Weighted Ratio-based Adaptive Lossless Image Coding

AbdulWahab Kabani and Mahmoud R. El-Sakka
The University of Western Ontario
Computer Science Department
London, ON, Canada
{akabani5,elsakka}@csd.uwo.ca

***Abstract*— in this paper, we present an image compression algorithm called Weighted, Ratio-Based, Adaptive, Lossless image Codec (WRALIC). The algorithm utilizes 5 ratio predictions. The weight of each prediction is learned during a training stage offline, whereas the prediction parameters are adjusted using error context. The absolute value of the error is encoded. The algorithm does not encode the sign. Instead, it attempts to guess the sign of the error from the sign context of the pixel. Using the energy and the average errors around a pixel, the error is added to an encoding bin. Experimental results demonstrate good compression performance compared to other state of the art algorithms.**

Keywords*— *image compression; lossless compression; context modeling; adaotive prediction; entropy coding.

I. Introduction

Data compression is the process of representing an information source using fewer bits than the original representation would use. Compression can be applied on various types of data such as text, speech, image or video [1][2]. In the field of compression, there are two major approaches, namely: lossy and lossless. Lossless compression schemes ensure that the exact original information is recovered after decompressing the compressed file, but they do not provide high compression. Lossy schemes, on the other hand, provide high compression at the cost of information loss in the original data because they reconstruct an approximated replica of the original information after decompression. The research presented in this paper is focused only on lossless image compression.

Lossless image compression schemes can be further classified into statistical-based, dictionary-based, prediction-based and context-based methods. Statistical-based methods encode each pixel value using its probability of occurrence, such as Huffman encoding [3] and arithmetic encoding [4]. Dictionary-based methods utilize the existence of self-similarity among image data and encode the current sequence of pixel values as a pointer to another part of the image that has been already encoded, such as *Lempel-Ziv-77* (LZ77) scheme [5] and *Lempel-Ziv-78* (LZ78) scheme [6]. Prediction-based methods attempt to estimate (predict) the current pixel value, using surrounding pixels and encode the prediction error, instead of the actual pixel value, such as *Differential pulse-code modulation* (DPCM) scheme [7]. Context-based methods encode current pixel value using statistical models built from contexts, such as *Context-based Adaptive Lossless Image Codec* (CALIC) scheme [8], *LOw COmplexity LOssless COmpression for Images* (LOCO-I) scheme [9], and *Prediction by Partial Matching* (PPM) scheme [10].

Statistical modeling [11][12] plays a significant role in determining how good a compression algorithm is. Our algorithm takes advantage of statistical modeling to achieve a relatively high compression rate.

In this article, a new lossless image compression scheme, called *Weighted, Ratio-Based, Adaptive, Lossless Image Coding (WRALIC),* is introduced. The WRALIC algorithm is nearly symmetric, i.e., the time and space complexity to encode is nearly equal to the time to decode. In addition, the WRALIC algorithm is amenable for parallel realization.

The rest of the paper is organized as follows. Section II provides a general overview about the proposed algorithm. Sections III to VI explain each component of the codec in more details. Section VII presents our experimental works by showing the bit rates we achieved along with a comparison with other lossless compression algorithms Finally, Section VIII concludes this work.

II. The algorithm

WRALIC encodes and decodes an image in a raster scan order. While iterating over the image, both the encoder and decoder collect statistical information about the nearby pixels to improve the encoding performance. It is worth mentioning that our algorithm only encodes half of the image (each other row of the image). The rest of the rows are encoded using any other algorithm. In this work, we chose CALIC scheme [8] to encode the rest of the image rows. Figure 1 shows the arrangement of rows that are encoded by our proposed scheme and the rows that are encoded by CALIC.

