Lossless Image Compression

Article · March 2002

Source: CiteSeer

CITATIONS
13

READS
207

6 authors, including:

Baba Vemuri
University of Florida
300 PUBLICATIONS **10,144** CITATIONS
SEE PROFILE

Sartaj Sahni
University of Florida
463 PUBLICATIONS **15,219** CITATIONS
SEE PROFILE

Fen Chen
Apple Inc.
101 PUBLICATIONS **1,558** CITATIONS
SEE PROFILE

Christiana Leonard
University of Florida
161 PUBLICATIONS **12,362** CITATIONS
SEE PROFILE

LOSSLESS IMAGE COMPRESSION

B. C. Vemuri, S. Sahni, F. Chen, C. Kapoor, C. Leonard, and J. Fitzsimmons

Abstract

There are numerous applications of image processing, such as satellite imaging, medical imaging, and video where the image size or image stream size is too large and requires a large amount of storage space or high bandwidth for communication in its original form. Image compression techniques can be used effectively in such applications. Lossless (reversible) image compression techniques preserve the information so that exact reconstruction of the image is possible from the compressed data. In this paper we survey existing coding and lossless compression schemes and also provide an experimental evaluation of various state of the art lossless compression algorithms that have been reported in the literature. The experimental evaluation is done using a large number (45) of publicly available image data sets. In the case of coding algorithms, we present a comparison of the achieved compression for a variety of linearlization schemes applied to the image data sets.

Keywords

Lossless Compression, Coding, Linearization, Transform domain, Spatial domain.

I. Introduction

Image comression plays a very important role in applications like tele-videoconferencing, remote sensing, document and medical imaging, and facsimile transmission, which depend on the efficient manipulation, storage, and transmission of binary, gray scale, or color images.

Image compression techniques can be classified into two categories: lossless and lossy schemes. In lossless methods, the exact original data can be recovered, while in lossy schemes, only a close approximation of the original data can be obtained.

Image compression can be achieved via *coding methods*, *spatial domain compression*, *transform domain compression methods* or a combination of these methods. **Coding** methods are directly applied to the raw images treating them as a sequence of discrete numbers. Common coding methods include: Arithmetic, Huffman, Lempel-Ziv Welch (LZW), and Run-Length. **Spatial domain** methods, which are a combination of spatial domain algorithms and coding methods, not only operate directly on the gray values in an image but also try to eliminate the spatial redundancy. In **transform domain** compression, the image is represented using an appropriate basis set, and the goal is to obtain a sparse coefficient matrix. Discrete Cosine Transform (DCT) based compression and wavelet transform are two examples of transform domain methods.

The authors are with the Dept. of CISE, Univ. of Florida, Gainesville, Fl. 32611. This work was supported in part by the NIH under the grant R01LM05944-03.

Recently, there has been a growing interest in reversible/lossless image compression of medical images. The sheer amount of data involved in the fields of picture archival and communication systems (PACS) coupled with the fact that no loss of fidelity can be tolerated in compression and decompression in most medical applications, has motivated this interest.

There are numerous reversible image compression schemes in the literature, each has its own advantages and disadvantages.

This paper is organized as follows. In Section II we present performance measures for lossless compression. Section III describes different linearization schemes. These schemes can be used to convert a two-dimensional image into a one-dimensional grayscale stream. This conversion is necessary if we are to use standard coding schemes to compress images. Huffman coding, Lempel-Ziv coding, and arithmetic coding are discussed in Section IV. Various spatial domain compression methods including Variable Block Size Compression, JPEG's lossless compression and two context-based lossless compressions are presented in Section V. In Section VI we describe a transform domain compression scheme called the (S+P) transform which is based on the wavelet transform concept. Although experimental results are presented throughout the paper, Section VII summarizes the experiments with a comparison of the best compression methods from each category. This is followed by conclusions in Section VIII.

II. Performance Measurement of Lossless Image Compression

The performance of a lossless image compression algorithm can be specified in terms of compression efficiency and complexity. **Compression efficiency** is measured by the *compression ratio* or by the *bit rate*. *Compression ratio* is the ratio of the size of the original image to the size of the compressed image; and *bit rate* is the number of bits per pixel required by the compressed image. For example, a 256×256 8-bits per pixel image requires $256 * 256 * 8$ bits $= 65536 * 8$ bits $= 65{,}536$ bytes when stored in uncompressed form. If the compressed image requires 32768 bytes, then the compression ratio is $65536/32768 = 2.0$. Since the image has $256 * 256 = 65{,}536$ pixels, the compressed file needs $32768 * 8/65536 = 4$ bits per pixel, on average. Hence the bit rate is 4.

The compression ratio and bit rate are related. Let b be the number of bits per pixel of the uncompressed

image, CR the compression ratio, and BR the bit rate. The following equality is readily obtained $CR = b/BR$ The effectiveness of a compression method can be compared to the entropy of the source image. The **source entropy** is defined as the amount of information contained in a source. Suppose that the pixel gray values range from 0 to $M-1$. Let p_i be the probability (i.e., frequency divided by the total number of pixels) of the gray value i. The information content of the image is given by its entropy [3]:

$$H(I) = -\sum_{i=0}^{M-1} p_i \log p_i.$$

The units for entropy are bits per pixel. The effectiveness of a lossless compression method is measured by determining how closely its bit rate approximates the source entropy, as the source entropy is a lower bound on the bit rate any lossless compression method can achieve [3]. Therefore, if the source entropy of an image is 4 bits/pixel and our lossless compressor has a bit rate of 4 bits per pixel, our lossless compressor has done the best job possible.

The **complexity** of an image compression algorithm is measured by the number of arithmetic operations required to perform both the encoding and decoding processes. This is an important factor for applications involving online image compression and decompression where speed is crucial.

III. Linearlization Schemes

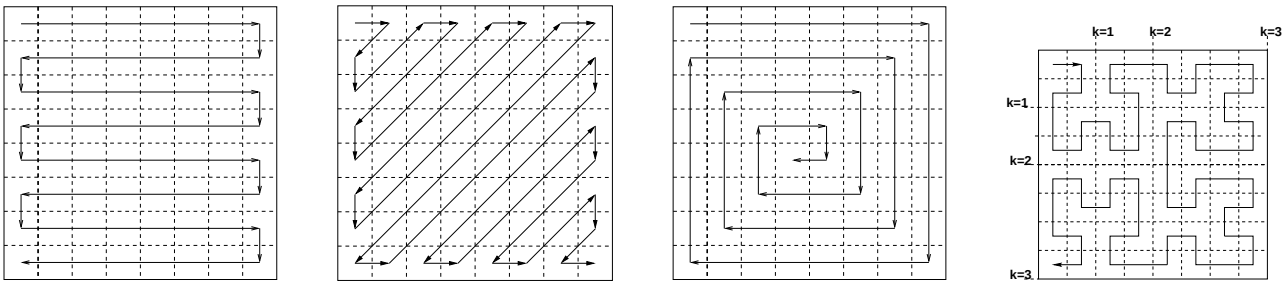
When coding schemes such as Huffman coding, arithmtic coding and LZW coding are used to compress a two-dimensional image, the image must first be converted into a one-dimensional sequence. This conversion is referred to as **linearization**. Coding schemes such as Huffman coding depend only on the frequency of occurrence of different gray values. Since linearization does not affect this frequency, coding schemes in this category are unaffected by the particular method used to linearize a two-dimensional image. On the other hand, coding schemes such as arithmetic coding and Liv-Zempel coding depend on the relative order of gray scale values and so are sensitive to the linearization method used.

Most (real) images have local and global redundancy. Local redundancy causes a given neighborhood in the image to exhibit coherence or correlation (referred to as smoothness of data). Some linearization schemes are more effective than others at (most closely) preserving the neighborhood relationship contained in the image. Therefore, these schemes are more effective in preserving the local redundancy of the image and are expected

to yield better compression when coupled with a coding scheme that can take advantage of local redundancy.

Some of the more popular linearization schemes are given below. Each scans the image pixels in some order to produce the one-dimensional sequence.

