NEW METHODS FOR LOSSLESS IMAGE COMPRESSION USING ARITHMETIC CODING

(extended abstract)

Paul G. Howard[1] and Jeffrey Scott Vitter[2]

Department of Computer Science
Brown University
Providence, R. I. 02912–1910

1 Introduction

Lossless image compression presents a unique set of challenges. Considerable research has already been done on lossless text compression [1,2,3,4,5]; all good methods found to date involve some form of moderately high-order exact string matching. However, this work cannot easily be carried over to lossless image compression, for two reasons: First, images are two-dimensional, so the contexts are more complicated than one-dimensional strings. Second and more important, images are essentially quantized analog data, so the exact matches needed for high-order modeling are only rarely found in the data.

For lossless image compression, a better plan is to find and encode as much of the image structure of the data as possible, and then to efficiently encode the unstructured, noisy residual. We do this in three steps: we *predict the value* of each pixel, we *model the error* of the prediction, and we *encode the error* of the prediction. The predictions correspond to lossy compression; the error encoding makes it lossless.

Most of the output code length comes from the error encoding. If we have a probabilistic model for the errors, we can use arithmetic coding [6] to encode the errors efficiently with respect to the model. To obtain good compression we want the model to assign as high a probability as possible to the prediction error that actually occurs at each pixel. To obtain this, first we need a good predictor, one whose errors have a small variance so that only a few error values are likely and the probability of each is high. In addition, we need a good model for the errors, so that the arithmetic coder is using realistic probabilities. Prediction errors for images are usually distributed according to a Laplace distribution with zero mean, so to obtain a realistic model we need a good estimate of the variance of the errors. Too high a variance allocates too much probability (and code space) to large, unlikely errors, while too low an estimate assigns too little probability to errors that actually occur.

[1]Support was provided in part by NASA Graduate Student Researchers Program grant NGT–50420. Additional support was provided by a Universities Space Research Association/CESDIS appointment.

[2]Support was provided in part by an NSF Presidential Young Investigator Award with matching funds from IBM, by NSF grant CCR-9007851, by ARO grant DAAL03–91–G–0035, and by ONR grant N00014–83–K–0146, ARPA Order No. 4786.

Our new methods clearly separate the steps of image modeling, error modeling, and error coding. Compression is completely lossless, and we obtain excellent compression, typically 2:1 or 3:1, using a modest amount of memory. This compression is better than that obtained by currently-used lossless image encoders.

The first of our methods has the additional advantages that it is easily parallelized and is *progressive.* By "progressive" we mean that the image is encoded in *levels*; the first level corresponds to a very highly compressed and very lossy encoding, and each successive level provides more detail, until ultimately the encoding is completely lossless. Truncating the encoded file will result in lossy compression of the entire image, rather than exact compression of part of the image. A progressive method allows an end user to get a preview of an image without having to decode much of the encoded data; if more detail is desired, the image can be successively refined as more of the encoded data is decoded.

In both of our new methods, we obtain lossless image compression by iterating a three-step coding process. For each pixel (with actual value A_p), we first make a prediction P_p of the pixel's value and compute $\Delta_p = A_p - P_p$; then we model the prediction error by estimating its variance σ_p^2; finally we encode Δ_p using the estimated variance. We can then use the actual value for predicting other pixels, since the decoder can reconstruct the actual value from the prediction and the coded difference.

The error encoding uses arithmetic coding, assuming that the errors are distributed according to a Laplace distribution; for each pixel we choose one of a number of precomputed distributions. In Section 2 we describe the error coding process in detail.

In both our methods the prediction is a linear combination of the values of pixels whose values are already known. Our methods differ from each other in the sequence in which pixels are coded, the specific prediction function, and the method of estimating variances. In Section 3 we describe MLP, a multi-level progressive method that is readily partitioned for parallel processing. It uses a many-point interpolation function for prediction; a number of different methods are available for variance estimation. In Section 4, we describe PPMI, a context-based method. PPMI encodes an image in raster-scan order, predicting with a four-point average; variance estimation of each pixel is based on its context, allowing for inexact matches in a novel way. PPMI gives slightly better compression than does MLP, but it is not progressive; proposed extensions described later may improve MLP's compression significantly. Both MLP and PPMI achieve better image compression than do currently-used methods.