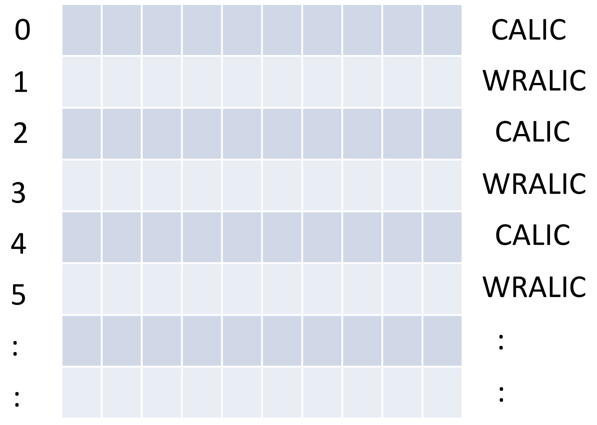


Figure 1. The arrangement of the rows that are encoded by the proposed scheme (WRALIC) and by CALIC

In WRALIC, there are 5 major encoding and decoding stages. These stages are:

- Initial predictions using neighborhood pattern,
- Energy quantization and context selection,
- Getting final prediction,
- Error sign guessing, and
- Entropy coding.

Each of these stages will be elaborated in Sections III to VI (one stage per Section)

III. Initial Predictions

The initial Prediction $I_{initial}$ is based on five ratio predictions. The weight of each ratio prediction is determined offline (during a training stage).

A. The 5 RatioPredictions

Figure 2 shows the neighbors around the pixel being encoded (X). These neighboring pixels are North-West pixel (NW), North pixel (N), North-East pixel (NE), West pixel (W), South-West pixel (SW), South pixel (S), and South-East pixel (SE).

NW	N	NE
W	**X**	
SW	S	SE

Figure 2. The neighbors of the pixel X being encoded: North-West (NW), North (N), North-East (NE), West (W), South-West (SW), South(S), and South-East (SE)

WRALIC uses five prediction rules to predict the value of pixel X. These five prediction rules are based on the following approximations:

$$\frac{W}{N} = \frac{x}{NW} \tag{1}$$

$$\frac{W}{SW} = \frac{x}{S} \tag{2}$$

$$\frac{NW}{x} = \frac{x}{SE} \tag{3}$$

$$\frac{NE}{x} = \frac{x}{SW} \tag{4}$$

$$N - x = x - S \tag{5}$$

The above ratios can be re-arranged to look like the equations below.

$$I_{r0} = \frac{W \times NW}{N} \tag{6}$$

$$I_{r1} = \frac{W \times S}{SW} \tag{7}$$

$$I_{r2} = \sqrt{NW \times SE} \tag{8}$$

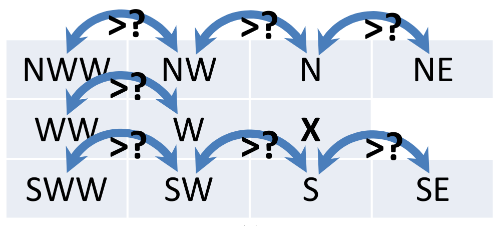
$$I_{r3} = \sqrt{NE \times SW} \tag{9}$$

$$I_{r4} = \frac{N + S}{2} \tag{10}$$

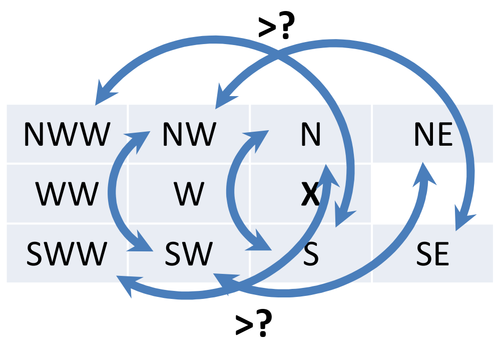
B. Patterns

The initial prediction $I_{initial}$ is a weighted sum of the five ratio predictions (I_{r0}, I_{r1}, I_{r2}, I_{r3}, and I_{r4}). The five weights are determined based on the pattern around the pixel being encoded (context) using a training set.

The context is identified based on a set of 14 comparisons between the pixels in the current and the previous blocks. Seven of these comparisons are between individual immediate neighboring pixels (see Figure 3(a)). Another three are based on diagonal comparisons between (NW+SE, NWW+S), (NE+SW, N+SWW), and (N+S, NW+SW) (see Figure 3(b)). The last four comparisons are based on comparing two group of pixels, including, (N + NW, W + WW), (W + WW, S + SW), (||NW − SE||, ||NWW − S||), and (||NE − SW||, ||N − SWW||).



