice space ($m = m_r x m_c$), and updated not only at the location of the winner, Y_c, but also in a spatial neighborhood, N_c. The function, $\alpha(t)$ in Eq. 5 is replaced by $h_{cj}(t)\varepsilon(t)$, which not only decreases with time, but also becomes more peaked spatially with time. The function $h_{cj}(t)$ initially has a broad spatial spread, which becomes more localized at the winner location, Y_c, in each ensuing iteration. Thus, the cell values, which initially have random value, organize themselves to match the events of the input space, while retaining the distance relations of the higher dimensional input space in two-dimensional lattice locations of the cells [5-7].

III. Massively Parallel Implementation of VQ

VQ image compression consists of a training phase, coding phase, and decoding phase. In our case, the training phase consists of generating a codebook using either the LBG or SOFM algorithm. The coding phase consists primarily of searching through the codebook for the best match. The decoding phase consists of a straightforward table lookup from the codebook, producing a reconstructed image.

The training phase is very computationally very intensive. For every sample presented, the entire codebook must be searched to decide the winner. Then the codebook must be updated in the winner's location and in the neighborhood of winner. With a naive codebook structure, the algorithm makes an exhaustive search of the codebook as many times as the network is presented with training samples.

On sequential machines, search times may be improved by organizing the entries in efficient tree structures, such as pruned tree K-D tree structures. These tree structures reduce the search from O(m) to O(log m), provided the tree is nearly balanced, where m is codebook size. These speedups are obtained by effectively increasing codebook size, thus increasing storage requirements, and decreasing the compression factor. However, on parallel machines, it is straightforward to improve the search time without increasing the codebook size.

Being readily available in our facility, the MasPar model MP-1 was used as our implementation platform. The MasPar MP-1 is a fine grained SIMD machine with 8192 4-bit processors organized in a 128 row by 64 column array. The codebook size was set to be less than or equal to 128x64, so each entry of codebook maps on single processor, minimizing the data movement among the processors. VQ compression using LBG and SOFM was

implemented on the MasPar MP-1 using a C-like language called MPL (MasPar Language). In MPL, "plural" variables are arrays equal in size to the processor array, which have a value in each processor's local memory.

A. Training

1. Initializing the Codebook. The first steps in the training process are to select the codebook and vector sizes, and to initialize the codebook. The codebook size, m, and vector size, k, are normally selected based on the compression factor and distortion specifications for a particular application. However, as we noted earlier, the change in compression rate, r, is more rapid with k than m. Therefore, larger-sized codebooks provide improvement in signal to noise ratio without appreciably affecting compression ratio (doubling the codebook size reduces the compressor rate by only one bit/pixel). Further, in the MasPar implementation, the computational requirement, T, is unaffected by codebook size as long as it is less than the PE array size. However, when the codebook size is more than PE array size, the computational requirements go up as follows:

$$T = T_0 \lceil m/nproc \rceil \qquad \text{for } m > nproc$$

where nproc is number of PE's in the array (size of PE array), and m is the size of codebook. T_0 is computational requirement for m = nproc.

Considering the above, the best choice for codebook size in the MasPar implementation is an integer multiple of the number of processors in the MasPar, nproc. Further adjustments in compressor rate can be best achieved by varying the vector size, k.

For LBG, the codebook is initialized by randomly selected vectors from the input image. The SOFM training algorithm uses random number generator for codebook initialization. For this purpose, a plural floating point array of random numbers between 0 and 1.0 is generated on MasPar in few machine cycles using an MPL built-in function called p_random.

2. Computing the winner for each sample. After the codebook is initialized, the input image pixel values are normalized with respect to the maximum pixel value, and the image is decomposed into a set of vectors at randomly chosen locations in the image. Each of these vectors are broadcast in turn to all PE's in the array, and the Euclidean distance between the input vector and codebook is computed at all locations. The location where the value is minimum can be found by using the reduction operator which finds the minimum value of the array.

3. Updating. For the LBG algorithm, the winner locations accumulate all the vectors during the iteration. At the end of the iteration the contents of the winner is updated to be the centroid of all the vectors. For SOFM, however, the updating is performed for every incoming vector in the neighborhood, N_c, of the winner.

Each cycle of computing the winners and updating the codebook is called an epoch. the codebook is updated until there is no appreciable change in the codebook from one epoch to the next. For most practical problems, for error less than 1%, these algorithms converge in 10-15 epoches.