1. **Row-Major Scan**: The image is scanned row by row from top to bottom, and from left to right within each row.
2. **Column-Major Scan**: The image is scanned column by column from left to right, and from top to bottom within each column.
3. **Diagonal Scan**: The image is scanned along the antidiagonals (i.e., lines with constant row plus column value) beginning with the top-most antidiagonal. Each antidiagonal is scanned from the left bottom corner to the right top corner.
4. **Snake-like Row-Major Scan**: This is a variant of the row-major scan method described above. In this method, the image is scanned row by row from top to bottom, and the rows are alternately scanned from left to right and from right to left. The top-most row is scanned from left to right (as in Figure 1 (a)). Snake-like variants of column-major and diagonal scans can be defined in a similar manner (see Figure 1 (b)).



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

Fig. 1. (a)snake-like row major scan path (b)snake-like diagnal scan path (c) spiral scan path (d) Peano scan path

5. **Spiral Scan**: In this scan, the image is scanned from the outside to the inside, tracing out a spiral curve starting from the top left corner of the image and proceeding clockwise (see Figure 1 (c)).
6. **Peano-Hilbert Scan**: This scan method is due to Peano and Hilbert, and is best described recursively as in Figure 1 (d). This method requires the image to be a $2^k \times 2^k$ image. When k is odd, the scan path starts at the leftmost pixel of the first row and ends at the leftmost pixel of the bottom row. When k is

even, the path starts at the leftmost pixel of the first row and ends at the right-most pixel of this row. In a Peano-Hilbert scan, the image is scanned quadrant by quadrant. The scan path for a $2^k \times 2^k$ image for $k = 1$, 2, and 3 is shown in Figure 1 (d).

Table I gives the compression ratios achieved by gzip on a sample suite of 10 images. Different linearization schemes result in different compression ratios when gzip is used for compression. For example, the compression ratios for the *man* image range from 1.3 to 1.4, and those for *brain1* range from 1.63 to 1.75. Although no single linearization method provides the highest compression for all images, the Peano-Hilbert scan did the best most often.

TABLE I
COMPARISON OF DIFFERENT LINEALIZATION SCHEMES USING GZIP

Image	row major	diagonal	snake	spiral	peano
lenna	1.18	1.18	1.17	1.19	1.20
man	1.39	1.31	1.39	1.40	1.39
chall	1.43	1.39	1.43	1.43	1.47
coral	1.32	1.26	1.32	1.28	1.29
shuttle	1.44	1.43	1.44	1.42	1.47
sphere	1.30	1.29	1.30	1.33	1.31
brain1	1.75	1.63	1.75	1.69	1.67
slice15	1.73	1.63	1.73	1.68	1.66
head	1.80	1.75	1.80	1.78	1.77
sag1	1.68	1.66	1.69	1.74	1.71

Table II gives the compression ratios achieved by the Unix compression utility compress on our sample suite of 10 images. Compress did not provide better compression than provided by gzip on any of our test images. Consequently, we shall not report further results using compress.

IV. CODING METHODS

Coding schemes that are designed for sequential data compression can be used to compress two-dimensional images provided they are linearized. In this section, we first discribe coding schemes for sequential data and then compare their performance in the context of compressing two-dimensional images.

TABLE II
COMPARISON OF DIFFERENT LINEALIZATION SCHEMES USING COMPRESS COMMAND

Image	row major	diagonal	snake	spiral	peano
lenna	1.07	1.08	1.07	1.11	1.10
man	1.30	1.23	1.31	1.32	1.32
chall	1.32	1.29	1.33	1.37	1.38
coral	1.21	1.15	1.21	1.21	1.20
shuttle	1.37	1.29	1.37	1.33	1.36
sphere	1.17	1.17	1.18	1.22	1.21
brain1	1.70	1.57	1.70	1.65	1.63
slice15	1.67	1.56	1.67	1.64	1.62
head	1.79	1.71	1.78	1.77	1.75
sag1	1.58	1.55	1.59	1.61	1.60

A. Huffman Coding

In Huffman coding each symbol of the uncompressed data is replaced by a code [12]. The symbol codes are of variable length and symbols that occur more frequently in the uncompressed data have codes of smaller length than symbols that occur less frequently. The symbol codes satisfy a **prefix property**–no code is a proper prefix of another code. By encoding frequently occurring symbols by short codes and infrequently occurring symbols by longer codes, an overall reduction in the space needed by the data is obtained. For the case of images, individual pixel values are considered to represent individual symbols and the symbol set consists of all gray values. For an 8 bits per pixel image, the size of the symbol set is $2^8 = 256$.

Assume that the data to be compressed has five symbols: {s1, s2, s3, s4, s5} with the following probabilities: {0.1, 0.3, 0.25, 0.15, 0.2}. The Huffman codes for n symbols can be computed in $O(n \log n)$ time using a greedy algorithm (see [14], for example). Using the greedy code construction algorithm, we obtain the following symbol codes: {100, 11, 01, 101, 00}. Notice that no code is a prefix of another code and that the symbol s2 with higher probability has the 2-bit code 11 while the symbol s1 with lower probability has the 3-bit code 100.

In Huffman coding. we use a fixed preset coding table which is based on an estimated frequency distribution of the values to be coded. In adaptive Huffman coding the coding table, is generated individually (adaptively) from the symbol probabilities (or frequencies) for the data to be coded. Although adaptive coding results in higher compression ratios, all the data to be compressed must be available before compression can start

because the symbol frequencies have to be determined before we can determine the symbol codes.

To decompress the data, we need the code table. Using this, the original uncompressed data can be reconstructed by examining the compressed data from begin to end.

Table IV-A gives the compression ratios obtained on our 10 image suite. The results are inferior to those obtained using gzip on all 10 images. This is not surprising as gzip is essentially LZW coding followed by Huffman coding. For the head image, Huffman coding is almost as good as gzip; suggesting that on this image gzip gets most of its compression from its postprocessing Huffman coding step.

Image	lenna	man	chall	coral	shuttle	sphere	brain1	slice15	head	sag1
CR	1.09	1.09	1.17	1.16	1.18	1.25	1.60	1.57	1.79	1.60

B. LZW Coding

Lempel and Ziv [13] and Welch [15] have proposed an adaptive coding method that does not require all the data that is to be compressed to be available at the start. Rather, their technique generates codes as it examines the source file from begin to end. In Huffman coding a variable length code is constructed for each symbol in the source file. In Lempel-Ziv coding fixed-length codes are constructed, on the fly, for variable-length sequences of symbols.

Assume that the symbols which occur in the source file are a, b, c, and that the string *ababcabc* is to be compressed. First, we initialize a code-to-string dictionary to contain the single-symbol strings that are possible. For our example, these strings are a, b, and c. The string code is given by its position in the dictionary. So the code for a is 0, that for b is 1, and that for c is 2. Then starting from the first symbol in the file to be compressed, we find the longest prefix of the input which is in the dictionary. The longest prefix of our source file *ababcabc* that is in the dictionary is a. Its code 0 is output as part of the compressed file and the prefix plus the next input symbol entered into the dictionary. In our case, the string ab is entered into the dictionary. Its code is 3. The portion of the source file that remains to be compressed is *babcabc*. Next we find the longest prefix of this remaining file that is in the dictionary. The prefix is b. Its code 1 is output to the compressed file, and the prefix b and the next symbol a of the input entered into the dictionary as the string ba with code 4. Following this, we again determine the longest prefix of the remaining input file that

is in the dictionary. This time the longest prefix is string *ab*. The code 3 for *ab* is output and we put string *abc* (*c* is the symbol following the prefix) into the dictionary. The code for *abc* is 5. The next longest prefix found is *c*. Its code 2 is output and *ca* is entered into the dictionary. The remaining input string is *abc*. It is in the dictionary. So its code 5 is output. Therefore, the coding of the string *ababcabc* is 01325. Given the encoded string, we can decode it in the reverse way.

We do not report any experimental results for the LZW method as gzip is a combination of LZW and Huffman coding, and results for gzip were presented in Section III.

C. Arithmetic Coding

In this method, a message is coded as a subinterval of the interval [0, 1), where [x, y) denotes a half open interval, which includes x but excludes y [2]. There are two fundamental concepts in arithmetic coding: the probability of a symbol, and encoding interval range for a symbol. The occurrence probabilities of source symbols determine the compression efficiency as well as the interval ranges of source symbols for the encoding process. These interval ranges are contained within the interval from zero to one and determine the compression output.

We use a simple example to explain the process of arithmetic coding. Let's assume that the source symbols are {a, b} and the probabilities of these symbols are 0.4 and 0.6, respectively. Then, based on these probabilities, the interval [0, 1) can be divided into the two sub-intervals: [0, 0.4) and [0.4, 1). The range [0, 0.4) is used to further subdivide an interval when the symbol a is encountered in the input file and the range [0.4,1) is used when the symbol b is encountered.