(a)



(b)

Figure 3. The context is identified based on a set of comparisons between the pixels in the current and the previous blocks.

Each comparison result is either 0 or 1, as shown in (11)

$$Compare(x, y) = \begin{cases} 1, if\ y > x \\ 0\ otherwise \end{cases} \tag{11}$$

With this arrangement, there are $2^{14} = 16384$ possible contexts.

C. Weighted Average

The initial prediction, $I_{initial}$, is determined by (12):

$$I_{initial} = \theta_0 I_{r0} + \theta_1 I_{r1} + \theta_2 I_{r2} + \theta_3 I_{r3} + \theta_4 I_{r4} \tag{12}$$

The values of the weights θ_0, θ_1, θ_2, θ_3, and θ_4 are determined offline for each possible context. At the beginning of the training process, all weights are set to have the same value, i.e., $\theta_0 = \theta_1 = \theta_2 = \theta_3 = \theta_4 = 0.2$. Then, the algorithm goes over the training images. While looping through each training image, the counter of the ratio prediction that gives the closest approximation to the current pixel is incremented, as shown in (13).

$$increment = \alpha \left\| I_{pixel} - I_{initial} \right\| \tag{13}$$

The symbol α denotes the energy in the current block. Simply, the energy α is a measure of how smooth the current block is. In non-smooth areas, the value of α should be large. Therefore, there's a high probability to have a prediction that is far from the pixel. Therefore, the increment should be high in areas with high frequency. In smooth blocks, the weights should be small, as the probability to get a wrong prediction here is lower. We will explain how to compute the energy in the Section IV.

The value $\|I_{pixel} - I_{initial}\|$ is the initial error. If the initial error is small, then the current weights are ideal. Therefore, we don't want to have a large increase that may change the ideal weights. On the other hand, if the error is large, we should have a large increment.

In other words, there's a linear dependency between an increment for a certain weight and the energy. Also, the relation between an increment to a certain weight and the initial error is also linear.

IV. Energy and Error Context

The energy is a measure of how smooth the neighborhood is. The smoother the neighborhood, the more accurate our prediction will be. The energy value is quantized to 8 bins. These 8 bins form our coding contexts. We do entropy coding based on estimated conditional probabilities like in Xu *et al.* [8].

A. Energy

The energy is a sum of 3 values: d_h (horizontal absolute changes), d_v (vertical absolute changes), and e_w (previous error), where the value of α is calculated according to (17).

$$d_h = \|NE - N\| + \|N - NW\| + \|SE - S\| + \|S - SW\| \quad (14)$$

$$d_v = \|NW - W\| + \|W - SW\| + 0.5\,\|N - S\| + 0.5\,\|NE - SE\| \quad (15)$$

$$e_w = W - I_{initial(previous)} \quad (16)$$

$$\alpha = d_h + d_v + 2\,e_w \quad (17)$$

It is worth mentioning that the energy value α controls the amount of increment in (13). In addition, it is used to construct the error context, (see Section IV-B) and to determine the bin that will be used to encode the final error (see Section VI).

The energy is quantized using the following levels [9, 18, 30, 45, 90, 128, and 210]. For example, if the energy is ≤ 9, the quantized energy is 0. If the energy is ≤ 18, the quantized energy is 1 and so on.

B. Error Context

The error context is created by comparing the initial prediction $I_{initial}$ with each neighboring pixel. If the neighboring pixel is larger or equal to the initial prediction, we append 1 to the context. Otherwise, we append 0. After that, we append the quantized energy/2. Therefore, the total number of possible error contexts is $2^7 \times 4$ (quantized energy) = 512 error contexts.

For each error context, the initial error (how far the initial prediction $I_{initial}$ is from the pixel being encoded) is summed during the online learning stage (at the end of each iteration). Also, a running count of the pixels that fall under each context is kept. Using the running sum with the running count of the initial error, the final prediction I_{final} is calculated (see (18) and (19)).

$$\epsilon(C) = \frac{running\_sum(C)}{running\_count(C)} \quad (18)$$

$$I_{final} = I_{initial} + \epsilon(C) \quad (19)$$

V. Error Sign Guessing

A. Encoding the Absolute Error

Instead of encoding the error with its sign, we encode the absolute error. In order to do that, we create a sign context that can help us determine the sign of the error. This context cannot predict the sign correctly all the time. However, using the sign context, we can create a skewed distribution, which is great for encoding.

