Adaptive Weighted Neighbors Lossless Image Coding

AbdulWahab Kabani and Mahmoud R. El-Sakka(✉)

Department of Computer Science,
The University of Western Ontario, London, ON, Canada
{akabani5,melsakka}@uwo.ca

Abstract. Adaptive Weighted Neighbors Lossless Image Coding *AWN* is a symmetric lossless image compression algorithm. *AWN* makes two initial predictions, creates a weighted combination of the initial predictions before adjusting the prediction to end up with the final prediction. In order to achieve more compression, we encode the error in multiple bins depending on the expected error magnitude. Also, instead of encoding the signed error, the algorithm attempts to guess the sign and encodes the error magnitude and whether guessing the sign was successful or not.

Keywords: Image compression · Lossless compression · Context modeling · Adaptive prediction · Entropy coding

1 Introduction

Data compression is the process of representing information using fewer bits than the original representation would use. The main objective of data compression is to reduce the size of the information being encoded. As the size of different kinds of data (text, audio, and video) is growing, the need to have better compression techniques is increasing [5, 7].

In general, compression can be broken down into two major fields, namely: lossy compression and lossless compression. Lossy compression usually achieves excellent compression rates at the expense of information loss. In other words, the reconstructed information after compression and decompression is not an exact replica of the original information before compression. On the other hand, lossless compression achieves less compression than lossy. The main advantage of lossless compression is that the reconstructed information matches the original information exactly. This is very important for legal and medical applications. The research presented in this paper is a lossless image compression. Therefore, the reconstructed image is exactly the same as the original image.

There are many methods that are used on image to achieve compression. These methods include statistical methods (Huffman encoding [3] and Arithmetic encoding [9]), dictionary methods (Lempel-Ziv-77 (LZ77) scheme [11] and Lempel-Ziv-78 (LZ78) scheme [12]), prediction methods (Differential pulse-code modulation

M. Kamel and A. Campilho (Eds.): ICIAR 2015, LNCS 9164, pp. 129–138, 2015.
DOI: 10.1007/978-3-319-20801-5_14

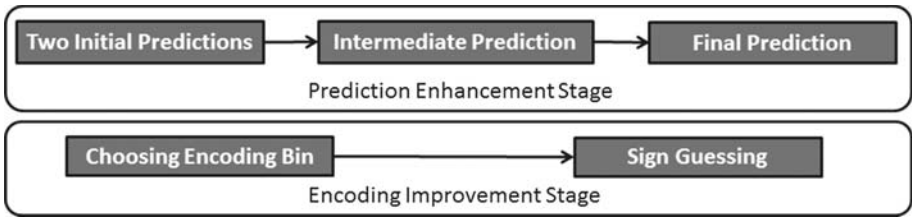

Fig. 1. Algorithm Overview: AWN consists of two main stages. The first stage is the *Prediction Enhancement Stage*, which consists of three steps that starts with two *initial predictions* and ends with a *final prediction*. The second stage aims to improve the entropy to achieve a better compression. This is achieved by grouping the errors in different encoding bins and attempting to guess the sign of the error.

(DPCM) scheme [2]), and context methods (Context-based Adaptive Lossless Image Codec (CALIC) scheme [10], LOw COmplexity LOssless COmpression for Images (LOCO-I) scheme [8], Prediction by Partial Matching (PPM) scheme [1], and Weighted Ratio-based Adaptive Lossless Image Coding [4]).

In this paper, we introduce Adaptive Weighted Neighbors (AWN) Lossless Image Coding. AWN is a lossless image codec. It combines statistical-based, prediction-based, and context-based techniques to achieve an excellent compression rate. The rest of the paper is organized as follows. Section 2 provides a general overview about the proposed algorithm. Sections 3 to 6 explain each component of the codec in more details. Section 7 presents our experimental works by showing the bit rates we achieved along with a comparison with other lossless compression algorithms. Finally, Sect. 8 concludes this work.

2 General Overview

The system can be broken down into 6 major steps. These steps are: calculating two initial predictions, combining the two predictions, prediction adjustment, error sign guessing, choosing an encoding bin and entropy encoding.

Figure 1 shows how these steps can be broken down into two stages: *prediction enhancement* stage and *encoding improvement* stage. In the *prediction enhancement* stage, we start with two *initial predictions*. Then, we combine them into an *intermediate prediction*. After that, the prediction is adjusted through an *error context feedback*. The aim of this stage is to come up a prediction of the pixel being encoded.