B. Coding

After the training phase, the codebook is frozen and used for coding images of the class whose subset was used for generating the codebook. The coding consists of exhaustive searching through the codebook for every sample to be coded. Since the codebook is loaded into

the PE array's local memory, the computational requirement for exhaustive search is proportional to the size of the vector. Further, it is independent of the size of the codebook as long as the codebook size is less than or equal to the size of the PE array. In sequential coding this search would be proportional to the product of codebook size and vector size. Thus, the parallel version could be faster than the sequential search by a factor as much as the size of the codebook. The coding procedure is as follows.

1. Extract a vector of size k from the input image (in raster scan order),
2. Broadcast this vector into the local memory of all PE's,
3. Compute a distance measure (such as Euclidean distance) between the vector and the codebook at each PE,
4. Find the minimum of the Euclidean distance, and
5. Store or transmit the address bits of the PE where minimum value of the Euclidean measure is obtained.
6. Repeat 1-5 for until the entire image is scanned.

The coding procedure is basically the same as computing the winner in one epoch of the training phase. The difference is in the order in which the vectors are extracted from the image. In the coding phase, the data can be extracted in any convenient order. However, in the training phase, the extraction order should be random to avoid introducing an order dependent bias into the codebook.

C. Decoding

Decoding is reconstructing the image from the address bits of each vector. This is a table lookup process in which the codebook vector for each address is substituted for the image data values. This process is sequential and can be carried out on sequential machines more efficiently than parallel machines.

IV. Evaluation Scheme

The data compression approaches are compared here in terms of distortion measures for a given compression ratio, and in terms of an analysis application. The distortion measures employed are Mean Square Error (Eq. 2) and SNR (Eq. 3). The compression ration (CR) is the ratio of number of input bits to the number of output bits.

AVIRIS data sets are high spectral resolution images which are used for many applications such as geological, vegetation, and atmospheric studies. The Spectral Analysis Manager (SPAM) software package has been developed at Jet Propulsion Laboratory to analyze this and other imaging spectrometry data. It enables the user to store the spectral characteristics of land features obtained by field experiments and detect the pixels having similar characteristics in the image.

A. Spectral signature matching

SPAM's high speed spectral matching algorithm, *find*, employs binary encoding of the spectral characteristics of pixels or minerals (in mineral analysis application) to matching them to stored patterns [8]. Given a prototype spectrum, SPAM binary encodes each pixel in an image and decides whether or not the pixel belongs to a stored prototype class by comparing their amplitude and slope Hamming distances to user specified thresholds.

B. Mixture Component Analysis

The spectrum of an image pixel represents the total energy reflected by the atmosphere and earth's surface within the gathering instruments field of view. Ignoring the atmospheric effects, the pixel energy can be considered to be a weighted combination of the energy reflected by several surface components and is expressed as follows:

$$S = \sum_{k} C_k D S_k + A \qquad (6)$$

where C_k is a nonnegative constant proportional to the fractional surface composition for k^{th} mineral, D is a diagonal matrix whose entries are the product of the solar illumination and atmospheric transmissivity for each of the k minerals, S_k is the spectrum of the k^{th} mineral, and A is a constant vector. Given the values of the S and S_k, C_k can be estimated by a constrained least squares technique.

V. EXPERIMENTAL RESULTS

The original data set used this study consists of a data cube of 128 rows by 256 columns by 32 spectral bands from the April 13, 1989 Rogers Dry Lake data set. The 32 spectral bands correspond to band 172 (wavelength = 1.9343) thru 203 (wavelength = 2.2416) in the AVIRIS data format. The first six bands of the image data set is shown in Fig. 1(a). The input data cube was divided into 4 full image size parts, but with only 8 contiguous spectral bands. Separate codebooks were generated for each of these four subdata cubes. To exploit both spatial and spectral correlations, a 32 element vector was used for Vector Quantization, made up of a 2 by 2 spatial component and an 8 band spectral component.

Codebooks were generated using both LBG and SOFM algorithms on the MasPar, and the original data set was coded and further compressed with lossless coding. The compression results are shown in the Table I.