2 Encoding Prediction Errors

Both of our methods encode prediction errors by applying arithmetic coding to a Laplace distribution. In this section we explain our choice of the Laplace distribution and describe the application of arithmetic coding.

2.1 The Laplace distribution

It has long been known that for most images, prediction errors can be very closely approximated by a Laplace (or symmetric exponential) distribution [7,8,9,10]. In particular, the distribution of prediction errors is sharply peaked at zero, which is characteristic of the Laplace distribution but not of the normal distribution.

The probability density function of the Laplace distribution with mean μ and variance σ^2 is given by

$$f_{\mu,\sigma^2}(x) = \frac{1}{\sqrt{2\sigma^2}} \exp\left(-\sqrt{\frac{2}{\sigma^2}}\,|x-\mu|\right).$$

We assume that $\mu = 0$, and define the discrete probability mass function by

$$p_{\sigma^2}(k) = \int_{k-1/2}^{k+1/2} \frac{1}{\sqrt{2\sigma^2}} \exp\left(-\sqrt{\frac{2}{\sigma^2}}\,|x|\right) dx. \qquad (1)$$

2.2 Description of the error coding procedure

If we use probabilities computed from $p_{\sigma^2}(\cdot)$ to encode random variates from a Laplace distribution with a variance other than σ^2, the average code length will be longer than the ideal code length, but if the variances are close, the extra code length will be small. By appropriate coder design, we place a limit on the average extra code length.

Precomputation of the distributions. We wish to obtain probabilities to pass on to the arithmetic coder without excessive computation. To this end, we select a closely-spaced set of variances, with the idea of encoding a prediction error by using the closest variance from the set. We use (1) to compute the Laplace distribution probabilities corresponding to each variance, and build these variances and distributions into both the encoder and the decoder. By careful choice of the set of variances, we can guarantee that the average extra code length due to using an approximate variance will always be less than any given amount. Only 37 variances and distributions are needed to ensure that the coding loss caused by the approximation is less than 0.005 bit per pixel, a barely measurable quantity. We use this set of 37 variances and distributions for the remainder of this paper.

Coding the error. Once we have prepared the probability distributions, arithmetic coding allows us to do the coding without difficulty. Given any discrete probability distribution, we can use arithmetic coding to encode random variates from that distribution with an average code length almost exactly equal to the entropy of the distribution, $\sum_k -p_k \log_2 p_k$. The mechanism of arithmetic coding is straightforward [6], and the extra code length introduced in the coding process is provably small [11]. Unlike Huffman coding, arithmetic coding performs well even when the probabilities are not integer powers of 2.

To encode a prediction error Δ_p (presumed to be a random variate from a Laplace distribution with zero mean and a given estimated variance), we select the "nearest" distribution from the set, that is, the one that minimizes the average extra code length. We then simply use arithmetic coding to encode Δ_p according to the distribution selected.

3 MLP, A Multi-Level Progressive Method

In this section we describe MLP, a new progressive method for image coding. The prediction step can be done completely in parallel by the encoder, and with a high degree of parallelism by the decoder. This is a practical scheme; it uses memory proportional to the size of the image.

We encode the file in a series of *levels*, each level including twice as many pixels as the previous one. As we begin to encode each level, the pixels with known values are on the lattice points of a square grid. Using only the values of pixels on the lattice points, we predict the value of the pixel at the midpoint of each grid square, and encode the prediction error. (Of course the decoder can make the same prediction, and hence can use the decoded error to reconstruct the original value.) After all pixels in the level have been encoded, the set of known pixels form a checkerboard pattern; by rotating and scaling the coordinate system we obtain a new square grid of pixels with known values, with the distance between adjacent pixels only $1/\sqrt{2}$ as much as before.

Clearly the method is progressive: each level gives us the values of pixels selected uniformly from the entire image. For example, just before encoding the last level, we know the values of half the pixels of the image, in a checkerboard pattern over the whole image. If we stopped the encoding at that point, the decoder could compute the known pixels and use the prediction function to interpolate the remaining pixels. In practice, even skipping the last two levels ($3/4$ of the pixels in the image) leaves us with an image virtually indistinguishable from the original.