Better compression can be achieved when encoding the error in an effective way. This is done in the *encoding improvement* stage. In this stage, we choose the encoding bin that promises to yield the lowest entropy. In addition, we do not encode the sign of the error. Instead, we try to guess the sign. Finally, we perform entropy encoding such as: arithmetic encoding.

3 Predictors: Calculating Initial Predictions

Statistical redundancy in a set of pixels is the smoothness of the intensity function. In other words, pixels spatially close to each other tend to have similar values. The predictor of AWN views the prediction as depending on the direction of the small changes. There are four directions that we use to calculate the prediction. These directions are: horizontal, diagonal (45 degrees), vertical, diagonal (135 degrees). The direction with the smallest absolute change tends to give the best prediction.

Based on this notion, we designed our predictor. The predictor gives more weight to predictions that are inferred from the directions with the least changes. Equations 1–4 define the gradient magnitude estimates in the four directions:

$$GM_h = ||I_W - I_{WW}|| + ||I_{NW} - I_{NWW}|| + ||I_N - I_{NW}|| + ||I_{NE} - I_N|| \quad (1)$$

$$GM_{D1} = ||I_W - I_{NWW}|| + ||I_{NW} - I_{NNWW}|| + ||I_N - I_{NNW}|| + ||I_{NE} - I_{NN}|| \quad (2)$$

$$GM_v = ||I_W - I_{NW}|| + ||I_{NW} - I_{NNW}|| + ||I_N - I_{NN}|| + ||I_{NE} - I_{NNE}|| \quad (3)$$

$$GM_{D2} = ||I_W - I_N|| + ||I_{NW} - I_{NN}|| + ||I_N - I_{NNE}|| + ||I_{NE} - I_{NNEE}|| \quad (4)$$

where $I_W, I_{WW}, ..., I_{NNEE}$ are the values of the neighbours of the pixel. For example, I_W is the pixel to the west of the current pixel and I_{NW} is the northwest neighbour of the pixel.

Figure 2 shows how we calculate the gradient magnitude estimates in the four directions. The gradient magnitude estimate in each direction is the summation of the absolute differences of the neighboring pixels. The lower the value of the magnitude, the more likely a prediction based on the corresponding direction can yield better results.

The less the magnitude of absolute changes in one direction, the higher the weight of the corresponding pixel should be. In other words, the weight of each pixel is determined by dividing the total absolute changes in all directions by the directional absolute changes.

$$\delta = GM_h + GM_{d1} + GM_v + GM_{d2} \quad (5)$$

$$w_h = \frac{\delta}{GM_h}, w_{d1} = \frac{\delta}{GM_{d1}}, w_v = \frac{\delta}{GM_v}, w_{d2} = \frac{\delta}{GM_{d2}} \quad (6)$$

We define two initial predictions. The first initial prediction is a weighted combination of the neighboring pixels: W, NW, N, and NE. The *Horizontal*, *Diagonal 1*, *Horizontal*, and *Diagonal 2* directions correspond to W, NW, N, and NE, respectively. The resulting weights are normalized. We need to normalize them when we calculate the predictions.

$$w_{total1} = w_h + w_{d1} + w_v + w_{d2} \quad (7)$$

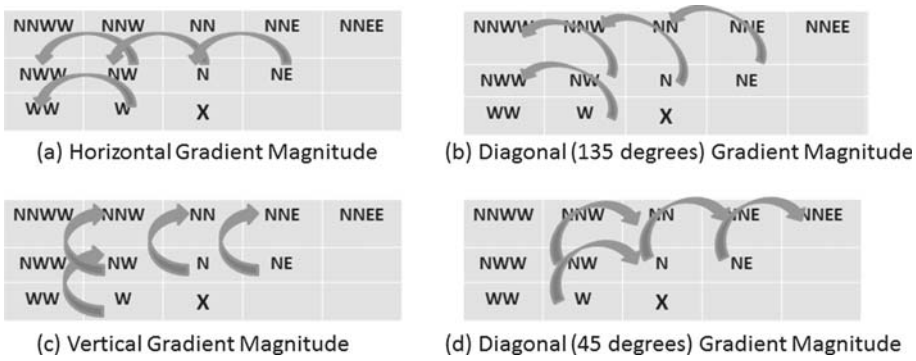

Fig. 2. Gradient magnitude estimation: (a) The estimation of the horizontal gradient magnitude, which is performed by taking the absolute values of the differences indicated by the arrows. (c),(b) and (d) are the gradient magnitude estimations in the vertical directions and the diagonals respectively.