The compressed and reconstructed results and the error images from these techniques are shown in Fig. 1 (a-e). LBG technique performed better than SOFM in terms of mean squared distortion. This is also evident in Fig.'s 1(c) and 1(e). The coding time was about 30 seconds for the data size of 128x256x32. This corresponds roughly to a 0.3 megabit/second rate. Efforts are under way to improve the coding speed to 1 megabit/second.

A. Spectral signature matching

The test samples taken from the image data have spectral characteristics similar to some minerals stored in the library of SPAM. These minerals are sphalerite, colemanite, and brucite. We are not claiming that these minerals are present in this data set. However, if these minerals are detected in the compressed data similarly to how they were detected in the original data, then we infer that the compressed data worked for this application as well as original data. The results of the spectral matching are given in the Table II.

The spatial locations where these minerals are detected roughly are same. Sphalarite locations in the three images are shown in Fig. 2 (a-c) with yellow spots. It is very interesting to see that in all three mineral detections SOFM performs better than LBG algorithm in this application implying that MSE is not the appropriate evaluation criterion for this application.

B. Mixture Component Analysis

In this experiment a spectral plot at location pixel, line (26,17) is broken up into fractions of above three mineral spectral characteristics. The results are given in the Table III. The results of this analysis from SPAM are shown in Fig. 3 (a-c). From this, one can see that both LBG and SOFM algorithms have same composition at the above pixel location and they each differ very little from the original data.

VI. Conclusions

We have shown that hyperspectral data from instruments like AVIRIS can be compressed for archival and distribution over the computer network. In this work we have used two algorithms of vector quantization and obtained compression ratio of over 20 with minimal effects on the analysis of the data. This has been shown by objective evaluation of data applying the compressed as well original data to mineral studies in geology.

References

1. Tilton, J. C., Han, D. and Manohar, M., "Compression Experiments with AVHRR data", Proc.of Data Compression Conference, April 8-11, 1991, pp. 411-420.
2. Shannon, C. E., "A Mathematical Theory of Communication," Bell Systems Technical Journal, Vol. 27, pp. 379-423, 1948.
3. Lookabaugh, T. M. and Gray, R. M., "High Resolution Quantization Theory and the Vector Quantizer Advantage," IEEE Transactions on Information Theory, Vol. 35, No. 5., 1989.
4. Gray, R. M., "Vector Quantization," IEEE ASSP Magazine, pp. 4-28, April 1984.
5. Kohonen, T., "Clustering Taxonomy, and Topological Maps of Patterns," Proceedings of the International Conference on Pattern Recognition, pp. 114-128, 1982.
6. Kohonen, T., "Self-Organized Formation of Topologically Correct Feature Maps," Biological Cybernetics, Vol. 43, pp. 59-69, 1982.
7. Kohonen, T., "The Self-Organizing Map," Proceedings of the IEEE, Vol. 78, No. 9, pp. 1464-1480, 1990.
8. Mazer, A. S. et al., "Image Processing Software for Imaging Spectrometry Data Analysis," Remote Sensing of Environment, Vol. 24, pp. 201-210, 1988.

Table I. Comparison of LBG and SOFM coding performance.