The time-consuming step in this method is prediction (not arithmetic coding), and the predictions are easy to parallelize. The set of predicting pixels for a given pixel is fixed at the beginning of coding, so for encoding we could even use one processor per pixel to predict in constant parallel time. Since the predictions made during decoding depend on values obtained at earlier levels, only the pixels at a given level can be predicted in parallel; the parallel prediction time for decoding an $n \times n$ image is still small, proportional to the number of levels, $2 \log_2 n$.

Precursors of MLP, which use much simpler predictors and sophisticated variance estimation, are developed in [12,13,14]. The rotating coordinate system appears in [15,16].

3.1 Description of the MLP algorithm

The encoding algorithm proceeds as follows, for each successive level L: First we make a prediction P_p of every pixel $p \in L$, and compute $\Delta_p = A_p - P_p$. Next, for every pixel $p \in L$, we estimate the variance σ_p^2 to be used in encoding Δ_p (or we compute the variance and encode it). Then for every pixel $p \in L$, we encode Δ_p

as described in Section 2. Finally we do housekeeping to rotate and scale the grid. Decoding is very similar. We now examine in detail the steps specific to MLP.

Prediction. To predict the value of a pixel, we use a linear combination of the values of a nearby group of previously coded pixels. A 4×4 array of pixels works well in practice, giving better compression and speed than 6×6 or 8×8 and better compression than 2×2. The coefficients applied to the predicting pixels are selected so that they do midpoint interpolation. We have tried polynomial and Fourier interpolation; in practice, polynomial interpolation works slightly better.

Variance estimation. There is considerable choice in the method of variance estimation. In theory, for each pixel we could find the distribution which encodes the pixel's error as compactly as possible. For the Laplace distribution with mean 0, we find (by solving the equation $\partial p_{\sigma^2}(x)/\partial(\sigma^2) = 0$) that the optimal variance is $\sigma^2 = 2x^2$, twice the square of the error. Of course, this method is not practical because of the large overhead of encoding the optimal variance for each pixel. However, by optimally encoding the errors and *ignoring* the cost of encoding the variances, we can obtain a reasonable lower bound on the attainable code length. This lower bound applies to MLP for a given prediction function and is based on the assumption that the errors follow the Laplace distribution, but it does give a good indication of the compressibility of a particular image.

A more practical method of encoding the variances is to compute and encode the variance of all the errors at a given level; the overhead is minimal, only $\log_2 37 = 5.21$ bits per level. A refinement of this approach is to divide each level into blocks of $k \times k$ pixels and compute and encode the variance of the errors of the pixels in each block. The overhead increases with the number of blocks, but for medium-sized values of k (say $k = 16$) the net code length is often reduced because of the more realistic error model.

Since we expect that predictions made at a given pixel will be about as good as those made at nearby pixels, we can use the errors made in predicting those nearby pixels to estimate the variance of the prediction errors at a given pixel. This method has considerable promise for yielding substantially improved compression because of its potential for accurate variance estimation with no overhead in coding the variances. We are continuing investigation of it.

Housekeeping. After all the pixels in a level have been coded, we rotate the coordinate system by 45 degrees and scale it by $1/\sqrt{2}$. In practice we do this by alternating between diagonal and axis-parallel coordinate systems, reducing the step size at each level.

It is necessary to deal with startup and edge effects, when the points that should be used as predictors are not present in the image. Because of the small number of pixels involved, these effects have little practical significance. In our implementation we simply take the prediction coefficients of missing points to be zero.

4 PPMI, A Context-Based Method with Approximate Matches

The success of high-order, context-based methods for text compression naturally leads us to consider similar ideas for lossless image compression. We present a method similar in spirit to the PPM (prediction by partial matching) method of Cleary and Witten [3]. PPM encodes each input symbol using the longest context in which the symbol has already occurred, up to a specified maximum length. For image compression this method is not completely satisfactory because the exactly-repeated strings that make text compression possible are not present in images. However, we have discovered a new way of detecting approximate two-dimensional matches, eminently suited to image compression. We call this method PPMI.