To illustrate, the number of negative errors is usually similar to the number of positive errors. Since the probability of both types of errors is almost the same for each sign, we know from information theory that the arithmetic encoder will not perform well due to the high uncertainty.

Using sign contexts, we don't have to encode the sign of the error. Instead, we encode whether the encoder was able to guess the correct error sign or not. The probability distribution of our guessing success or failure is skewed. This leads to better entropy Encoding.

B. The Sign Error Context

Given the final prediction I_{final} , we construct a context in a similar manner to the way we did it in Section IV.B, where, we constructed a context using $I_{initial}$ and used the context to calculate the average error for each context. Here, we construct a context in the same manner but using I_{final}.

For each context of the 512 sign contexts, the encoder and decoder keep 2 running counts of the positive and negative errors. This is done during the online learning stage, which comes after encoding the pixel. Both the encoder and decoder keep track of the number of positive and negative errors for each sign context. This is illustrated in Figure 4.

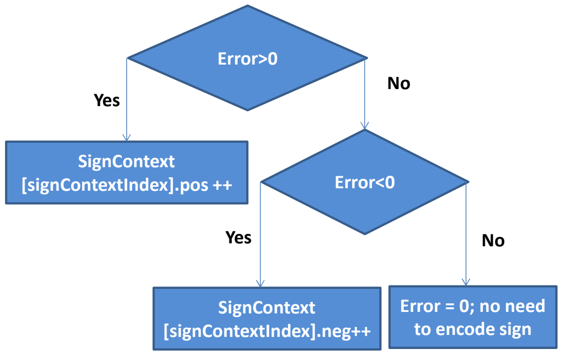

Error>0
Yes
No
SignContext [signContextIndex].pos ++
Error<0
Yes
No
SignContext [signContextIndex].neg++
Error = 0; no need to encode sign

Figure 4. A flowchart of the content of the Sign Context. During the learning stage (after encoding the pixel), if the error is positive, the counter of positive errors is increased for this context. Otherwise, the counter of negative errors for this sign context is increased.

When encoding the pixel, if the sign of the current error is the same as the dominate sign of the context, success (0) is encoded. Otherwise, the encoder encodes failure (1).

As shown in Figure 5, in addition to encoding the sign guessing result, a counter for success and failure is incremented for each sign context. This is important for the next stage, which is entropy encoding. The result of sign guessing will be added to one of 6 bins depending on the context probability of success. This leads to better compression results because we are grouping symbols by their probability distribution.

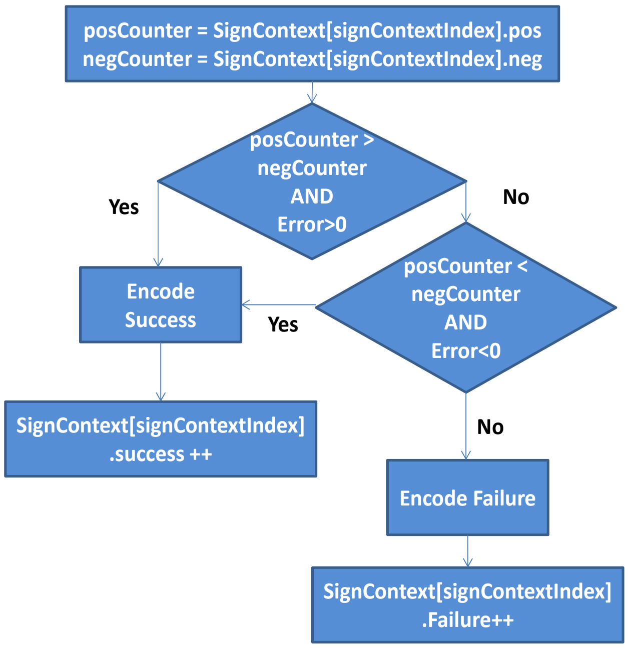

posCounter = SignContext[signContextIndex].pos
negCounter = SignContext[signContextIndex].neg
posCounter > negCounter AND Error>0
Yes
No
posCounter < negCounter AND Error<0
Encode Success
Yes
SignContext[signContextIndex] .success ++
No
Encode Failure
SignContext[signContextIndex] .Failure++

Figure 5. A flowchart of how both the success and failure counters are being incremented during the online learning stage. If the error is positive and the number of positive errors for this context is more than the negative errors, the encoder increments the "success" counter. Similarly, if the number of negative errors for this context is more than the positive and the current error is negative, the success counter is incremented. Otherwise, the encoder increments the "failure" counter.