To encode the string *baab* as a subinterval of [0, 1), we begin with [0, 1) as the **working subinterval** and use the encoding range of the first symbol b from the input string to narrow the working subinterval to the last 6/10 of the current working subinterval. Therefore, the interval [0, 1) is narrowed to [0.4, 1). Since the range of the second symbol a is [0, 0.4), the working subinterval [0.4, 1) is reduced to its first 4/10 portion to obtain the new working subinterval [0.4, 0.64). The third input symbol is a, and its encoding reduces the current interval [0.4, 0.64) to its first 4/10 portion, that is to [0.4, 0.496). Encoding the fourth symbol b produces the final interval [0.4576, 0.496). The input string *baab* is encoded as the interval [0.4576, 0.496).

From the symbol probabilities and the encoded interval, we can reconstruct the input file. For example, the interval [0.4576, 0.496) is a subinterval of bs interval [0.4, 1). So the first symbol of the source string is b and the working subinterval becomes [0.4, 1). Next we see that [0.4576, 0.496) is in the left 4/10 of the interval [0.4,1). So the next symbol in the source file is a and the working subinterval becomes [0.4, 0.64). Since [0.4576, 0.496) lies wholly in as portion of the working subinterval, the next source symbol is also a and the working subinterval becomes [0.4, 0.496). Now we see that [0.4576, 0.496) lies in the right 6/10 of the working subinterval. So the next source symbol is b and the working subinterval becomes [0.4576, 0.496). At this time decoding terminates because the working subinterval and the coded subinterval are the same.

If each source file is assumed to end with a unique symbol (such as end-of-file), the coded file can be represented by any number in the final working subinterval rather than by the endpoints of this interval.

Although our preceding description of arithmetic coding uses high-precision floating point arithmetic, the scheme can be adapted to use integers alone [2]. Furthermore the coding scheme can be made adaptive, that is the probablilities and hence subranges of [0, 1) associated with each symbol can be made to depend on the symbol frequencies in the source file that is being coded. For this we maintain the symbol frequencies as symbols are encountered during the coding process and adjust the ranges assigned to each symbol according to the current cummulative symbol frequencies. Since decoding generates the symbols in the same order as used in coding, the cummulative frequencies can be computed during decoding and the symbol ranges adjusted accordingly. The exact method used to adjust the symbol ranges is called a *model*.

Table III shows the compression ratios achieved by arithmetic coding. As was the case for gzip, the performance of arithmetic coding is sensitive to the linearization scheme used. Once again, no linearization scheme provided maximum compression on all images, though the Peano-Hilbert scan came close to doing this. Gzip provided more compression than did arithmetic coding on the first six images.

D. Run-Length Coding

In run-length coding we decompose the source file into segments of identical symbols, each segment is replaced by a pair of the form (symbol, number of occurrences). For example, the source file *aaaabaaabbb* is coded as (a, 4), (b, 1), (a, 3), (b, 3). This coding scheme works well when we have many long segments and

TABLE III
COMPARISON OF DIFFERENT LINEALIZATION SCHEMES USING ARITHMETIC CODING

Image	row major	diagonal	snake	spiral	peano
lenna	1.13	1.16	1.13	1.12	1.17
man	1.23	1.23	1.23	1.19	1.25
chall	1.33	1.30	1.33	1.23	1.33
coral	1.27	1.23	1.27	1.19	1.27
shuttle	1.32	1.35	1.32	1.25	1.38
sphere	1.29	1.29	1.29	1.29	1.30
brain1	1.71	1.71	1.71	1.77	1.73
slice15	1.70	1.69	1.70	1.79	1.72
head	1.93	1.91	1.93	2.01	1.94
sag1	1.95	1.98	1.97	2.07	2.02

works poorly when we have many short segments. Since our test images do not have long segments, we did not experiment with run-length coding. More details about this coding method can be found in [3].

V. Spatial Domain Algorithms

Spatial domain techniques involve methods for reducing the number of bits required to represent the information contained in the image by directly operating on the raw image. Such methods usually include two stages. The first stage involves the use of techniques such as image segmentation, subsampling and interpolation, etc. This is followed by a second stage involving efficient encoding of the results from the first stage. In the first stage, image segmentation schemes that have been used include partitioning the image into regular shapes (rectangles) or into irregular shapes. The former has the advantage that it takes less storage to code the shape [1] due to simplicity of the primitive used while the latter is comparitively storage intensive because of the generality and complexity of the shapes resulting from the segmentation algorithm. Simpler shape primivites require very few bits to code the shape but are restricted in their generality of representation and hence a large number of primitives may be required to represent the entire image. On the other hand, allowing generality in shape primitives would increase the complexity of coding the primitive itself even though, a small number of these may suffice to represent the entire image. We will not focus on the latter techniques since most of the existing techniques in the literature in this category are lossy compression algorithms [17]. In the following, we describe various state of the art image compression algorithms that primarily use spatial domain information either directly or in processed form (e.g., segmented data).

A. Variable Block Size Compression

In [1], Ranganathan *et al.* developed a lossless image compression algorithm which exploits local and global redundancy present in most images. Their algorithm segments the image into variable size blocks and encodes them based on the properties exhibited by the pixels within a block. This encoding is achieved via a run-length coding scheme when all the pixels within a block are of the same gray value and a base offset coding scheme when the degree of variation of gray value within a block is small. The encoding schemes in this case are exploiting the local redundancy present in the image. For blocks of size (2,2), a block matching scheme is used to determine if an identical block exists in the neighborhood of the reference block thereby exploiting the global redundancy present in the image. When the variation of the gray level values within a block exceeds a specific threshold, data in the block are transmitted as is. This algorithm performed better than the Huffman, Arithmetic, Lempel-Ziv, and the JPEG lossless schemes on all our test images.

B. Lossy+Residual Approaches

The lossy-residual approach includes many methods whose basic concept is to send a lossy image primarily, followed by the residual with respect to the original image. Using the lossy+residual, the original image can be reconstructed exactly.

The lossy image can be transmitted in many different forms. One way is to transmit it as the compressed data output of a lossy compression algorithm. There are numerous lossy compression algorithms that can be used to effectively compress images with high compression ratios. The reconstructed image in this case is an approximation of the original and can be used as an estimate of the original. If the estimate is a good approximation of the original, then the residual (difference between the original and reconstructed images) will contain only small values. This residual image can be linearized to a sequence and effectively compressed using schemes like arithmetic coding and Huffman coding. The main advantage of this kind of a lossless scheme is that it also exploits the high compression ratios obtained from lossy schemes.

The *lossless* JPEG compression standard is a simple version of lossy+residual approach. We will first describe the lossless JPEG compression standard, then give a more general approach based on the use of interpolation.

B.1 The JPEG Compression Standard

There are four modes of operation in JPEG namely, *sequential DCT based mode*, *sequential lossless mode*, *progressive DCT based mode*, and *hierarchical mode*. Even though the *JPEG Baseline* algorithm is the simplest and most widely used compression algorithm in the JPEG family, it is a lossy compression algorithm since quantization and finite precision in computation of DCT and LDCT introduce distortion. Since the focus of this section is on lossless compression algorithms, we will only be concerned with the JPEG standard of a predictive coding based lossless compression algorithm.

The JPEG standard specifies the following two coding methods for a lossless mode of operation.

- Lossless method with Huffman coding.
- Lossless method with arithmetic coding.

The JPEG lossless mode uses a predictive coding technique. The lossless mode in JPEG is fully independent of the tranform-based coding and instead applies a differential coding to form the residuals which are then coded using either the Huffman or arithmetic coding methods. The residual in the prediction is formed using previously encoded pixels in the current row and/or previous row of the image. The residual from the prediction at a pixel location $\mathbf{x}$ is defined as $R = P\mathbf{x} - \mathbf{x}$, where $P\mathbf{x}$ is the predicted value and can be any of the values defined in the table shown in figure IV. The pixel values at locations a, b and c (fig. 2) are available at the encoder as well as the decoder prior to processing of the location $\mathbf{x}$. The chosen prediction function is encoded into the header of the compressed stream and thus is easily readable at the decoder end. When prediction methods (4) through (7) are used, values in the first row are predicted using method (1) and those in the first column are predicted using method (2). For more details on the JPEG lossless mode of operation, we refer the reader to [20].