PPMI encodes an image in raster-scan order. We again separate the coding process into prediction, variance estimation, and error coding. For prediction we use a simple linear combination of a few nearby pixels. For variance estimation we attempt to make use of previous occurrences of the current context; however, instead of trying to maximize the *size* of the matching context as in PPM, we use a small, fixed-size context and focus on the *closeness* of the match. If we fail to find enough previous occurrences of the exact current context, we relax the exactness constraint, ignoring the least significant bit of each of the context pixels. If we still cannot find a match, we ignore the next significant bits until we finally find an approximate "context" that has occurred enough times; we use the statistics of that context to estimate the variance.

The prediction context and the variance-estimation context need not be the same. In our implementation we use the same set of pixels for each, the L-shaped neighborhood consisting of the three pixels above and the one pixel immediately to the left of the pixel being coded. For prediction the context consists of the intensity values of the predicting pixels; the variance-estimation context consists of the errors previously made in predicting the values of the pixels.

An alternative to using an L-shaped neighborhood for prediction of a pixel is to use all eight surrounding pixels. If this is done for each pixel, we can reconstruct the image exactly if we are given the intensity value of just one pixel in the image. However in practice this modification requires significantly more computation and achieves only marginally better compression.

4.1 Description of the algorithm

The encoding algorithm proceeds in raster-scan order. We encode each pixel p as follows: First we make a prediction P_p using the prediction context, and compute the error $\Delta_p = A_p - P_p$. Then we repeat the following steps as many times as necessary to estimate the variance: We construct a key, based on the prediction errors of the pixels in the variance-estimation context, and search (using a hash table) for previous occurrences of the key. If we find enough previous occurrences, we use the error statistics of those occurrences to obtain an estimated mean μ_p and variance σ_p^2 for the prediction error of the pixel, and leave the variance-estimation loop. If we do not find enough previous occurrences, we drop one low-order bit from the prediction errors of the pixels in the variance-estimation context, and construct a new key. After finding μ_p and σ_p^2, we encode $\Delta_p + \mu_p$ as described in Section 2, and update the statistics for one or more of the contexts just examined. Decoding is very similar. We now examine in more detail the steps specific to PPMI.

Prediction. In our current implementation, we predict a pixel's intensity by the rounded arithmetic mean of the intensities of the four pixels in the prediction context. At edges we simply use fewer pixels. Our prediction in this method is admittedly very simple; a more sophisticated extrapolation function would give reduced error variances and improved compression.

Variance estimation. As noted above, we construct "contexts" of decreasing precision based on the previously computed prediction errors of the four pixels in the variance-estimation context. (We could use intensity values, but in practice using error values works slightly better.) Since we are using errors rather than intensities, we have to deal with negative values; we use a signed-magnitude representation, the sign being associated with the most significant nonzero bit.

We follow the procedure described above to find a context that has occurred enough times to allow a satisfactory estimate of the variance of prediction errors made in that context. In practice, a context's statistics are unreliable until the context has occurred 10 or 15 times; we set the default threshold at 12. For the selected context we compute the mean μ_p and the variance σ_p^2 of its previous occurrences. Then we encode $\Delta_p + \mu_p$ as described in Section 2.

Updating the statistics. Each pixel occurs in a number of variance-estimation "contexts" of different precisions. Instead of updating each of them, we adopt a "lazy update" rule that saves time and gives better variance estimates. After coding each pixel we update the statistics for at most two contexts, the one used for coding and, if possible, the one with one additional bit of precision.

5 Experimental Results

As yet there is no standard for lossless image compression against which to compare our methods. There are three reasonable choices for our comparisons: the UNIX *compress* program, a high-order text compression program, and a DPCM image compression program. The *compress* program is the *de facto* standard for text compression, and in fact gives some compression on images, mostly attributable to its ability to capture the context-independent entropy of any file. High-order text compression programs outperform *compress* on text and on images; PPMC [5] is one of the best available programs in this class, so we include it in our comparisons. PPMC does best on most images with a first order model, so that is what we use. We also include a simple, fast, DPCM program that predicts each pixel by the value of the single pixel to its left; it is representative of the class of lossless image compression programs designed for speed.