VI. Entropy Encoding

We use an adaptive arithmetic encoder to encode prediction errors and sign guessing data. Depending on the context of the errors being encoded, the error can go into one of 16 encoding bins. In addition , the sign guessing of the error can get to one of 6 sign guessing bins. Each bin is encoded using a separate arithmetic encoder.

As per Section IV, there are 8 possible energy levels. We combine these 8 possible levels with average errors around the pixel being encoded to end up with 16 possible error bins. The errors around the pixel being encoded are: $\|NN - I_{NN}\|$, $\|NNW - I_{NNW}\|$, $\|NNE - I_{NNE}\|$, and $\|W - I_W\|$. We take the average of these errors and depending on the energy level, we decide to which error is added to.

The encoding bins are: binL0, binL1, binL2, binL3, binL4, binL5, binL6, binL7, binH0, binH1, binH2, binH3, binH4, binH5, binH6, and binH7. Each bin corresponds to an energy level. For each energy level, there are two possibilities, depending on the value of the average error around the pixel being encoded. The threshold values for each energy level are determined (20). These values are tuned empirically:

$$\begin{aligned} N_0 &= 1; \quad N_1 = 1 \\ N_2 &= 5; \quad N_3 = 6 \\ N_4 &= 8; \quad N_5 = 9 \\ N_6 &= 15; \quad N_7 = 15 \end{aligned} \tag{20}$$

For Example, if the energy level of the neighborhood of the pixel is 4, the average error around the pixel is calculated. If this value is less than N_4, i.e., less than 8, the error is added to binL4. Otherwise, it is added to binH4.

As mentioned in Section V, the encoder only encodes the absolute value of the error. The sign is not encoded. Instead, the encoder encodes whether we were able to guess the sign of the error (0) or not (1). The probability distribution of the guessing is skewed towards successful guessing (more 0s).

To improve the performance of the arithmetic encoder, we divide the sign guessing into 6 bins. The division is based on the success probability of each context. This algorithm is summarized in Figure 6. The decision boundaries (66%, 57%, 50%, 43%, 34%) are set empirically.

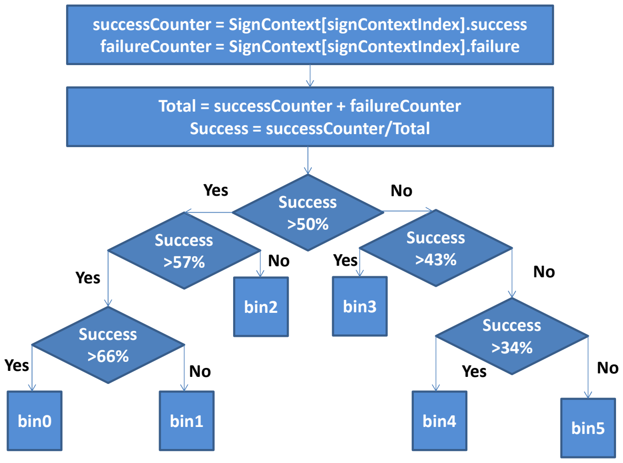

successCounter = SignContext[signContextIndex].success
failureCounter = SignContext[signContextIndex].failure
Total = successCounter + failureCounter
Success = successCounter/Total
Yes
Success >50%
No
Success >57%
No
Yes
Success >43%
Yes
bin2
bin3
No
Success >66%
Success >34%
Yes
No
Yes
No
bin0
bin1
bin4
bin5

Figure 6. Flowchart of the entropy coding of the sign of the error. The sign guessing is added to an encoding bin depending on the sign context success probability.