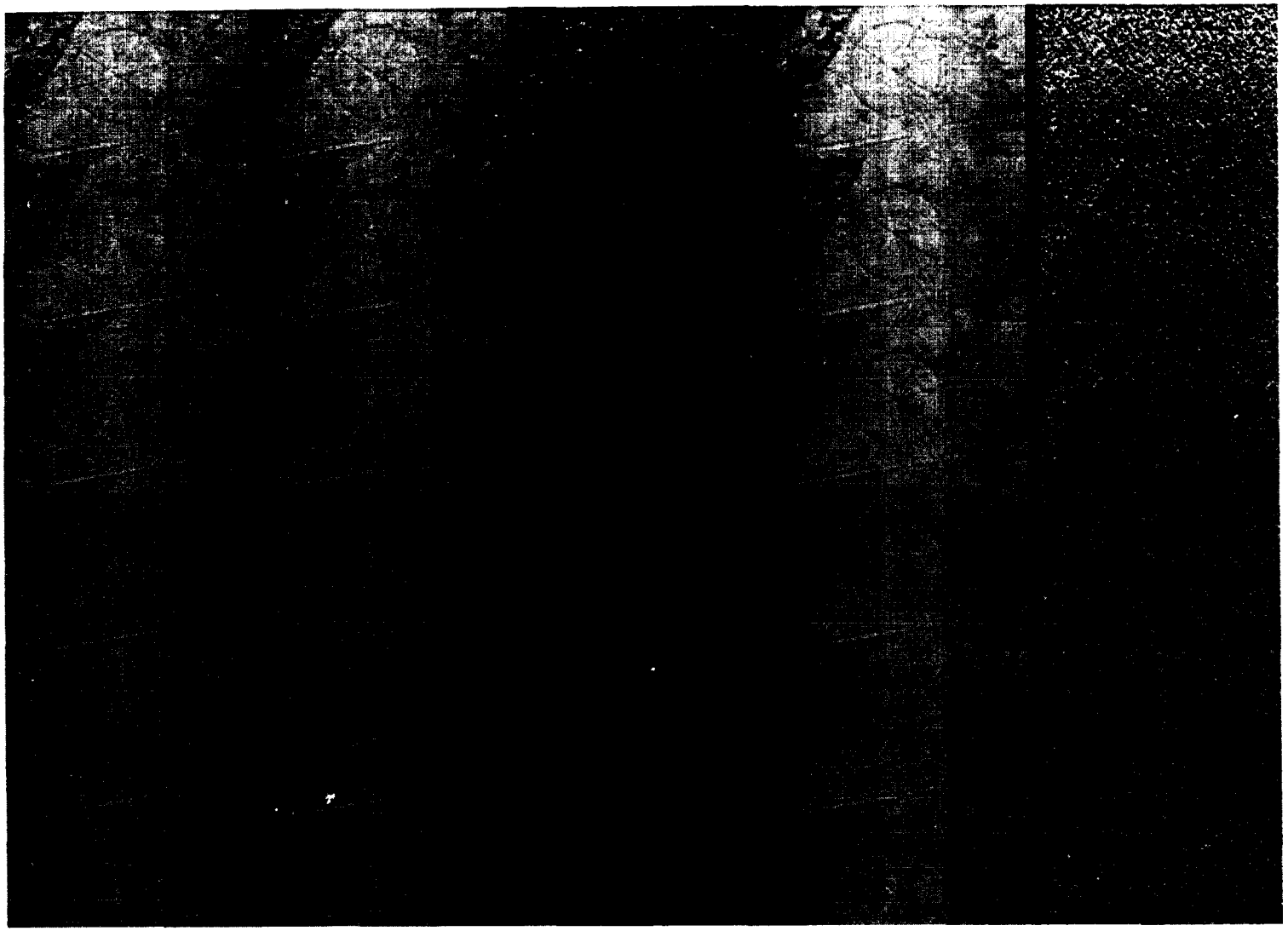
Method	CR	MSE	SNR (db)
LBG data	22.12	3.13	31.06
SOFM data	22.43	6.39	34.14

Table II. Comparison of reconstructed data from LBG and SOFM with original data in the detection of minerals.

Minerals	Error threshold	Original data	LBG data	SOFM data
Sphalerite.f	0	3636	3975	3666
Colemanite.m	1	3678	4013	3763
Brucite.c	1	106	70	112