Our test data consists of three seven-band Landsat Thematic Mapper data sets consisting of 8-bit grayscale images, over Washington, D.C., Donaldsonville, Louisiana (55 miles west of New Orleans), and Ridgely, Maryland (45 miles south-east of Baltimore). The Washington and Donaldsonville images are 512 × 512; the Ridgely images are 368 × 468. In the tables, we designate the various compression methods as follows:

Abbreviation	Compression Method
comp	UNIX *compress*
PPMC	PPMC, using contexts of order 1
DPCM	The simple DPCM program
PPMI	Our PPMI method
MLP	Our MLP method, using a 4 × 4 prediction context and a single variance for each level
MLP_B	Our MLP method, using a 4 × 4 prediction context and a variance encoded for each 16 × 16 block
MLP_opt	The bound for MLP based on using optimal variance estimation, as described in Section 3.1

The figures in the tables are compression ratios, original file size divided by compressed file size. For each file, the best compression ratio is indicated in boldface. The MLP_opt figures are italicized to indicate that they are bounds instead of attainable compression ratios.

Washington, D.C.

Band	*comp*	PPMC	DPCM	PPMI	MLP	MLP_B	MLP_opt
1	1.70	1.97	1.74	**2.18**	2.16	2.17	*2.67*
2	2.21	2.53	1.93	2.76	2.74	**2.76**	*3.63*
3	1.92	2.20	1.83	**2.40**	2.37	2.39	*3.02*
4	1.46	1.71	1.46	**1.91**	1.90	1.90	*2.27*
5	1.34	1.58	1.30	**1.78**	1.76	1.77	*2.09*
6	5.36	6.72	2.00	6.91	**8.66**	8.56	*16.15*
7	1.70	1.99	1.75	2.20	2.20	**2.20**	*2.70*

Donaldsonville, Louisiana

Band	*comp*	PPMC	DPCM	PPMI	MLP	MLP_B	MLP_opt
1	1.79	2.12	1.79	**2.39**	2.27	2.34	*2.97*
2	2.36	2.80	1.91	**3.16**	2.92	3.10	*4.31*
3	1.99	2.36	1.77	**2.72**	2.45	2.59	*3.47*
4	1.34	1.66	1.41	**1.98**	1.89	1.92	*2.36*
5	1.34	1.63	1.37	**1.95**	1.83	1.87	*2.31*
6	6.14	7.77	2.00	8.13	**9.61**	9.55	*18.75*
7	1.65	1.99	1.67	**2.30**	2.21	2.24	*2.81*

Ridgely, Maryland

Band	*comp*	PPMC	DPCM	PPMI	MLP	MLP_B	MLP_opt
1	1.79	2.14	1.85	**2.41**	2.40	2.39	*2.98*
2	2.26	2.71	1.96	3.04	3.04	**3.04**	*4.09*
3	1.86	2.25	1.85	**2.58**	2.55	2.56	*3.29*
4	1.76	2.08	1.80	**2.37**	2.32	2.34	*2.98*
5	1.34	1.64	1.45	**1.91**	1.85	1.86	*2.27*
6	1.58	1.92	1.68	**2.20**	2.17	2.17	*2.68*
7	5.50	6.99	2.00	6.72	**8.06**	7.96	*14.64*

For the Ridgely data, we can make additional comparisons with three other progressive methods: a simple block averaging method (BA), a method based on quadtrees (QT), and a method based on identifying regions containing similar points (RG). The values for these three methods are from [17], where the methods are described in detail.

Ridgely, Maryland

Band	BA	QT	RG	*comp*	PPMC	DPCM	PPMI	MLP	MLP_B	MLP_opt
5	1.2*	1.2	0.9*	1.34	1.64	1.45	**1.91**	1.85	1.86	*2.27*
All	1.2	1.6	1.6	1.91	2.31	1.78	**2.61**	2.60	2.60	*3.33*

*estimated

5.1 Discussion of the Results

For most of the images, PPMI gives the best compression, but it is only slightly better than the two variations of MLP. For all but one of the images, all of our methods outperform all of the other methods; PPMC is consistently the best of the others. The difference between our methods and the MLP_opt bounds indicates that attempting to improve methods of variance estimation should be a fruitful area of further research. We expect that the idea discussed in Section 3.1, estimating the variance of a pixel by using the variances of neighboring pixels, will give good variance estimation with no overhead; this should considerably improve the performance of MLP.