VII. Results and Experiments

For the sake of compression performance evaluation, the proposed method is tested on the Kodak image set, which contains 24 grey-scale images (9437544 bytes in total). Figure 7 shows the Kodak image set with the compression performance of WRALIC. In order to demonstrate the compression performance of WRALIC, we compare it with some other lossless compression techniques, where each image is compressed separately and the total size of compressed files of each method is reported. Table 1 lists the compression results achieved by WRALIC, JPEG-LS [9], PAQ [13], CALIC [8]. Our Algorithm outperforms JPEG-LS. In addition, we achieve a bit rate that is slightly better than the one achieved by CALIC. However, PAQ achieves a better compression rate. Yet, the execution time of PAQ is very high. The execution time for WRALIC (implemented using Python) is on average 9 seconds on a machine with 2GB memory. Currently, we are working on implementing it using C and further optimizing the code to achieve faster execution.

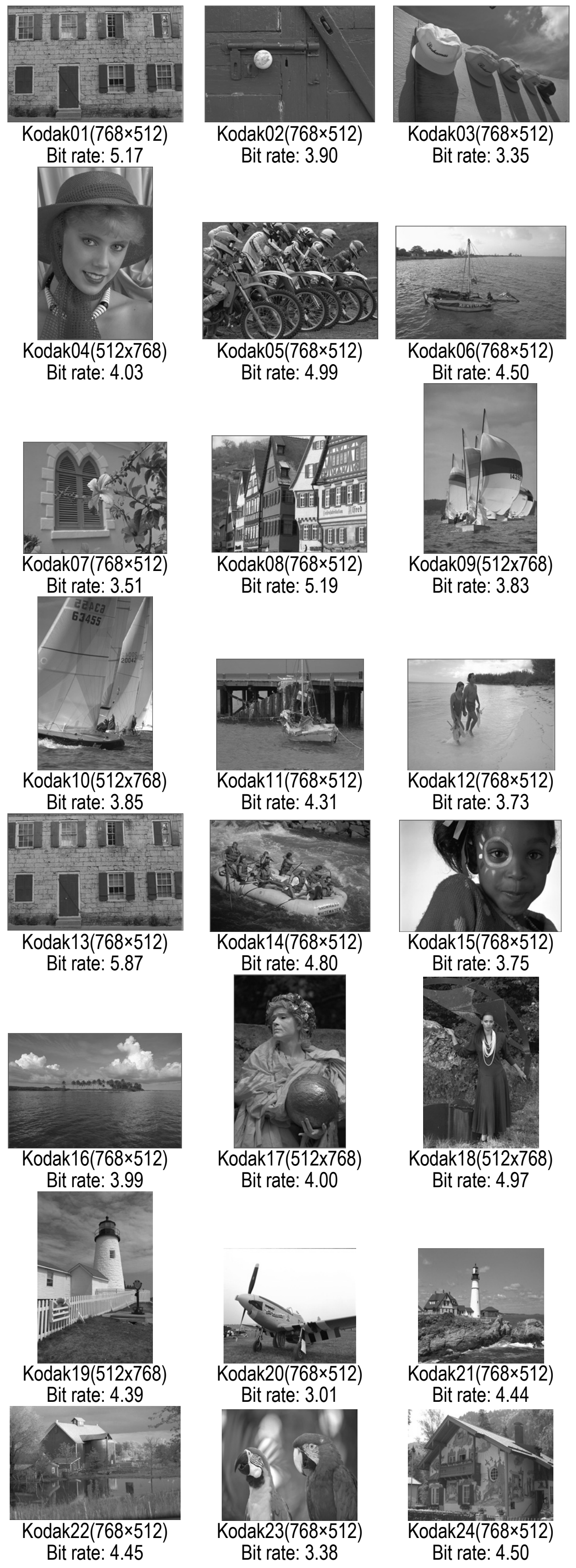

Kodak01(768×512) Bit rate: 5.17
Kodak02(768×512) Bit rate: 3.90
Kodak03(768×512) Bit rate: 3.35
Kodak04(512x768) Bit rate: 4.03
Kodak05(768×512) Bit rate: 4.99
Kodak06(768×512) Bit rate: 4.50
Kodak07(768×512) Bit rate: 3.51
Kodak08(768×512) Bit rate: 5.19
Kodak09(512x768) Bit rate: 3.83
Kodak10(512x768) Bit rate: 3.85
Kodak11(768×512) Bit rate: 4.31
Kodak12(768×512) Bit rate: 3.73
Kodak13(768×512) Bit rate: 5.87
Kodak14(768×512) Bit rate: 4.80
Kodak15(768×512) Bit rate: 3.75
Kodak16(768×512) Bit rate: 3.99
Kodak17(512x768) Bit rate: 4.00
Kodak18(512x768) Bit rate: 4.97
Kodak19(512x768) Bit rate: 4.39
Kodak20(768×512) Bit rate: 3.01
Kodak21(768×512) Bit rate: 4.44
Kodak22(768×512) Bit rate: 4.45
Kodak23(768×512) Bit rate: 3.38
Kodak24(768×512) Bit rate: 4.50

Figure 7. Compression Bit Rate (bits/pixel) for Kodak standard images

Table.1. The performance of WRALIC against other algorithms. Bit Rats (bits/pixel). Sizes in bytes.

	Original Size	**Compressed Size**	**Bit Rate**
WRALIC	9,437,544.00	5,012,001.00	4.25
JPEG-LS	9,437,544.00	5,120,281.00	4.34
PAQ	9,437,544.00	4,726,326.00	4.01
CALIC	9,437,544.00	5,020,278.00	4.26

VIII. Conclusions

We have presented an image context-based coding algorithm that achieves a good bit rate. The method utilizes five ratio predictions. Instead of encoding the sign of the error, we try to guess the sign. The probability distribution of the guessing is skewed. Therefore, we achieve more compression by encoding the guessing of the sign rather than the sign of the error. We perform entropy encoding by encoding the absolute error in one of 16 bins. The average error around the pixel plays a significant role in determining the error encoding bin.