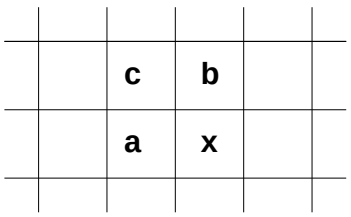


Fig. 2. Image samples used in JPEG lossless prediction

The JPEG lossless coding algorithm has several advantages. It involves very simple computations and hence

TABLE IV
PREDICTORS FOR JPEG LOSSLESS

Selection-value	Prediction equation
0	no prediction
1	Px=a
2	Px=b
3	Px=c
4	Px=a+b-c
5	Px=a+(b-c)/2
6	Px=b+(a-c)/2
7	Px=(a+b)/2

is easy to implement. It can easily and efficiently be mapped to hardware because of the simplicity of the algorithm.

C. Context-based Compression Algorithms

Recently, context-based modeling has become a popular technique in image compression algorithms. The concept of context was first used in text compression wherein the context of a symbol was defined as the N symbols preceding it. During the modeling stage of text compression, the model assigns a probability to a symbol based not only on the frequency of occurrence of this symbol but also on the frequency counts of this symbol's context. In the coding stage, the symbols are actually coded based on their probabilities conditioned on their contexts.

In general terms, context modeling in data compression involves techniques which can uniquely describe the source string. According to [28] it consists of two parts: the *structure*, which is the set of occurrences of the symbols and their contexts, and the *parameters*, which are the probabilities assigned to each occurrence of the symbols. If we consider the input stream of the images as the source strings, the *structure* is intended to capture the redundancies in the entire set of source strings under consideration, while the *parameters* are made to fit for each individual string separately. A *structure function* is defined as (see [28]): $g : Y^* \rightarrow I, I = \{0, ..., K-1\}$, which partitions the set of all strings Y^* into K equivalent classes or "contexts". The structure function makes it possible to construct a model for a set of strings that are similar in some respects. Each string will get its own representation of the context defined by the common structure function and probabilities tailored

to that string. Thus the probability of the string $y^i = x_1x_2...x_i$ with next symbol x_{i+1} will be conditioned as $P(x_{i+1}|g(y^i))$, and ideally, the code length contributed by symbol x_{i+1} is $-logP(x_{i+1}|g(y^i))$. The goal is to maximize the probability assigned to the entire sequence $P(y^n) = \prod_{i=0}^{n-1} p(x_{i+1}|y^i)$ which then minimizes the total code length according to Shannon's theory[3].

However, to find an optimum structure function is an undecidable problem [28]. When context modeling is applied to the compression of gray-scale images, it faces the "context dilution" problem, in which the frequency counts of each symbol must be spread over several contexts, thus affecting the accuracy of the corresponding estimates. Hence, techniques of defining the model's structure function or the context that gives a good classification of the data are the primary research focus among current context-based compression algorithms. In LOCO-I [27], context definition is based on the local image gradients allowing one to capture the level of activity surrounding a pixel. In [29] the context is defined in terms of the prediction errors incurred in previous encodings. In CALIC [26], the context is a combination of the local gradient and the previous prediction errors. Generally, such context-based algorithms include the following steps:

1. A prediction step, in which a value $\hat{x}_{i+1}$ is computed for the next pixel x_{i+1} based on a subset of the available past sequence y^i.
2. The determination of a *context* or *structure function* in which x_{i+1} occurs, based on the past sequence y^i.
3. A probabilistic model for the *prediction residual* $e_{i+1} = x_{i+1} - \hat{x}_{i+1}$, conditioned on the context of x_{i+1}.

To avoid context dilution, CALIC and LOCO-I use context selection and quantization to reduce the number of free variables. In CALIC, contexts are also used to estimate prediction errors to improve the prediction accuracy. Currently, due to LOCO-I's superior placement in the compression/complexity trade-off curve (that is, it has a very low complexity and good compression performance), it has been adopted as part of the new JPEG standard called JPEG-LS. While CALIC has a relatively higher complexity than LOCO-I, it is the best available compression scheme reported in the literature. We provide a brief description of CALIC and LOCO-I in the following subsections.

C.1 CALIC: A Context Based Adaptive Lossless Image Coder

CALIC is a context based adaptive lossless image coder that was developed by Wu and Memon [26]. Basically, it is a lossy+residual approach. In the first stage, the algorithm uses a gradient-based non-linear prediction to get a lossy image and a residual image. In the second stage, it uses arithmetic coding to encode the residuals based on the conditional probability of the symbols in different contexts. CALIC also has a mechanism to automatically trigger a binary mode which is used to code either uniform or binary subimages or both. CALIC is currently the best lossless image compression scheme reported in the literature. In the following we describe the main stages of the CALIC algorithm.

- **Gradient-based Prediction**

  CALIC uses a nonlinear predictor that adapts itself to the image gradient near the predicted pixel. This predictor improves the precision of traditional DPCM predictors, particularly in areas of sharp edges.

  Firstly, the horizontal and vertical gradients at the current pixel x (shown in Figure 3) are estimated as :

$$d_h = |w - ww| + |n - nw| + |ne - n|$$

$$d_v = |w - nw| + |n - nn| + |ne - nne|$$

  Based on d_h and d_v, a gradient-adjusted prediction is defined as:

$$\dot{I}[i,j] = \begin{cases} w & (d_v - d_h > 80)\{sharp\ horizontal\ edge\} \\ n & (d_h - d_v > 80)\{sharp\ vertical\ edge\} \\ 3/4w + 1/4n + (ne - nw)/8 & (d_v - d_h > 32)\{horizontal\ edge\} \\ 5/8w + 3/8n + 3(ne - nw)/16 & (d_v - d_h > 8)\{weak\ horizontal\ edge\} \\ 3/4n + 1/4w + (ne - nw)/8 & (d_h - d_v > 32)\{vertical\ edge\} \\ 5/8n + 3/8w + 3(ne - nw)/16 & (d_h - d_v > 8)\{weak\ vertical\ edge\} \\ (w + n)/2 + (ne - nw)/4 & otherwise \end{cases}$$

  Where: $n = I[i-1, j]$, $w = I[i, j-1]$, $ne = I[i-1, j+1]$, $nne = I[i-2, j+1]$, $nw = I[i-1, j-1]$, $nn = I[i-2, j]$, $ww = I[i, j-2]$. The positions of these pixels are shown in Fig. 3.

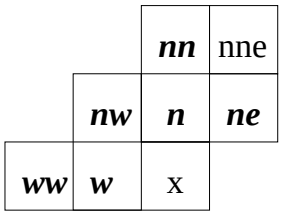


Fig. 3. Template for CALIC

However, this prediction is still not precise enough to completely remove the spatial redundancy in the image. CALIC uses an error energy to estimate the error for subsequent refinement of the prediction. The error energy is defined as: $\Delta = d_h + d_v + 2 * |ew|$, which is the least square estimator of $e = I - \dot{I}$ (ew is the prediction error at the preceding pixel $I[i, j-1]$). By conditioning the error distribution on Δ, the prediction errors are separated into classes of different variances. Thus, entropy coding of errors using estimated conditional probability $p(e|\Delta)$ improves coding efficiency over using $p(e)$. To achieve coding efficiency, Δ is further quantized into L levels (denoted as $Q(\Delta)$)

- **Context Modeling**

Since image gradients are inadequate for characterizing some of the more complex relationships between the predicted pixel $I[i, j]$ and its neighbors, context modeling of the prediction error is used in CALIC. Context modeling of the prediction error $e = I - \dot{I}$ exploits higher-order structures such as texture patterns in the image for further compression gains. The texture pattern is defined as $P = \{x_0, ..., x_6, x_7\} = \{n, w, nw, ne, nn, ww, 2n - nn, 2w - ww\}$. For space and time efficiency, P is quantized to an 8-bit binary number $B = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$ where:

$$b_k = \begin{cases} 0 & if\ x_k \geq \dot{I}[i, j] \\ 1 & if\ x_k < \dot{I}[i, j] \end{cases} \quad 0 \leq k < 8$$

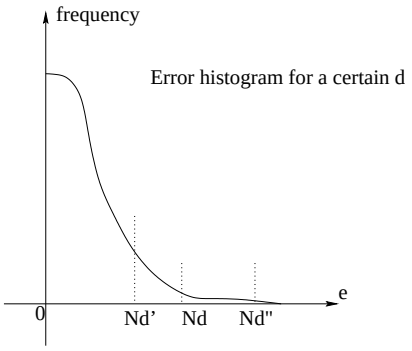
We can now condition the prediction errors by the context $C(Q(\Delta), B)$. Note that $C(Q(\Delta), B)$ denotes a tuple of two items $(Q(\Delta), B)$. In this paper, we follow the notation $C(Q(\Delta), B)$ used in [26].