$$w_{h,norm} = \frac{w_h}{w_{total1}}, w_{d1,norm} = \frac{w_{d1}}{w_{total1}}, w_{v,norm} = \frac{w_v}{w_{total1}}, w_{d2,norm} = \frac{w_{d2}}{w_{total1}} \tag{8}$$

After normalizing the weights for the initial prediction 1, we can now calculate the initial prediction 1. The weights play a significant role in computing the first initial prediction. If the gradient magnitude estimation in one of the four directions is low, the corresponding neighbor will have higher contribution to the first initial prediction.

$$I_{initial1} = w_{h,norm} \times I_W + w_{d1,norm} \times I_{NW} + w_{v,norm} \times I_N + w_{d2,norm} \times I_{NE} \tag{9}$$

The second initial prediction uses only the pixels that correspond to the directions with the least changes and the second least changes (lowest and second lowest gradient magnitude estimations). More weight is given to the pixel that corresponds to the direction with the least change. To do that, we boost the original weight of the pixel with the minimum change. The boosting value will always be larger than 1 because the nominator is always larger than the denominator.

$$w_{LowestBoosted} = \frac{w_{lowest}}{w_{2ndLowest}} \times w_{lowest} \tag{10}$$

In order to use these weights, we should normalize them first. The normalization process is similar to the one we did for the initial prediction 1. We first get the total of the 2 weights. Then, we divide these weights by the total.

$$w_{total2} = w_{LowestBoosted} + w_{2ndLowest} \tag{11}$$

$$w_{LowestBoosted,Norm} = \frac{w_{LowestBoosted}}{w_{total2}} \tag{12}$$

$$w_{2ndSmoothest,Norm} = \frac{w_{2ndSmoothest}}{w_{total2}} \tag{13}$$

Fig. 3. This figure shows the weights of the neighbours of the pixel being encoded.

Using the normalized weights, we can compute the second initial prediction (Eq. 14). The contribution of the neighbor that corresponds to the best prediction is always more than the one with the second best. For example, if the lowest and second lowest gradient magnitude estimations were in the horizontal and the vertical directions, the best and second neighbors that we use to compute the second initial prediction are the W and N pixels.

$$I_{initial2} = w_{LowestBoosted,Norm} \times I_{best} + w_{2ndSmoothest,Norm} \times I_{2ndbest} \quad (14)$$

4 Combining the Two Predictions

Using the two *initial predictions* we calculated, we compute the *intermediate prediction*. The *intermediate prediction* is a weighted combination of the two initial predictions. We have found that creating a weighted combination of the two initial predictions yields a better compression rate. In order to determine the weight of the two *initial predictions*, we examine their errors for pixels that are spatially close to the pixel being encoded. Pixels closer to the pixel we are encoding are much more important than pixels that are far away. Figure 3 shows the weights of the neighbours surrounding the pixel being encoded.

Using the weights shown in Fig. 3, we can now compute the spatial sum of errors that correspond for the initial prediction 1 and 2 as shown in Eqs. 15 and 16:

$$E_{initial1} = \sum_{n \in Neighbours} w_n ||I_{initial1,n} - I_n|| \quad (15)$$

$$E_{initial2} = \sum_{n \in Neighbours} w_n ||I_{initial2,n} - I_n|| \quad (16)$$

In order to speed up the computation in Eqs. 15 and 16, a shift left operation may be performed instead of the multiplication since all weights are multiples of 2.

To get the *intermediate prediction*, we combine the *initial predictions* for this block. The weight of each initial prediction depends on the value of the sum of *absolute error* E for each prediction. The prediction with the higher *sum of absolute error* E will contribute less to the *intermediate prediction*. On the other hand, the prediction with the lower *sum of absolute error* E will contribute more to the *intermediate prediction*. Equation 17 show how we calculate the weights of each prediction using the spatial error of each:

$$\begin{aligned} w_{initial1} &= 1 - \frac{E_{initial1}}{E_{initial1} + E_{initial2}} \\ w_{initial2} &= 1 - w_{initial1} \end{aligned} \quad (17)$$

The *intermediate prediction* is the weighted combination of the 2 initial predictions. It is calculated as shown in Eq. 18:

$$I_{intermediate} = w_{initial1} \times I_{initial1} + w_{initial2} \times I_{initial2} \tag{18}$$

5 Contexts

In order to improve the compression performance, we quantize the blocks into a set of contexts based on different features such as: comparisons between the prediction to other pixels in the block, the magnitude of the gradient, the direction of the gradient, and the quantization of the average prediction error in the block. Using contexts helps us to:

- Adjust prediction through error context feedback (Sect. 5.1)
- Guess the sign of the error (Sect. 5.2)
- Choose an Encoding Bin (Sect. 5.3)