We have also compared our algorithm to other state of the art algorithms. Based on our experiments, we found that WRALIC achieves an excellent bit rate. Its performance is less than PAQ; however, WRALIC is much faster than PAQ. Yet, we still have more room to further optimize our implementation. Currently, we are doing so.

Since we are encoding only half of the image, we have access to pixels below the pixel being encoded. If we were able to encode the whole image in this algorithm, we would end up with a bit rate that is better than most state of the art algorithms. This is because having access to the row below the pixel being encoded improves pixel predictions. However, encoding the whole image is not possible because the decoder will not be able to decode the image. The result of this tradeoff is a slight improvement in performance over state of the art algorithms.

Acknowledgement

This research is partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC). This support is greatly appreciated.

References

[1] K. Sayood, "Introduction to data compression," Amsterdam: Elsevier, 2012.

[2] D. Salomon, "Data compression: the complete reference," London: Springer, 2010.

[3] D. Huffman, "A method for the construction of minimum-redundancy codes", Proceedings of the I.R.E., pp. 1098–1102, September 1952.

[4] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," Communications of the ACM, 30(6), pp. 520–540, June 1987.

[5] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Transactions on Information Theory, 23(3), pp. 337–343, May 1977.

[6] J. Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding,” IEEE Transactions on Information Theory, 24(5), pp. 530–536, September 1978.

[7] C. Cutler, “Differential quantization of communication signals,” U.S. patent 2,605,361, July, 1952.

[8] X. Wu and N. Memon, “Context-based adaptive lossless image coding,” IEEE Transactions on Communications, 45(4), pp. 437–444, April 1997.

[9] M. Weinberger, G. Seroussi, and G. Sapiro, “The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS,” IEEE Transactions on Image Processing, 9(8), pp. 1309–1324, August 2000.

[10] J. Cleary and I. Witten. “Data compression using adaptive coding and partial string matching,” IEEE Transactions on Communications, 32(4), pp. 396–402, April 1984.

[11] J. Rissanen, “Universal coding, information, prediction and estimation,” IEEE Transactions on Information Theory, 30(4), pp. 629–636, July 1984.

[12] M. Weinberger, J. Rissanen, and R. Arps, “Applications of universal context modeling to lossless compression of gray-scale images,” IEEE Transactions on Image Processing, 5(4), pp. 575–586, April 1996.

[13] D. Salomon, G. Motta, (with contributions by D. Bryant), Handbook of Data Compression, 5th edition, pp.314–319, Springer, 2009.