Basically, the definition of the context (tuple $C(Q(\Delta), B)$) is the structure function defined in [28]. However, in CALIC, the context $C(Q(\Delta), B)$ is not only used to condition or partition the prediction errors in the encoding phase but also used to estimate the prediction errors. In CALIC, the prediction errors are estimated as the conditional expectations $E\{e|C(Q(\Delta), B)\}$ using the corresponding sample means $\overline{e}(Q(\Delta), B)$ for different contexts. Computing $\overline{e}(Q(\Delta), B)$ involves accumulating the errors of processed pixels with the same context and maintaining a count on the occurrence of this context. Thus the context model $\overline{e}(Q(\Delta), B)$ is updated on the fly, a process involving the training of the context model to obtain accurate estimates of the prediction errors under the context $C(Q(\Delta), B)$. After the error estimation, error feedback can be applied on the prediction error: $\ddot{I} = \dot{I} + E\{e|C(Q(\Delta), B))\}$ to get a smaller error

$e' = I - \ddot{I}$,

- **Encoding**

The next stage is encoding the errors. CALIC uses an adaptive arithmetic coding technique based on the conditional probability of the adjusted errors: $p(e|Q(\Delta))$. The errors are classified into several classes according to $Q(\Delta)$ and each class forms a model in the arithmetic coding as described in Section IV-C.



(a)

Fig. 4. Error histogram, d represents δ

If we look at the histograms of the errors in each class, for example Fig. 4, we see that large errors occur with diminishing frequency, but still occupy positions in the symbol table. This not only wastes memory space for storing frequency tables in arithmetic coding, but also introduces many unnecessary symbols. In the case of arithmetic coding, particularly for sharp conditional error probabilities $p(e|\delta)$ for small $\delta = Q(\Delta)$, we have to assign 1 to the frequency counts of the zero frequency symbols to safeguard the occurrence of an event of small probability. The forced counting of 1 in the error histograms can significantly distort the underlying error statistics, and thus reduce coding efficiency. This problem is called the zero frequency problem in the arithmetic coding.

CALIC uses a simple and effective way to solve this problem: it truncates the tails of the error histogram and uses an escape character to code the errors beyond the truncated code range. Specifically, it limits the size of each conditional error histogram to some value N_δ, $1 \leq \delta < L$, such that a large majority of errors to be coded under the coding context δ fall into the range of the N_δ largest entries of the $\delta - th$ error histogram. Essentially the symbol N_δ for each coding context δ serves as an escape symbol. Symbols greater than or equal to N_i are encoded using more than one codeword. The value of each N_δ here is

crucial because too small a value causes a large number of escape symbols to be coded, whereas too large a value does not cure the zero frequency problem. CALIC's authors empirically set the value for N_δ, based on experiments with an image set.

- **Binary Mode**

  CALIC can dynamically trigger a binary mode when the neighbors contain very few gray levels. In this case, it encodes the pixel value instead of prediction errors. During the scanning process, the algorithm checks the neighbors of the pixel x: $\{n, w, nw, ww, nn, ne\}$ (see Fig. 3) and if they contain only two gray values, CALIC goes into the binary mode and encodes a 0 or a 1 when x belongs to one of these two gray values. It encodes a 2 as an escape character when x is neither of the two gray values and then switches back to the continuous-tone mode to encode the prediction error of x. Binary mode is effective for binary images and very smooth images.

C.2 LOCO-I: A Low Complexity, Context-based Lossless Image Compression Algorithm

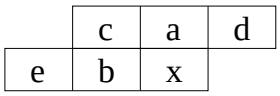


Fig. 5. a causal template for LOCO-I

The prediction and modeling units in LOCO-I are based on the causal template depicted in Fig. 5, where x denotes the current pixel, and a, b, c, d, and e are neighboring pixels in the relative positions shown in Fig. 5. LOCO-I uses the following predictor :

$$\hat{x}_{i+1} = \begin{cases} min(a,b) & if\ c \geq max(a,b) \\ max(a,b) & if\ c \leq min(a,b) \\ a+b-c & otherwise \end{cases}$$

to predict the pixel x_{i+1}'s value. The context of LOCO-I is built out of the differences $g_1 = d - a, g_2 = a - c, g_3 = c - b$ and $g_4 = b - e$, which represent the local gradient. To avoid "context dilution", parameter reduction is made by quantizing g_1, g_2 and g_3, into a small number or approximately equiprobable regions. Thus the prediction errors can be conditioned by the context and regions as: $Prob\{e_{i+1}|C_i = [q_1, q_2, q_3, q_4]\}$ where C_i represents the quantized context quartet and q_1, q_2, q_3 and q_4 are quantized differences corresponding, respectively, to g_1, g_2, g_3 and g_4. Finally, the prediction residuals are encoded by the Golomb-Rice codes [30]. LOCO-I achieves a good compression ratio with a low time complexity. However, CALIC is able to obtain

slightly better compression at the expense of additional compute time.

VI. Transform Domain Algorithms

Transform domain algorithms exploit spatial frequency information contained in the image to achieve compression. For lossless compression, the most popular multiresolution transform-based scheme in the field of medical imaging has been the S-transform (*sequential transform*) [16], [17], [18]. Although this transform is quite efficient, the study by Kuduvalli and Rangayyan [19] shows that it may not be as effective as a predictive coding scheme.

More recently, Said and Pearlman [7] developed a novel multiresolution transform based scheme well suited for lossless and lossy compression of images. This transform is called the $S+P$ transform, where P denotes a prediction stage. Calderbank *et al.*, [9] have recently developed wavelet transforms that map integers to integers. These transforms are quite useful for lossless image compression. In the following, we breifly describe the tranform based compression methods of [7] and [9].

A. $S+P$ Transform based Compression

Said and Pearlman [7] developed an image multiresolution transform that is suited for both lossless (reversible) and lossy compression. This tranform requires only integer addition and bit-shift operations. By using careful scaling and truncation, the number of bits used to represent the transformed image is kept low. Experimental results show that this transform yields far superior compression ratios than the single resolution linear predictive coding based schemes like lossless JPEG.

The S transform which is similar to the multiresolution transform using Harr basis [5] can be represented by the following set of equations for a given input sequence of integers:

$$l[n] = \lfloor \frac{c[2n] + c[2n+1]}{2} \rfloor, n = 0, ..., \frac{N}{2} - 1, \tag{1}$$

$$h[n] = c[2n] - c[2n+1], n = 0, ..., \frac{N}{2} - 1, \tag{2}$$

where $c[n], n = 0, ..., N-1$ is a sequence of integers with an even N, $\lfloor \cdot \rfloor$ denotes the floor function i.e., downward truncation. Note that the truncation removes the redundancy in the least significant bit becuase the sum and difference of two integers is either two odd or two even integers. Also, the division and floor

operations can be very efficiently achieved via a single bit-shift. The inverse transform is given by

$$\begin{aligned} c[2n] &= l[n] + \lfloor \frac{h[n]+1}{2} \rfloor \\ c[2n+1] &= c[2n] - h[n] \end{aligned} \tag{3}$$

The 2D transform simply involves a sequential application of the 1D transform to the rows and columns of the 2D image.

The performance of the S-transform can be improved by combining it with nonlinear predictive coding [7]. Instead of performing the prediction as a post transform operation, Said and Pearlman developed a scheme which predicts the high frequency components from a combination of computed low and high frequency components. Denoting the estimates by $\hat{h}[n]$ we can replace $h[n]$ by the differences $h_d[n]$ which are given by

$$h_d[n] = h[n] - \lfloor \hat{h}[n] + \frac{1}{2} \rfloor, \quad n = 0, 1, \cdots, \frac{N}{2} - 1, \tag{4}$$

to form a new transformed image. The estimation/prediction equation is then given by

$$\hat{h}[n] = \sum_{i=-L_0}^{L_1} a_i \Delta l[n+i] - \sum_{j=1}^{H} b_j h[n+j] \tag{5}$$

where $\Delta l[n] = l[n-1] - l[n]$ is used to obtain zero mean estimated terms. The inverse transform can be achieved with the prediction following a reverse order as given below.

$$h[n] = h_d[n] + \lfloor \hat{h}[n] + \frac{1}{2} \rfloor, \ n = (N/2) - 1, (N/2) - 2, \cdots, 0. \tag{6}$$

The inverse S-transform can be easily computed after recovering the values of $h[n]$. The coefficients a_i, b_i in the above prediction equations can be obtained using filter design techniques and we refer the interested reader to [7].