6 Conclusions

We identify the three components of a good predictive lossless image compression method: image modeling and prediction, error modeling, and error coding. We give a practical method for error coding based on arithmetic coding applied to a set of Laplace distributions. We also give two new methods, MLP and PPMI, that make the predictions and variance estimations needed by the coder. Both give excellent compression, and MLP is both progressive and parallelizable.

We are continuing work on mathematical modeling of the prediction errors. We are working with Yali Amit of Brown University on a method based on initial transform coding followed by coding of the error file. With Amit we are also investigating modeling prediction errors with the normal distribution instead of the Laplace distribution, since Gaussian fields are more amenable to mathematical analysis.

References

[1] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Inform. Theory*, IT-23, no. 3, pp. 337–343, May 1977.

[2] ———, "Compression of Individual Sequences via Variable Rate Coding," *IEEE Trans. Inform. Theory*, IT-24, no. 5, pp. 530–536, Sept. 1978.

[3] J. G. Cleary and I. H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Trans. Comm.*, COM-32, no. 4, pp. 396–402, Apr. 1984.

[4] G. V. Cormack and R. N. Horspool, "Data Compression Using Dynamic Markov Modelling," Computer Science Dept., Univ. of Waterloo, Ontario, Research Report CS-86-18, May 1986.

[5] A. Moffat, "A Note on the PPM Data Compression Algorithm," Dept. of Computer Science, Univ. of Melbourne, Victoria, Australia, Research Report 88/7, 1988.

[6] I. H. Witten, R. M. Neal and J. G. Cleary, "Arithmetic Coding for Data Compression," *Comm. ACM*, 30, no. 6, pp. 520–540, June 1987.

[7] J. B. O'Neal, "Predictive Quantizing Differential Pulse Code Modulation for the Transmission of Television Signals," *Bell Syst. Tech. J.*, 45, no. 5, pp. 689–721, May-June 1966.

[8] A. Habibi, "Comparison of nth-Order DPCM Encoder With Linear Transformations and Block Quantization Techniques," *IEEE Trans. Comm. Tech.*, COM-19, no. 6, pp. 948–956, Dec. 1971.

[9] A. N. Netravali and J. O. Limb, "Picture Coding: A Review," *Proc. of the IEEE*, 68, no. 3, pp. 366–406, Mar. 1980.

[10] A. K. Jain, "Image Data Compression: A Review," *Proc. of the IEEE*, 69, no. 3, pp. 349–389, Mar. 1981.

[11] P. G. Howard and J. S. Vitter, "Analysis of Arithmetic Coding for Data Compression," IEEE Data Compression Conference, Snowbird, Utah, 1991.

[12] K. Knowlton, "Progressive Transmission of Gray-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes," *Proc. of the IEEE*, 68, no. 7, pp. 885–896, July 1980.

[13] N. Garcia, C. Munoz and A. Sanz, "Image Compression Based on Hierarchical Coding," *SPIE Image Coding*, 594, pp. 150–157, 1985.

[14] H. H. Torbey and H. E. Meadows, "System for Lossless Digital Image Compression," presented at SPIE, Visual Communication and Image Processing, Nov. 1989.

[15] T. Endoh and Y. Yamakazi, "Progressive Coding Scheme for Multilevel Images," presented at Picture Coding Symp., Tokyo, 1986.

[16] P. Roos, M. A. Viergever, M. C. A. van Dijke and J. H. Peters, "Reversible Intraframe Compression of Medical Images," *IEEE Trans. Medical Imaging*, 7, no. 4, pp. 328–336, Dec. 1988.

[17] J. C. Tilton and M. Manohar, "Hierarchical Data Compression: Integrated Browse, Moderate Loss, and Lossless Levels of Data Compression," preprint, 1990.