5.1 Prediction Adjustment Through Error Context Feedback

After we calculated the intermediate prediction, we adjust the prediction to end up with the final prediction. Adjusting the intermediate prediction to get the final prediction is a very important step. This step removes any redundancy in predicting pixels that belong to the same context. In other words, this step allows the algorithm to improve the quality of the prediction for each context. Equation 19 shows how we calculate the adjustment value. The adjustment value is the result of dividing the running sum of the error for a context by the running count of the pixels that belong to this context.

$$\begin{aligned} e_C &= \frac{sum(C)}{count(C)} \\ I_{final} &= I_{intermediate} + e_C \end{aligned} \tag{19}$$

5.2 Sign Guessing

When encoding an image, it is expected that the number of positive and the number of negative errors to be almost the same. For example, the total number of $+2$ errors is expected to be similar to -2. Therefore, instead of encoding the sign, we can encode our success or failure in guessing the sign. For example, when the encoder receives the error -2, it knows that the absolute error 2 and we were not successful at guessing the sign. Since both the encoder and decoder use the same method to guess the sign, the sign can be inferred.

In order to guess the sign, we collect the above features about the block and keep track of the number of positive and negative errors for each context. When encoding a pixel, we check its context, if the number of negative error is more than the positive, it is more likely that the sign of error is negative. Therefore, the error magnitude is encoded and in case the guess was not successful, a negative error is encoded.

5.3 Choosing Encoding Bin

Instead of encoding all errors as one sequence of numbers, the performance can be enhanced by grouping the pixels into different bins. Ideally, if all errors in a bin have the same values, the entropy is 0 (for the bin). Of course, this is very unlikely to happen. However, having similar errors in each bin tends to yield better compression.

In order to determine the best bin to add the error to, we examine both the spatial neighbors of the pixel and the context of the pixel.

$$Bin = round(W_{spatial} \times E_{spatial} + W_{context} \times E_{context}) \tag{20}$$

where $E_{Spatial}$ is the average absolute error of the neighbours surrounding the pixel and $E_{Context}$ is the average absolute error of the context.

In other words, the bin is a weighted combination of the spatial absolute error and the context absolute error. The weights in this equation are calculated as shown in Eqs. 21. If the weighted average of the absolute errors of the encoding block is higher than the context average absolute error, the spatial weight will be low. On the other hand, if it is lower than the context average absolute error, the spatial weight will be high. The values of the spatial weight and the context weight add up to 1.

$$\begin{aligned} W_{spatial} &= 1 - \frac{E_{Spatial}}{E_{Spatial} + E_{Context}} \\ W_{context} &= 1 - W_{Spatial} \end{aligned} \tag{21}$$

The outcome of this process is an integer that determines the bin number that we will add the error to encode. This number is a weighted combination of the spatial weighted average of the absolute error of the block and the average context error.

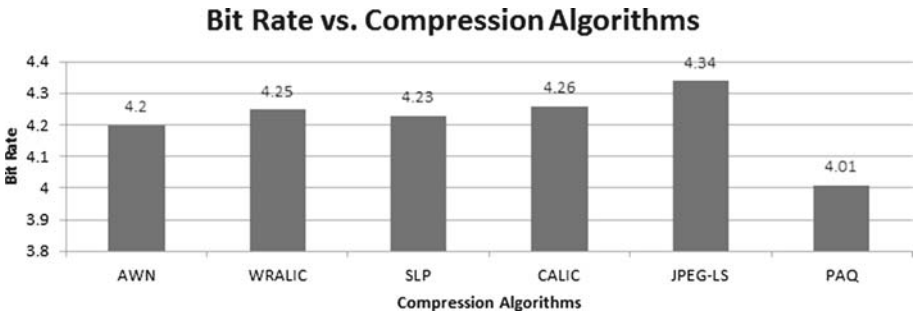

Fig. 4. This figures shows the bit rate (lower values are better) of our algorithm along with other algorithms. *AWN* achieves good results than many well known algorithms.

Table 1. Table showing the compressed size, compression rates, and encoding/decoding times (in seconds) achieved for each image in the Kodak image set. The original size for all images is 393,231 bytes.