B. Wavelet Transform based Compression

Traditionally, the Wavelet transform is very popular for lossy compression [21], [22]. More recently, Wavelet transforms that map integers to integers were developed by several researchers [23], [10], [9]. We will briefly describe the technique developed in Calderbank et al., [9] since it is the most general of all the integer wavelet transforms.

Calderbank et al., describe two methods for achieving the mapping of integers to integers using wavelet transforms. One method is a modification of the precoder developed by Laroia et al., [24] for information transmission. The second is a method which is a combination of the lifting scheme [25] and a reversible way of rounding. In this section, we will briefly discuss the latter scheme via an example involving a rewrite of the S-transform using the lifting scheme.

In the lifting scheme, we simply rewrite the S-transform equations as follows:

$$d[n] = c[2n+1] - c[2n] \tag{7}$$

$$l[n] = c[2n] + \lfloor d[n]/2 \rfloor \tag{8}$$

These equations are identical to the S-tranform becuase, $c[2n] + \lfloor d[n]/2 \rfloor = c[2n] + \lfloor c[2n+1]/2 - c[2n]/2 \rfloor$. The inverse tranform can be immediately written as,

$$c[2n] = l[n] - \lfloor d[n]/2 \rfloor \tag{9}$$

$$c[2n+1] = d[n] + c[2n]. \tag{10}$$

After simple algebraic manipulation and using the fact that $k - \lfloor k/2 \rfloor = \lfloor (k+1)/2 \rfloor$ we observe that this is the same as the inverse transform of S. This example shows that lifting makes it possible to obtain an integer transform simply by using truncation without sacrificing invertibility.

In general, the concept of *lifting* from a transform point of view consists of first partitioning an input sequence into even and odd indexed samples – called the lazy wavelet transform – followed by a repeated alternate application of lifting and dual lifting steps. The lifting step involves applying a filter to the odd samples and subtracting the result from the even samples. The dual lifting step does the opposite, it involves applying a filter to the even samples and subtracting the result from the odd samples. After N such applications of dual and primal lifting operations, the even and odd samples produce the low pass and high pass coefficients (upto a scaling factor) respectively. The inverse transform can be obtained by simply reversing the operations and flipping the signs. The lifting scheme implementation of the wavelet transform has numerous advantages over the traditional wavelet transform and we refer the reader to [9] for a detailed discussion on lifting schemes.

In Table V, we present a comparison of the compression results obtained using the integer wavelet transform the same 10 images used earlier for comparing the spatial domain methods. From this table, we see that the

(S+P) algorithm provided the best compression for 7 of our 10 images and tied with the wavelet method on another 2.

TABLE V
COMPARISON OF TRANSFORM DOMAIN SCHEMES

Image	2, 2	4, 2	2+2, 2	S+P
lenna	1.74	1.74	1.74	1.77
man	2.55	2.64	2.58	2.65
chall.	1.77	1.77	1.77	1.79
coral	1.62	1.63	1.63	1.64
shuttle	2.01	2.00	2.01	2.01
sphere	1.80	1.84	1.82	1.85
brain1	1.65	1.68	1.66	1.68
slice15	2.19	2.24	2.21	2.25
head	2.17	2.18	2.17	2.20
sag1	2.82	2.94	2.88	2.92

VII. Experimental Results

In this section, we present a comparison of the various lossless compression algorithms discussed in the previous sections. The effectiveness for lossless compression is measured using the compression ratio CR which is the ratio of the size of the original image to the size of the compressed data stream.

For the evaluation, a selection of 45 images from the ISO test images, the NASA image database, slices from several MR and CT volume image data sets and a diverse set of publicly available images, were used. These images are shown in Figs. 6, 8, 7 and 9. A description of these images follows.

- Among ISO test images, the images–*lenna, man, chall, coral* and *shuttle* are natural gray level images. The image *sphere* is a computer generated image. All images are 256 $\times$ 256 pixels in size and have resolution of 8 bits/pixel, with the exception of *mri, ct, air1, finger* and *cmpnd1* are images of varying sizes with a resolution of 8 bits/pixel. In addition, *cmpnd1* is an image with combination of text and grayscale patterns.
- In the collection of Medical Images, *brain1, brain2, brain3, slice10, slice45, slice90* and *head* are slices from different MRI volume image data sets of the brain. All these are 256 $\times$ 256 pixels in size and have resolution of 8 bits/pixel.

  The images *skull, wrist, carotid, Aperts* are slices from different MR volume image data sets and *liver1,*

liver2, sag, ped_chest are from different CT volume image datasets. All these are 256 × 256 pixels in size and have resolution of 8 bits/pixel.

- NASA Images are a collection from the NASA image research database, which includes different kinds of pictures from space and earth. These images have a resolution of 8 bits/pixel, have different size, and are generally rectangular in shape.
- The diverse set of publicly available images *mandrill, fprint, peppers, country, city, animals, heart, miranda, w6* and *w7* contain various natural scenes and a fingerprint data set. They are of varying sizes but all have a resolution of 8 bits/pixel.

The results obtained using the lossless compression methods: Ranga S+P [7], Wavelet [9]and CALIC [26] are compared. Ranga is a spatial domain compression scheme that uses segmentation. JPEG and LOCO-I/JPEG-LS are the ISO 2D image lossless compression standards. S+P is an effective transform domain algorithm based on the principle of the wavelet transform. Wavelet is a general version of S+P using different basis functions. CALIC is a context-based spatial domain algorithm, which yields the best compression ratios reported in the literature to date. Our experiments were conducted using executables obtained from the authors of the different compression techniques.