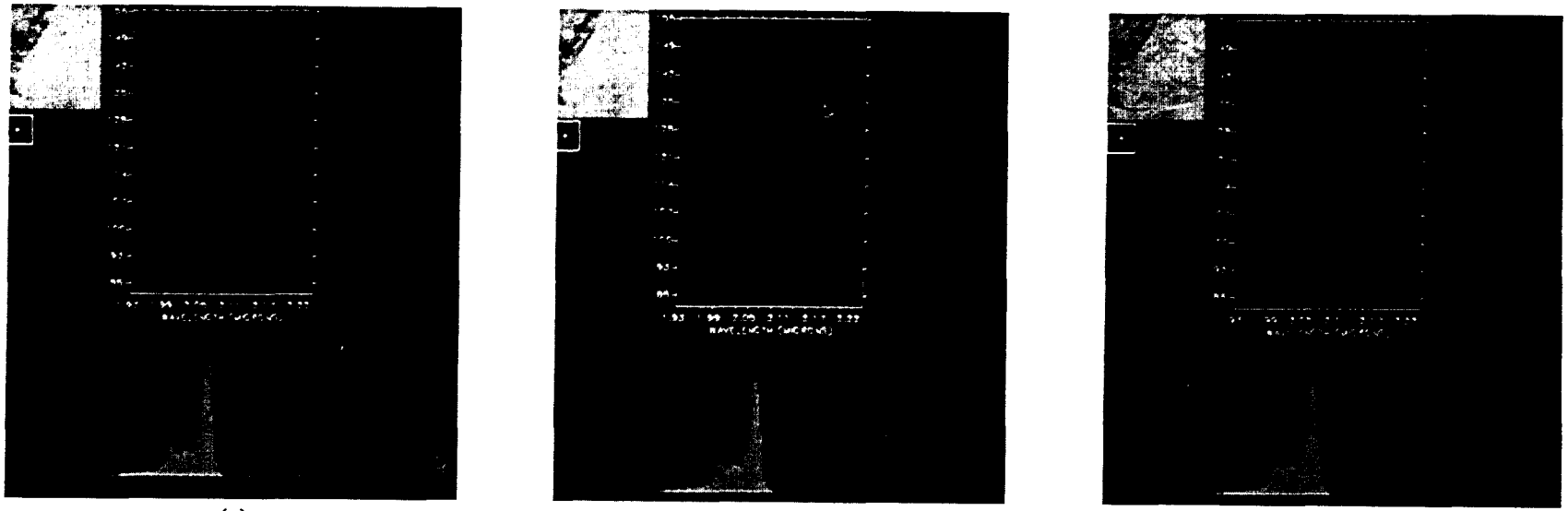
Table III. The mineral composition of a spectral sample taken at column 26 and row 17.

Data	Sphalerite	Colemanite	Brucite
Original	0.1718	0.0000	0.9428
LBG	0.1788	0.0000	0.9450
SOFM	0.1714	0.0000	0.9443



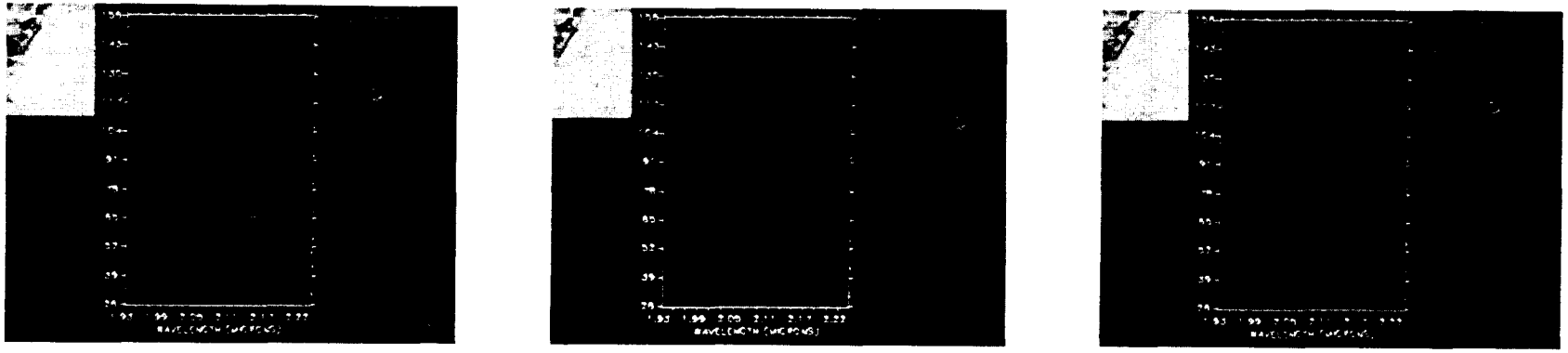
(a) (b) (c) (d) (e)

Figure 1. (a) Bands 172-177 of Rogers Dry Lake data, (b) reconstructed image from LBG coding, (c) error image between *a* and *b*, (d) reconstructed image from SOFM coding, and (e) error image between *a* and *d*.



(a) (b) (c)

Figure 2. Location of samples (yellow) that matched sphalerite with 0 threshold. (a) Location of matched 3636 samples in original data, (b) location of 3976 matched samples in LBG reconstructed data, and (c) location of 3666 samples matched in SOFM reconstructed data.



(a) (b) (c)

Figure 3. Mineral composition of sample at line 17 and column 26 using least square fitting. Mineral composition (a) for original data, (b) for LBG reconstructed data, and (c) for SOFM reconstructed data.