Image name	Compressed (Bytes)	Bit rate	Encoding time	Decoding time
Kodim01	251,815	5.12	1.417	0.997
Kodim02	190,330	3.87	1.401	0.919
Kodim03	161,017	3.28	1.308	0.872
Kodim04	194,411	3.96	1.307	0.919
Kodim05	237,627	4.83	1.479	0.981
Kodim06	220,010	4.48	1.416	0.967
Kodim07	166,452	3.39	1.337	0.889
Kodim08	253,493	5.16	1.556	0.982
Kodim09	185,926	3.78	1.323	0.903
Kodim10	187,353	3.81	1.339	0.903
Kodim11	210,567	4.28	1.354	0.936
Kodim12	181,273	3.69	1.274	0.887
Kodim13	288,567	5.87	1.604	1.043
Kodim14	234,230	4.77	1.417	0.965
Kodim15	181,350	3.69	1.34	0.935
Kodim16	194,952	3.97	1.307	0.903
Kodim17	191,591	3.9	1.339	0.919
Kodim18	244,072	4.97	1.432	0.982
Kodim19	213,896	4.35	1.369	0.919
Kodim20	149,730	3.05	1.229	0.889
Kodim21	217,703	4.43	1.4	1.013
Kodim22	217,811	4.43	1.386	0.966
Kodim23	164,854	3.35	1.277	0.888
Kodim24	217,589	4.43	1.386	0.951
Average	-	4.2	1.38	0.94

6 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 a similar manner to the entropy encoding in [10], each encoding bin is further split into 2 bins. In other words, the total number of encoding bins is 32. Depending on the value of the error and the bin, the encoder may encode an escape symbol and encode the error in an extended bin. The values of the bins are: $\{5, 9, 12, 13, 15, 17, 21, 25, 29, 33, 37, 41, 46, 57, 93, 128\}$. If the absolute value

of an error being encoding is higher than the boundary, an escape symbol is encoded and the error is encoded in a separate bin.

7 Results and Experiments

We tested *AWN* on the well known Kodak image set, which is comprised of 24 gray-scale images. The total size of the images is 9437544 bytes. The overall bit rate we achieved for the whole set is 4.2, which is comparable to many state of the art algorithms.

Figure 4 shows a comparison between our algorithm with other algorithms. *AWN* achieves better results than our older proposed algorithm *WRALIC* [4]. In addition, *AWN* outperforms JPEG-LS [8] and CALIC [10]. On the other hand, PAQ [6] achieved better results than our proposed algorithm. However, because PAQ uses neural nets, the execution time is very high. Table 1 shows the compression rates and encoding/decoding times for each image in the Kodak set. As shown in Table 1, the average encoding and decoding times is 1.38 s and 0.94 s, respectively, on a machine with 2.2 GHz processor.

8 Conclusion and Future Work

We have presented a symmetric lossless image compression algorithm. The algorithm makes two initial predictions, creates a weighted combination of the initial predictions before adjusting the prediction to end up with the final prediction. In order to achieve more compression, we encode the error in multiple bins depending on the expected error magnitude. Also, instead of encoding the signed error, the algorithm attempts to guess the sign and encodes the error magnitude and whether guessing the sign was successful or not.

While developing our solution, we noticed that using multiple predictions enhances the compression rate. Therefore, we intend to build on this observation in the future and make more initial predictions. We can create a ensemble (weighted combination) of these initial predictions by using an on-line stochastic gradient descent *SGD*. The objective of the *SGD* is to give more weights to predictions that are closer to the real pixel values in a certain image.

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

References

1. Cleary, J.G., Witten, I.: Data compression using adaptive coding and partial string matching. IEEE Trans. Commun. **32**(4), 396–402 (1984)
2. Cutler, C.C.: Cutler (July 29 1952), uS Patent 2,605,361
3. Huffman, D.A., et al.: A method for the construction of minimum redundancy codes. Proc. IRE **40**(9), 1098–1101 (1952)

4. Kabani, A., El-Sakka, M.R.: Weighted ratio-based adaptive lossless image coding. In: 2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 1–6. IEEE (2014)
5. Salomon, D.: Data Compression: The Complete Reference. Springer, New York (2004)
6. Salomon, D., Motta, G.: Handbook of Data Compression. Springer Science & Business Media, London (2009)
7. Sayood, K.: Introduction to Data Compression. Newnes, Amsterdam (2012)
8. Weinberger, M.J., Seroussi, G., Sapiro, G.: The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls. IEEE Trans. Image Process. **9**(8), 1309–1324 (2000)
9. Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Commun. ACM **30**(6), 520–540 (1987)
10. Wu, X., Memon, N.: Context-based, adaptive, lossless image coding. IEEE Trans. Commun. **45**(4), 437–444 (1997)
11. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theory **23**(3), 337–343 (1977)
12. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Trans. Inf. Theory **24**(5), 530–536 (1978)