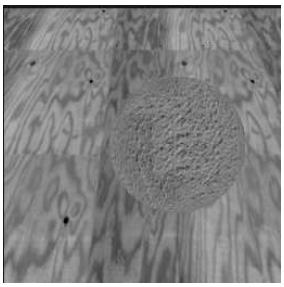
(a) lenna

(b) man

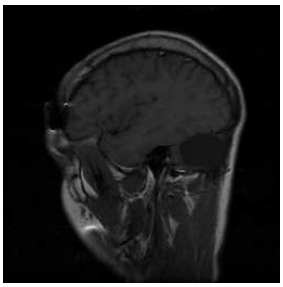
(c) chal

(d) coral

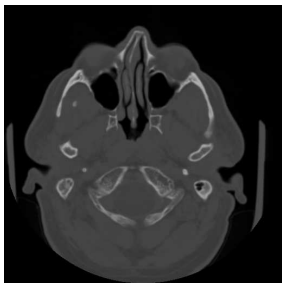
(e) shuttle



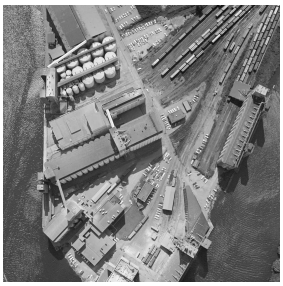
(f) sphere



(g) mri



(h) ct



(i) air1

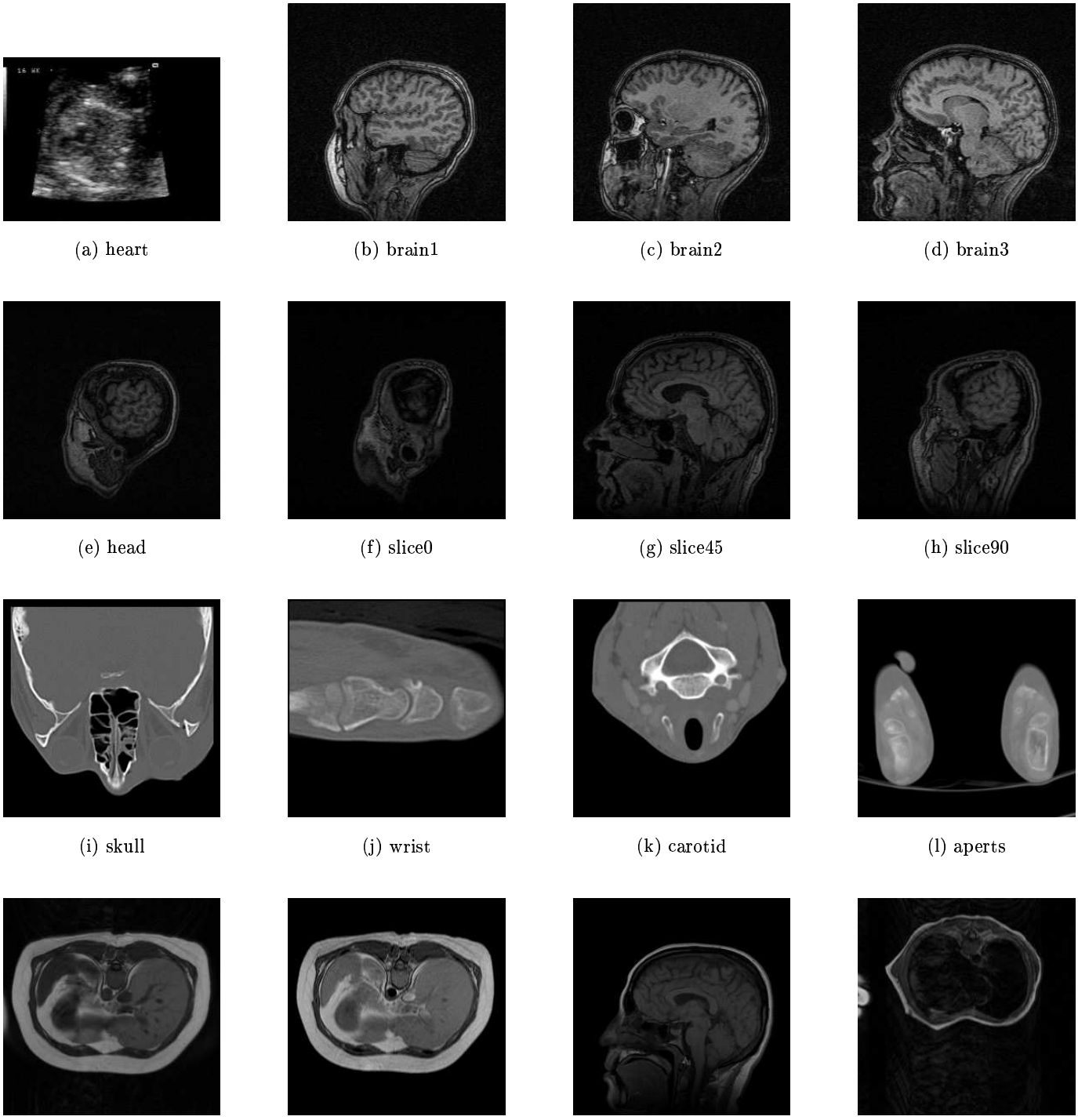
(j) finger

(k) cmpnd1

Fig. 6. ISO Images

In accordance with the four categories of images described earlier, we present four separate comparison tables (Tables VI, VII, VIII, IX) that give the compression ratios obtained by each of the compression methods. The best compression result for each image is marked with a †.

The results (in Tables VI, VII, VIII, IX) indicate that CALIC performed better than S+P, Wavelet, Ranga and JPEG schemes on all the test images. CALIC did better than the LOCO-I/JPEG-LS technique on 42 of 45 images. LOCO-I/JPEG-LS and transform domain schemes perform better than Ranga and JPEG on most



(a) heart (b) brain1 (c) brain2 (d) brain3

(e) head (f) slice0 (g) slice45 (h) slice90

(i) skull (j) wrist (k) carotid (l) aperts

(m) liver1 (n) liver2 (o) sag (p) ped_chest

Fig. 7. Medical Images

TABLE VI
COMPARISON OF THE DIFFERENT 2D SCHEMES ON ISO TEST IMAGES

Image	Ranga	JPEG	JPEG-LS	S+P	Wavelet	CALIC
lenna	1.59	1.56	1.76	1.77	1.75	1.83†
man	2.19	2.04	2.71	2.65	2.65	2.77†
chal	1.67	1.59	1.85	1.79	1.78	1.90†
coral	1.48	1.49	1.64	1.64	1.63	1.68†
shuttle	1.83	1.80	2.05	2.01	2.01	2.12†
sphere	1.63	1.59	1.85	1.85	1.84	1.89†
mri	2.25	2.16	2.69	2.76	2.72	2.78†
ct	4.45	3.51	6.05	5.27	5.64	6.71†
air1	*	1.34	1.45	1.45	1.44	1.49†
finger	*	1.34	1.41	1.45	1.47	1.47†
cmpnd1	*	2.87	6.12	3.33	3.35	6.19†

† indicates the best result. A $*$ indicates that the results were not available

of the images. Comparing the transform domain schemes with LOCO-I/JPEG-LS, we see that the transform domain schemes performed better than LOCO-I/JPEG-LS on square images with noisy background e.g., the brain images, and performed worse on rectangular images. This difefrence is probably due to the fact that the transform schemes require that the number of rows and columns in an image be a power of 8.

VIII. CONCLUSION

We have reviewed state of the art techniques for lossless image compression. These techniques were evaluated experimentally using a suite of 45 images that represent several application domains. Among these techniques we considered, the best compression ratios were achieved by CALIC.

REFERENCES

[1] N.Ranganathan, S.G.Romaniuk and K.R.Namuduri, "A Lossless Image Compression Algorithm Using Variable Block Size Segmentation", *IEEE Transactions on Image Processing*, vol 4, No. 10, pp.1396 - 1406, October 1995.

[2] I. H. Witten, R.M. Neal, and J. G. Cleary, "Arithmetic coding for data compression", *Commum. ACM*, vol. 30, pp. 520-540, June 1987.

[3] W. Kou, *Digital Image Compression - Algorithms and Standards*, Kluwer Academic Publishers, 1995.

[4] R. J. Clarke, *Digital Compression of Still Images and Video*, New York, Academic Press, 1995.

[5] E. H. Adelson, E. P. Simoncelli and R. Hingorani, "Orthogonal pyramid transforms for image coding,", *Proc. SPIE*, Cambridge, MA, Oct. 1987, vol. 845, pp. 50-58.

[6] Gilbert and Strang, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, 1996.

[7] A. Said and W. A. Pearlman, "An Image Multiresolution Representation for Lossless and Lossy Compression", *IEEE Trans. Image Process.*, vol. 5, no. 9, Sept. 1996.

[8] A.Said and W. A. Pearlman, "Reversible Image Compression via Multiresolution representation and predictive coding", *IEEE Trans. Image Processing*, Vol. 2094, pp. 664-674, Nov. 1993.

[9] A. R. Calderbank, I. Daubechies, W. M. Sweldens, and Boon-Lock Yeo, "Wavelet transforms that map integers to integers", *Department of Mathematics, Princeton University*, August, 1996.

[10] A. Zandi, M. Boliek, E. L. Schwartz, and M. J. Gormish, "CREW lossless/lossy medical image compression", *Technical Report CRC-TR-9526, RICOH California Research Center*, 1995.

TABLE VII
COMPARISON OF THE DIFFERENT 2D SCHEMES ON MEDICAL IMAGES

Image	Ranga	JPEG	JPEG-LS	S+P	Wavelet	CALIC
heart	1.71	2.03	2.88	2.70	2.63	2.96†
brain1	1.55	1.49	1.63	1.68	1.68	1.71†
brain2	1.53	1.50	1.65	1.71	1.71	1.73†
brain3	1.47	1.44	1.56	1.62	1.62	1.64†
head	1.92	1.92	2.16	2.20	2.18	2.23†
slice0	2.19	2.17	2.52	2.54	2.51	2.60†
slice45	1.82	1.80	2.05	2.12	2.13	2.15†
slice90	1.96	1.93	2.25	2.29	2.29	2.34†
skull	2.48	2.01	2.82	2.59	2.58	2.94†
wrist	3.63	2.63	4.55	4.09	4.27	4.49†
carotid	3.73	2.76	4.51	4.01	4.05	4.75†
aperts	5.43	3.47	6.95	5.89	5.66	7.13†
liver1	2.19	2.07	2.61	2.65	2.67	2.70†
liver2	2.95	2.22	3.37	3.30	3.32	3.58†
sag	2.53	2.35	3.16	3.02	2.82	3.22†
ped_chest	2.13	2.11	2.47	2.58	2.58	2.60†

† indicates the best result.

TABLE VIII
COMPARISON OF THE DIFFERENT SCHEMES ON IMAGES FROM NASA

Image	Ranga	JPEG	JPEG-LS	S+P	Wavelet	CALIC
EL36	*	1.53	1.81	1.83	1.83	1.86†
EL92	*	1.80	2.34	2.29	2.29	2.46†
EL93	*	1.90	2.56	2.47	2.46	2.59†
EL94	*	1.85	2.47	2.40	2.39	2.53†
EL107	*	1.90	2.53	2.47	2.51	2.63†
EL199	*	2.11	3.06	3.06	3.06	3.13†
EL213	2.76	3.24	4.94	4.46	4.56	5.53†
EL216	2.04	2.77	3.95	3.57	3.50	4.19†
EL475	*	1.95	2.67	2.48	2.48	2.78†

[11] "MPEG, CCITT, H.261, JPEG : Image and Image sequence compression/decompression C software engines", *Portable Video Research Group, Stanford*, 1991.

[12] D.Huffman, "A Method for the Construction of Minimum Redundancy Codes", *Proc. IRE*, Vol 40, pp. 1098-1101, 1952.

[13] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression", *IEEE Trans. on Information theory,* Vol. 24, No. 5, pp. 530-536, 1978.

[14] E. Horowitz, S. Sahni, and D. Mehta, *Fundamentals of Data Structures in C++*, W. H. Freeman, 1994.

[15] T. Welch, "A Technique for High-Performance Data Compression", *IEEE Computer*, June 1994, 8-19.

[16] M. Rabbani and P. W. Jones, "Digital Image compression Techniques". Bellingham, WA: *SPIE*, 1991

[17] V. K. Heer and H. E. Reinfelder, "A comparison of reversible methods for data compression", *Proc. SPIE-Med. Imaging IV*, 1990. vol. 1233, pp. 354-365

[18] S. Takamura and M. Takagi, "Lossless image compression with lossy image using adaptive prediction and arithmetic coding", *Proc. Data Compress. Conf.*, Snowbird, UT, Mar. 1994, pp. 155-174.

[19] G. R. Kuduvalli and R. M. Rangayyan, "Performance analysis of reversible image compression techniques for high-resolution digital teleradiology", *IEEE Trans. Med. Imaging*, vol. 11, pp. 430-445, Sept. 1992

[20] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards, Algorithms and Architectures*, pp. 47-49,

TABLE IX
COMPARISON OF THE DIFFERENT 2D SCHEMES ON NATURAL IMAGES

Image	Ranga	JPEG	JPEG-LS	S+P	Wavelet	CALIC
w6	1.93	2.41	2.69	1.76	1.76	4.24†
w7	1.94	2.41	2.69	1.77	1.77	3.83†
mandrill	1.16	1.16	1.21	1.25	1.25	1.26†
fprint	1.43	1.36	1.77†	1.39	1.39	1.47
peppers	*	1.53	1.73	1.67	1.68	1.76†
city	1.59	1.46	1.90†	1.45	1.45	1.65
country	1.67	1.59	1.99†	1.58	1.58	1.82
animals	*	1.18	1.22	1.21	1.22	1.27†
miranda	*	1.98	2.63	2.66	2.62	2.79†

1995, Kluwer Academic Publishers.

[21] R. A. Devore, B. Jawerth, and B. J. Lucier. "Image compression through wavelet transform coding". *IEEE Trans. Inform. Theory*, 38(2):719-746, 1992.

[22] M. V. Wickerhauser, "High-Resolution Still Picture Compression", *DIGITAL SIGNAL PROCESSING*, 2, 204-226(1992).

[23] S. Dewitte and J. Cornelis. , "Lossless integer wavelet transform", Technical Report IRIS-TR-0041, Royal Meteorological Institute Belguim, 1996.

[24] R. Laroia, S. A. Tretter, and N. Farvardin. "A simple and effective precoding scheme for noise whitening on intersymbol interference channels", *IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 309-312, May 1995.

[25] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets", *Journal of Appl. and Comput. Harmonic Analysis*, 3(2):186-200, 1996.

[26] X. Wu, N. Menon, "CALIC-A context based adaptive lossless image codec," *Proc. of 1996 International Conference on Acousticsss, Speech, and Signal Processing*, pp.1890-1893, 1996.

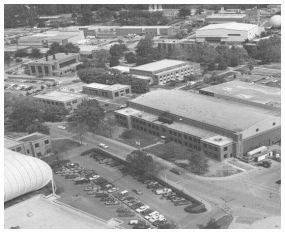
[27] M. J. Weinberger, G. Seroussi, and G. Sapiro, "LOCO-I: A low complexity, context-based lossless image compression algorithm," *Proc. of 1996 Data Compression Conference*, pp. 140-149, 1996.

[28] J. Rissanen, G. Gg. Langdon, Jr., "Universal modeling and coding," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 12-23, Jan. 1981.

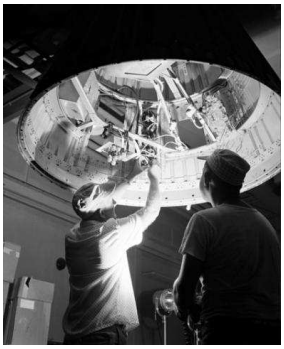
[29] G. G. Langdon, Jr. and C. A. Haidinyak, "Experiments with lossless and virtually lossless image compressin al gorithms," *Proc. SPIE*, vol. 2418, pp. 21-27, Feb. 1995.

[30] D. Salomon, *Data Compression: The Complete Reference*, Springer-Verlag, 1997.

[31] A. Said, W. Pearlman, "A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Trans. On Circuits And Systems For Video Technology*, vol. 6, No. 3, June, 1996.



(a) EL36

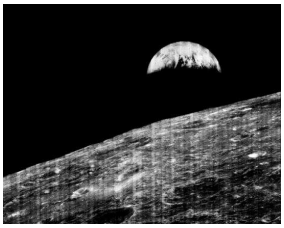


(b) EL92

(c) EL93

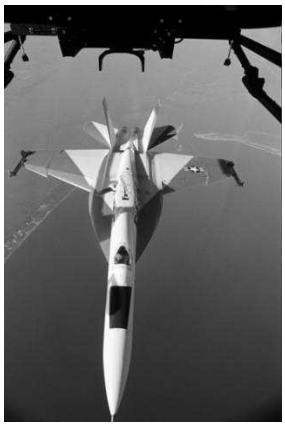
(d) EL94

(e) EL107



(f) EL199

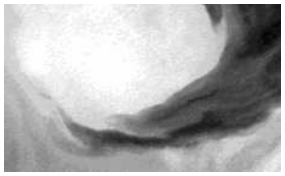
(g) EL213



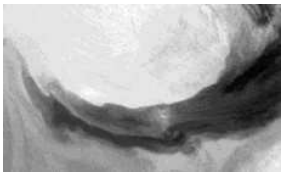
(h) EL216

(i) EL475

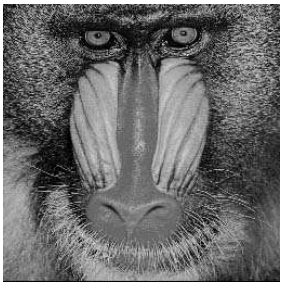
Fig. 8. NASA Images



(a) w6

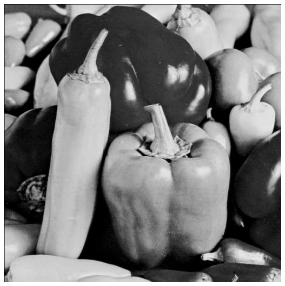


(b) w7



(c) mandrill

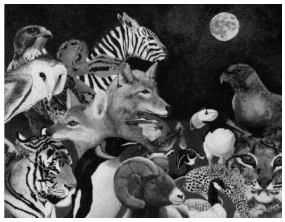
(d) fprint



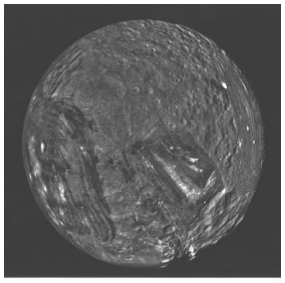
(e) pepper

(f) city

(g) country



(h) animals



(i) miranda

Fig. 9. Other Images